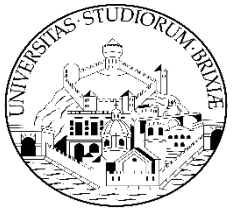


UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

ING-INF/03 - Telecomunicazioni

CICLO XXXV

An ensemble architecture for forgery detection and localization in digital images

Candidato:
Dott. Marcello Zanardelli

Supervisore:
Ch.mo Dott. Nicola Adami

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT ITALIANO

Questa tesi presenta un approccio d'insieme unificato - "ensemble" - per il rilevamento e la localizzazione di contraffazioni in immagini digitali. Il focus della ricerca è su due delle più comuni ma efficaci tecniche di contraffazione: "copy-move" e "splicing". L'architettura proposta combina una serie di metodi di rilevamento e localizzazione di manipolazioni per ottenere prestazioni migliori rispetto a metodi utilizzati in modalità "standalone". I principali contributi di questo lavoro sono elencati di seguito.

In primo luogo, nel Capitolo 1 e 2 viene presentata un'ampia rassegna dell'attuale stato dell'arte nel rilevamento di manipolazioni ("forgery"), con particolare attenzione agli approcci basati sul deep learning. Un'importante intuizione che ne deriva è la seguente: questi approcci, sebbene promettenti, non possono essere facilmente confrontati in termini di performance perché tipicamente vengono valutati su dataset personalizzati a causa della mancanza di dati annotati con precisione. Inoltre, spesso questi dati non sono resi disponibili pubblicamente.

Abbiamo poi progettato un algoritmo di rilevamento di manipolazioni copy-move basato su "keypoint", descritto nel capitolo 3. Rispetto a esistenti approcci simili, abbiamo aggiunto una fase di clustering basato su densità spaziale per filtrare le corrispondenze rumorose dei keypoint. I risultati hanno dimostrato che questo metodo funziona bene su due dataset di riferimento e supera uno dei metodi più citati in letteratura.

Nel Capitolo 4 viene proposta una nuova architettura per predirre la direzione della luce 3D in una data immagine. Questo approccio sfrutta l'idea di combinare un metodo "data-driven" con un modello di illuminazione fisica, consentendo così di ottenere prestazioni migliori. Al fine di sopperire al problema della scarsità di dati per

l'addestramento di architetture di deep learning altamente parametrizzate, in particolare per il compito di scomposizione intrinseca delle immagini, abbiamo sviluppato due algoritmi di generazione dei dati. Questi sono stati utilizzati per produrre due dataset - uno sintetico e uno di immagini reali - con lo scopo di addestrare e valutare il nostro approccio.

Il modello di stima della direzione della luce proposto è stato sfruttato in un nuovo approccio di rilevamento di manipolazioni di tipo splicing, discusso nel Capitolo 5, in cui le incoerenze nella direzione della luce tra le diverse regioni dell'immagine vengono utilizzate per evidenziare potenziali attacchi splicing.

L'approccio ensemble proposto è descritto nell'ultimo capitolo. Questo include un modulo "FusionForgery" che combina gli output dei metodi "base" proposti in precedenza e assegna un'etichetta binaria (forged vs. original). Nel caso l'immagine sia identificata come contraffatta, il nostro metodo cerca anche di specializzare ulteriormente la decisione tra attacchi splicing o copy-move. In questo secondo caso, viene eseguito anche un tentativo di ricostruire le regioni "sorgente" utilizzate nell'attacco copy-move. Le prestazioni dell'approccio proposto sono state valutate addestrandolo e testandolo su un dataset sintetico, generato da noi, comprendente sia attacchi copy-move che di tipo splicing. L'approccio ensemble supera tutti i singoli metodi "base" in termini di prestazioni, dimostrando la validità della strategia proposta.

ABSTRACT

This thesis presents a unified ensemble approach for forgery detection and localization in digital images. The focus of the research is on two of the most common but effective forgery techniques: copy-move and splicing. The ensemble architecture combines a set of forgery detection and localization methods in order to achieve improved performance with respect to standalone approaches. The main contributions of this work are listed in the following.

First, an extensive review of the current state of the art in forgery detection, with a focus on deep learning-based approaches is presented in Chapter 1 and 2. An important insight that is derived is the following: these approaches, although promising, cannot be easily compared in terms of performance because they are typically evaluated on custom datasets due to the lack of precisely annotated data. Also, they are often not publicly available.

We then designed a keypoint-based copy-move detection algorithm, which is described in Chapter 3. Compared to previous existing keypoints-based approaches, we added a density-based clustering step to filter out noisy keypoints matches. This method has been demonstrated to perform well on two benchmark datasets and outperforms one of the most cited state-of-the-art methods.

In Chapter 4 a novel architecture is proposed to predict the 3D light direction of the light in a given image. This approach leverages the idea of combining, in a data-driven method, a physical illumination model that allows for improved regression performance. In order to fill in the gap of data scarcity for training highly-parameterized deep learning architectures, especially for the task of intrinsic image decomposition, we developed two data generation algorithms that were used to produce two datasets - one synthetic and one of real images - to train and evaluate our approach.

The proposed light direction estimation model has then been employed to design a novel splicing detection approach, discussed in Chapter 5, in which light direction inconsistencies between different regions in the image are used to highlight potential splicing attacks.

The proposed ensemble scheme for forgery detection is described in the last chapter. It includes a "FusionForgery" module that combines the outputs of the different previously proposed "base" methods and assigns a binary label (forged vs. pristine) to the input image. In the case of forgery prediction, our method also tries to further specialize the decision between splicing and copy-move attacks. If the image is predicted as copy-moved, an attempt to reconstruct the source regions used in the copy-move attack is also done. The performance of the proposed approach has been assessed by training and testing it on a synthetic dataset, generated by us, comprising both copy-move and splicing attacks. The ensemble approach outperforms all of the individual "base" methods, demonstrating the validity of the proposed strategy.

Chapter 1

INTRODUCTION

The worldwide spread of smart devices, which integrate increasing quality cameras and image processing tools and apps, the ubiquity of desktop computers, and the fact that all these devices are almost permanently connected with each other and to remotely located data servers through the Internet, have given ordinary people the possibility to collect, store, and process an enormous quantity of digital visual data on a scale just until recently quite unthinkable.

As a consequence, images and videos are often shared and considered information sources in several different contexts. Indeed, a great number of everyday facts are documented through the use of smartphones, even by professionals [93]. Massive sharing of visual content is enabled by a variety of digital technologies [108], such as effective compression methods, fast networks, and specially designed user applications. These latter, in particular, include Web platforms, *e.g.*, social networks such as Instagram, and forums like Reddit, that allow the almost instantaneous spreading of user-generated images and video. On the other hand, user-friendly, advanced image editing software, both commercial like Adobe Photoshop [2], and free and open source like GIMP [5], not to mention smartphone-based apps that can apply basic image manipulations on the fly[1], are widely available to everyone.

All these factors have contributed to the spread of fake or *forged* images and videos, in which the semantic content is significantly altered. Sometimes this is done for malevolent purposes, such as political or commercial ones [129]. As of 2022, all of the major social network platforms are struggling to filter manipulated data, and so avoid

[1]For example Instagram, Snapseed, Prisma Photo Editor, Visage, and many more.

that such fake content, often directed to the most vulnerable users, could "go viral" [134]. Legal conundrums are also emerging regarding where to put the responsibility for the possibly damaging fallout of fake content spreading [52].

Such problems arise because most times humans are easily fooled by forgeries, and in some cases, they are even demonstrably not able to detect any but the less sophisticated modifications undergone by visual content, due to the so-called change blindness cognitive effect [127, 103]. Thus, there is a need for carefully designed digital techniques.

Semantic alterations can be carried out on all types of digital media content, like video or even audio. However, the focus of this thesis is on methods and algorithms specifically designed for forgery detection on *still images*, which is by far the most common case.

In this context, the general problem of determining if a given image has not been altered so as to modify its semantics is referred to as image authentication, or image integrity verification [75]. If the emphasis is put on expressly establishing if a given image has undergone a semantic alteration, or *forgery*, the same application is often referred to as *image forgery detection* in the literature [46].

In this introductory chapter, we first give a broad overview of the considered application, mainly to fix some definitions. Next, we provide a concise summary of the most commonly found types of forgery. We finally provide the organization of the remainder of the thesis and its main contributions.

1.1 Image forgery detection applications

Image forgery detection can mainly be divided into two categories: *active* and *passive*. Sometimes these methods also give a localization of the altered/forged areas of the image, and even provide an estimate of the original visual content.

Active methods for general visual content protection are based on technologies like *digital signatures* [104] and *digital watermarking* [15]. Digital signatures are

straight cryptographic methods that authenticate the bit-stream. However, the authentication in this case is fragile, meaning that any change in the bit-stream invalidates the signature, and thus it is more tailored to alternative applications such as copyright protection. This is instead not desirable when verifying image semantic content, since alterations that does not change the semantics (*e.g.*, a mild amount of compression) should be tolerated. In other words, the authentication method needs to be robust. Another serious drawback is that the signature has to be attached as metadata to the image, and therefore could be discarded or sometimes even substituted by a malicious user.

To address these shortcomings, robust methods have been proposed. For example, robust digital watermarking embeds security information in the content itself by controlled imperceptible modifications. Ideally, an attacker should not be able to alter the content of an image without changing the embedded watermark, while being able to safely apply selected processing such as compression, thus allowing the consumer of the image to detect the manipulation.

Note that variants of the aforementioned approaches exist, namely, robust signatures (based on content hashing techniques) [128, 122], and fragile watermarking [35]. Sometimes these variants have been cleverly combined [95]. However, they still inherit the same problems associated to metadata presence and fragility that we have just outlined.

In the end, active methods have the advantage of being able to convey side information which may be useful to detect the attempted forgery, but they need the watermark or signature to be computed on the unaltered version of the image, ideally at acquisition level. This in turn requires the capturing camera to have specific hardware and/or in-board post-processing software. Furthermore, any entity interested in verifying the semantic content of a given image must be able to decode the authentication information, which means having access to the (private or public) key of the creator and/or the watermark detector. However, leaving to potentially malicious users both the security information embedding and the decoding devices is usually a threat

to the entire framework[2].

As an alternative, a trusted third party could be set up to verify the image integrity, for instance, a Web site able to embed and decode the watermarks. However, scalability problems prevent such architecture to be feasible for everyday images shared on the Internet. Recently, commercial solutions based on the blockchain paradigm aimed at image integrity have also appeared to get rid of the trusted third party presence, though little details at the present time are known of their inner workings[3]. Blockchain methods can be considered active only in the sense that a block needs to be generated for each protected image, but the image itself is released without modifications. To the best of the authors' knowledge, however, these techniques are not widespread for forgery detection. That may well be because, while the distributed ledger paradigm does not need a trusted third party, fragile authentication is unavoidable since in the end blockchain has a cryptographic core, and furthermore scalability issues are still present. Still, new solutions are being proposed in this field, for instance [73].

Conversely, passive methods do not need the presence of additional data attached to the image, and they are commonly known as *forensics* [112]. Their goal is thus to tell whether an image is authentic or not by analysing only the image itself, searching for traces of specific processing undergone by the image. In the case of massively shared, ordinary images, this solution has been traditionally considered the only feasible one.

Often, an attacker can apply one or a set of successive manipulations on the target image, either on the whole image or only on a tampered region, such as a semantic alteration, *e.g.*, object duplication, JPEG compression, geometric transformations,

[2]The on-board image signature algorithm developed by Nikon, for example, has been long compromised [3]. Another high profile case is Blu-Ray, which protection scheme used a combination of cryptography, digital signatures, and watermarking [9]

[3]For example, Photo Proof Pro by Keeex [7] and Numbers Protocol [8]

up-sampling, filtering, *e.g.*, contrast enhancement, and so on. When this chain of manipulations is used by an attacker to disguise the original forgery it is referred to as *anti-forensics*.

The task of determining the history of attacks that a target image has undergone is sometimes called image *philogeny* [99]. Of course, this is a more challenging problem than simply telling apart pristine and forged images, as it involves the detection of multiple kind of attacks while also determining the order in which they were performed. Let us consider, for example, a scenario in which the attacker can perform three different manipulations, and assume for simplicity that each attack is applied at most once. The number N of possible processing histories is thus the sum of simple dispositions of k attacks from the possible three, as in:

$$N = \sum_{k=0}^{3} D_{3,k} = \sum_{k=0}^{3} \frac{3!}{(3-k)!} = 16 \tag{1.1}$$

Note that $k = 0$ means that the image is pristine. As can be observed, the number of possible histories grows exponentially with the number of available attacks. A possible solution can be found in [23, 88, 89], where the authors formulated the problem of determining the processing history as a multi-class classification problem. Therein, each of the N histories corresponds to a class, and a fusion-decision algorithm tries to combine the outputs of multiple forgery detection methods by means of an agreement function, which aims to give a higher weight to decisions on which more forgery methods agree and less to the ones on which there is less consensus.

As a final note, there is another possible forensics application, that is the trustworthy attribution of the visual content to its creator, for example, the device that generated the image. The forensics traces could be present all the way back at the acquisition level (*e.g.*, the camera-specific acquisition noise, known as Photo Response Non Uniformity [49], or PRNU) down to the post-processing stage (that is, after the original image has been stored in digital form) [75].

Sometimes, however, forgery detection follows the "in-the-wild" assumption that

Figure 1.1: An example of a copy-move forgery, taken from MICC-F600 dataset [14]. One of the front towers is a surreptitious duplicate.

the creator of a particular image is not safely attributable to any entity, and thus it is to be considered coming from a possibly anonymous, unreliable source.

1.2 Image forgery types

We now present the most common forgeries and manipulations found in the context of the just discussed applications.

1.2.1 Copy-move

The copy-move forgery is performed by copying one or more regions of an image and pasting them in the same image in different locations. Copy-move forgeries are typically used to hide information or to duplicate objects/people, thus severely altering the semantic content of the target image. An example of copy-move forgery is shown in Fig. 1.1, where the right building tower has been inserted as a copy of the left one.

1.2.2 Splicing

This forgery is similar to copy-move, with the difference that the pasted regions/objects are cut from one or more other images. A splicing forgery can be done in order to hide some content, or to show a fake situation. For example, in Fig. 1.2, we

Figure 1.2: A high-profile, in-the-wild example of a splicing forgery, that widely circulated online in 2004. The figure of Jane Fonda, captured in an unrelated 1972 photo, has been spliced into a pre-existing John Kerry picture, taken in 1970 (image taken from [20]).

can see an image in which two famous people are depicted together, but the picture has been shown to be the composition of two different images.

1.2.3 Inpainting

This kind of attack consists in filling a region or a "hole" in the image with plausible content. Inpainting is typically employed to restore damaged patches in images. However, it can also be used by potential attackers as a malicious means to hide information from an image, or to remove a visible watermark. The filled region can either be copied from another part of the image, or synthesized with a specific algorithm, such as a GAN network (Generative Adversarial Network [53], see also below). Note that, in the former instance, this attack can be thought as a particular instance of copy-move.

A particularly interesting instance of inpainting is the reconstruction of deleted parts of faces, such as the eyes or the mouth. Promising results in this regard have been obtained by *Nvidia* [92] (an example is shown in Fig. 1.3).

Inpainting techniques are also employed in video processing, usually with the aim of restoring lost regions in frames or compression artifacts. In this case, temporal correlation between frames can be exploited as well as spatial relationship between pixels to fill gaps. In [72] an object-aware, occlusion-aware approach is presented that aims to reconstruct the complete shape of the occluded objects and their appearance in the video frames. Another important contribution by the authors is the release of a large-scale benchmark dataset for video object inpainting: Youtube-VOI. In [106] a novel framework for video inpainting is proposed that adopts an internal learning strategy. Unlike previous methods that use optical flow for cross-frame context propagation to inpaint unknown regions, the authors show that this can be achieved implicitly by fitting a convolutional neural network to known regions. In [157], a two-stream encoder-decoder network involving attention modules ([141]) is designed to detect inpainted regions in videos, by using both spatial and temporal clues.

As it is strictly related to inpainting, we also mention the manipulation technique referred to as outpainting. In this case, an image or frame is extended by generating content/objects beyond the boundaries of the original image. The goal of this technique is to construct a new image that is realistic and "natural" with respect to the starting one. As for the case of inpainting, this technique is also applied to video content, where the motion of objects (estimated *e.g.*, with optical flow algorithm [61]) can sometimes be used to infer the content "outside" the frame. Some recent works about image and video outpainting can be found in [51] and [32], respectively. In the former, a transformed-based U-net architecture is proposed to perform generalized outpainting for images. In contrast to previous approaches, which mainly focused on horizontal extrapolation, this method is able to extrapolate content all-side along a given image. In the latter, a method based on background estimation and optical flow is presented to transform portrait videos (9:16 aspect ratio) into landscape videos (16:9).

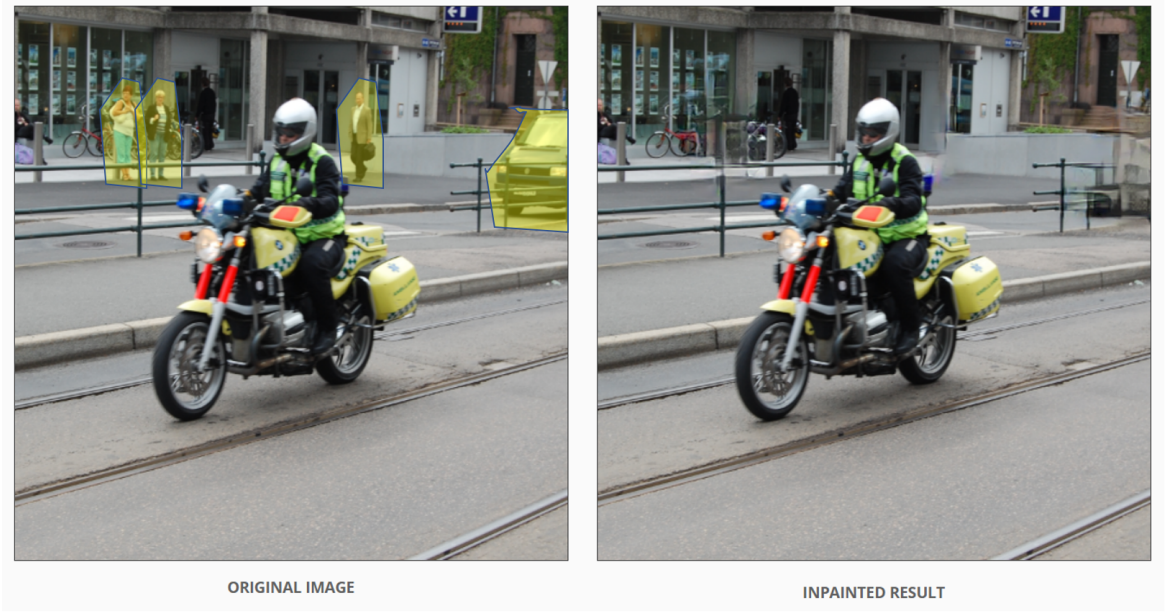


Figure 1.3: An example of inpainting generated with Nvidia interactive demo at https://www.nvidia.com/research/inpainting/index.html. The inpainting mask (*i.e.* the deleted region that must be filled by the inpainting algorithm) is shown in yellow in the left image. The original image is taken from the MICC-F2000 dataset [14]

1.2.4 DeepFake

DeepFake is a particular kind of manipulation in which a deep learning model is employed to synthesize fake content in images or videos. The "deep" term is used to emphasize the difference between the pre-Deep learning era, in which this task was manually done by experts with professional editing tools, and the current era, in which this is automatically done by deep models, such as GANs [53].

A typical application of DeepFake consists of the substitution of the face of a person with the face of another person (usually a VIP) taken from a second image or video (see Fig. 1.4). In another kind of DeepFake attack, usually referred to as *reenactment*, the facial expressions of a donor person are extracted and applied to the target person in another image or video. This is usually done by means of synthesis methods (namely, GANs) or by merging algorithms that aim to maximize the realism of the obtained face.

Figure 1.4: Examples of DeepFakes generated from different sources (image courtesy of https://www.cs.albany.edu/ lsw/celeb-deepfakeforensics.html). The first column shows the original frames, while the other five show the DeepFakes obtained with 5 different sources.

Even if most of the time DeepFakes are created for entertainment/comedy purposes, there have been cases in which a VIP was shown to be in certain situations in which he/she never was, thus damaging his/her image and leading to scandals. As a matter of fact, the vast majority of DeepFakes with the latter purpose are created in the video domain, because this kind of media usually poses a bigger semantic threat to the attacked person/VIP, especially when an appropriate audio track is available and can be matched to the facial expressions of the talking person. Furthermore, a number of easy-to-use tools have been developed to produce convincing DeepFakes, such as *FakeApp*, *faceswap-GAN*, and that available at [4]. As a consequence, many DeepFake videos have been spreading through the Web in the last few years.

DeepFakes for static images are less common, but they are still worthy of interest for forgery detection purposes. Note that this kind of attack can be thought of as a particular case of splicing.

1.2.5 CGI-generated images/videos

This approach consists in creating photo-realistic content as the rendering output of a computer graphics-generated 3D scene. Thanks to the recent advances in the video-gaming industry and in the GPU technology, techniques such as ray-tracing have been much easier to implement, thus making it possible to reach realism levels unthinkable just a few years ago (an example is shown in Fig. 1.5). In fact, in recent years a certain number of graphic engines, such as *Unity* and the *Unreal Engine*, have been developed and can be freely (or rather cheaply) used by everyone. So, more and more convincing rendered images/videos are being produced every day.

Consequently, the images generated through these engines can be almost indistinguishable from images taken with a real camera, and, of course, this can be used for malicious intents by potential attackers that can use these renderings to depict false scenes. It is worth noticing, though, that in the case of CGI generated content a certain level of expertise is still required in order to produce convincing results.

Figure 1.5: A photo-realistic CGI rendered image (created and published on *reddit* by user *u/Hary1495*

In this case, there is no clear parallels with splicing since the generated scene is generated from scratch.

1.2.6 GAN-based face synthesis

A particularly popular kind of fake content generation approach consists in the creation of a realistic face of a completely non-existing person, employing the previously cited GAN networks. This is done by feeding the trained model with a vector of random sampled noise, which is converted by the model to a realistic face different from any existing one. Again, as for the previously discussed CGI-generated content, the fake image is synthesized anew instead of being copied from another source.

In [70], *Nvidia* proposed a GAN architecture - styleGAN - that is considered a breakthrough for this technology. Interactive demos based upon this original work can also be found on the Web, such as [6]. Apart from artifacts that can sometimes still be noticeable in the background, the produced faces are really convincing and

Figure 1.6: An example of face portrait generated with styleGAN.

they are hardly detectable as fake by the naked eye. Subsequently, improved versions of the architecture in [70] were introduced: namely styleGAN2 [71] and styleGAN3 [69]. These new models improved the original one by generating better-quality images and suppressing some of the artifacts that were initially present. Also, they allow for better disentanglement during the generation process between features such as pose, background, and clothing. An example of a high-resolution face image generated with styleGAN is shown in Fig. 1.6. As can be seen, the result is remarkable from a qualitative point of view.

1.2.7 Diffusion models

Diffusion models are a class of generative models used in computer vision that can be used for tasks such as image synthesis, denoising, and inpainting. Remarkably, they have been shown to generate content even more realistic than GANs [34]. These models were inspired by diffusion processes in physics, *e.g.*, diffusion of heat in materials. These kinds of models are trained in a two steps process, involving a

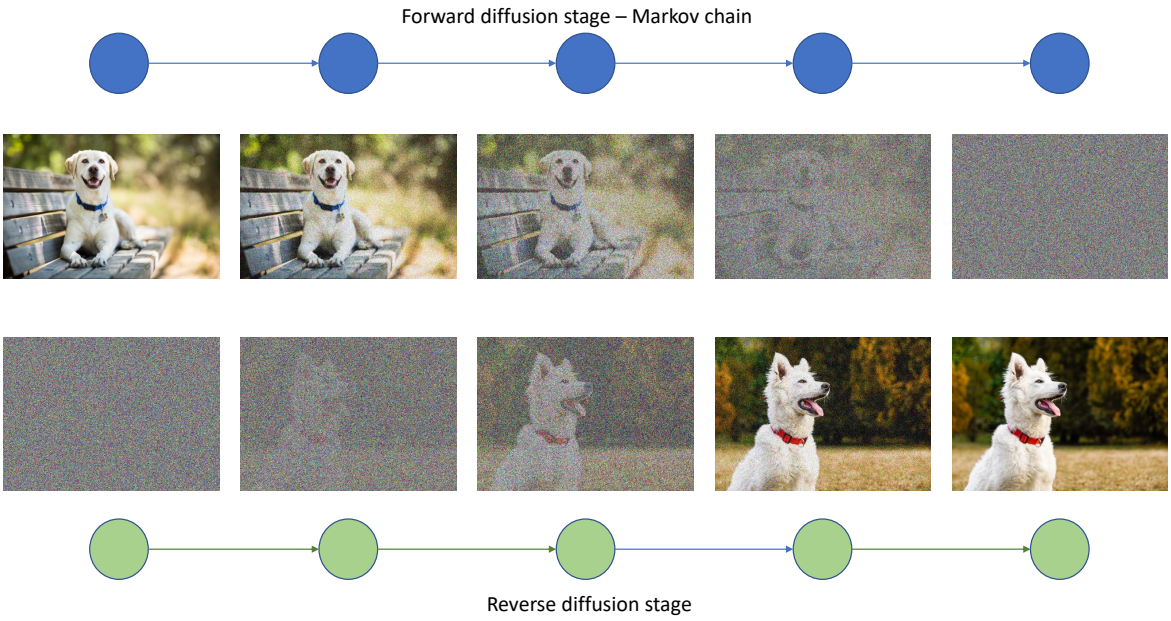


Figure 1.7: Schematic view of generative diffusion models.

forward diffusion stage and a reverse diffusion stage. In the first stage, an image is gradually degraded, *e.g.*, by adding Gaussian noise iteratively, by means of a Markov chain. Typically, the degradation stage involves a large number of iterations (even hundreds or thousands), each only slightly degrading the current image. In the second stage, the model tries to learn how to reverse the degradation process, with the goal of reconstructing the original image. This is usually done by employing a U-Net that, given the image at a certain iteration of the Markov chain, learns to predict the image at the previous step, *i.e.*, a less noisy one. Note that the assumption of the degradation process being Markov (*i.e.*, the next state only depends on the current one) is what makes the problem of reversing the degradation tractable. Once the model has been trained, the generation of an image consists in reversing the diffusion process from a completely degraded image - the input noise sample - to a clean one - the generated high-resolution image. An overview of the process is shown in Fig. 1.7.

Diffusion models were first introduced by Sohl-Dickstein et al. in 2015 [133]. However, it was not until 2020, with the publication of several works, that these models

gained significant attention and adoption. Notably, the work presented in [58] played a pivotal role in the widespread adoption of these models, in particular, for commercial use. OpenAI's paper [102] was one of the earliest works to introduce text-guided diffusion models. Among the most recent and promising products that rely on these models for image synthesis are OpenAI's DALL-E2 [4] and MidJourney [5]. These models takes as input a text prompt, which is typically a high-level description of the wanted semantic content and style of the target image that will be generated. These models are more sophisticated than the previously introduced unconditioned diffusion models, as they must also encode and process the input text and use it to condition the diffusion process itself. An example of image generated with DALL-E2 is shown in Fig. 1.8. As can be seen, not only the generated image is realistic, but also the content and style match the input prompt. In Fig. 1.9, a photo-realistic image generated with MidJourney is shown. In this case, a highly detailed text prompt was given to the model, which allowed for the synthesis of a specific scene. Note also how the handling of lighting and reflections in the glass is quite convincing.

1.3 Thesis organization

The thesis' contents and contributions are organized as follows.

Chapter 2 is a comprehensive review of the most recent state-of-the-art forgery detection methods, with a focus on deep learning-based approaches. The authors found that the current state of the art in forgery detection faces a major challenge in terms of performance comparison due to the use of custom, non-public datasets by many authors.

Chapter 3 presents a novel keypoint-based copy-move detection algorithm that outperforms a state-of-the-art method by incorporating a density-based clustering step to filter out noisy keypoint matches.

[4] https://openai.com/product/dall-e-2

[5] https://www.midjourney.com/home/

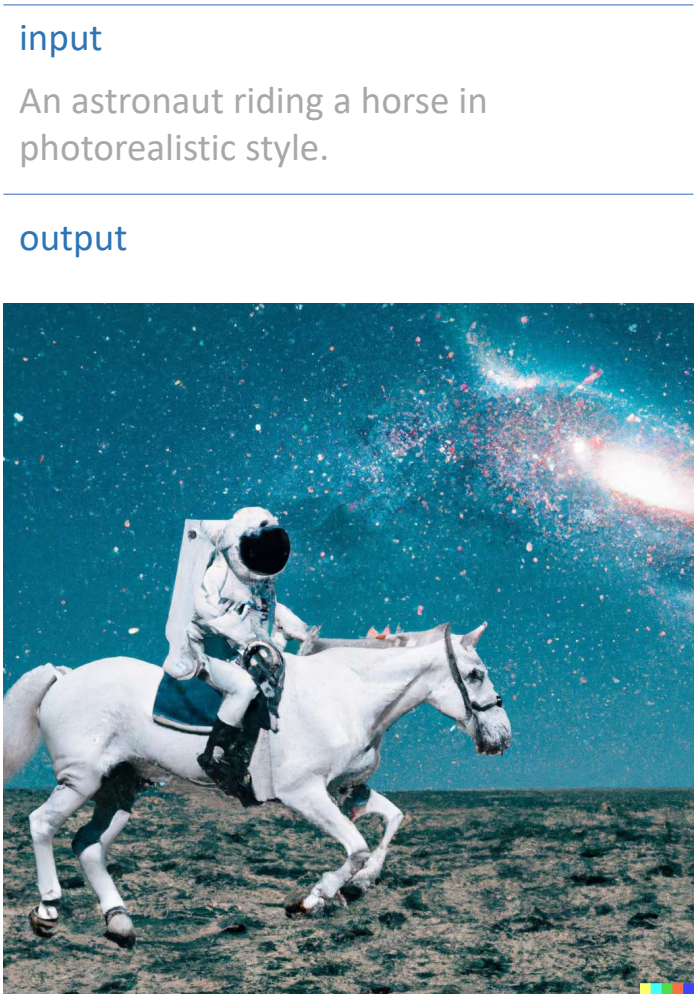


Figure 1.8: Example of text prompt and corresponding image generated with DALL-E2.

Chapters 4 and 5 explore the concept of forgery detection based on the light properties of the target image. In particular, Chapter 4 introduces a novel deep learning architecture for predicting the 3D light direction in an image. To overcome the issue of limited annotated data, two datasets were generated - one synthetic and one real - to

input

photography shot trough an outdoor window of a coffee shop with neon sign lighting, window glares and reflections, depth of field, grandpa in a suit with a cup of coffee in his hands sitting at a table, portrait, kodak portra 800, 105 mm f1. 8

output

Figure 1.9: Example of text prompt and corresponding image generated with MidJourney V.5.

train and evaluate the model. The light direction prediction model is then employed to design a novel splicing detection approach in Chapter 5.

Chapter 6 describes an ensemble scheme that combines the outputs of the previous methods and outperforms them in terms of detection performance. The approach also tries to predict the type of forgery (splicing or copy-move), and in the case of copy-move attacks, it also tries to reconstruct the source regions used in the attack. The proposed approach is trained and tested on a synthetic dataset that includes both copy-move and splicing attacks, demonstrating its effectiveness.

Finally Section 6.4 draws some conclusions and highlights some possible future directions ad challenges.

Chapter 2

DEEP LEARNING BASED FORGERY DETECTION METHODS - A PERFORMANCE COMPARISON

Since the early 2000s, a lot of approaches to image forgery detection have been proposed, and many excellent reviews can be found [75, 46, 63, 143, 117, 20]. However, deep learning techniques have proved to be a game-changer in many digital processing and computer vision applications, especially when a lot of training data are available [83, 155, 91]. Even if in the case of forgery detection this last assumption is not quite satisfied, nonetheless, as discussed in what follows, the best performance on standard benchmarks were obtained with algorithms that leverage Deep Learning (DL) models in one or more phases.

For this reason, we feel that it is very important to keep track of the breakthroughs made possible by deep learning in forgery detection. In particular, it is crucial that some degree of comparison between DL-based techniques that follow different perspectives is carried out. This is especially true since it is challenging to identify future (and even present) trends in a technology like DL, which is already vast and still expanding at a tremendous rate.

In this analysis, we mainly focus our discussion on copy-move and splicing detection methods. Even if these attacks are not as recent as GAN-based ones or DeepFakes, they are very prominent in the literature and lots of algorithms for their detection are still being published to date. These forgeries are so diffused mainly because of their simplicity, both related to end-user employment and experimental dataset building, but also because they are a very immediate threat to the image semantics integrity.

Even so, we discuss some of the DeepFake detection techniques, insofar as this kind of attacks can be seen of a special (and more sophisticated) case of splicing, or at

least a manipulation that usually involves a source or donor image/video and a target one. However, since this work aims to give an overview on *image* forgery detection methods, we do not deal with approaches specifically designed for video content, *i.e.*, that cannot be applied to single images. In fact, lots of video-specific methods leverage temporal clues between different frames or, if available, inconsistencies between the audio track and the facial expressions. Also, as frames in a video are often strongly compressed, even techniques that operate on a per-frame level can hardly be applied to high-quality images. We refer the reader interested in DeepFakes seen as a standalone researc h field to the review in [142].

As stated before, this analysis focuses on the most recent methods for copy-move and splicing detection that are specifically based upon DL. To better highlight the contrast with the previous state-of-the-art, it is useful to first recap in Section 2.1 several of the established forensics-based techniques for image forgery detection that instead follow traditional approaches. Then, we describe the key-aspects of the deep learning based methods, including their applicability and their limitations, and we illustrate their properties such as the kind of attacks they can detect and whether they give or not the localization of the forged areas. We concurrently discuss the datasets on which they were trained/tested. Then, in Section 2.8 we discuss their performance, which are also directly compared when possible (that is, tested on the same benchmark dataset). Finally, in Section 2.9 we follow up on the previous discussion by drawing some conclusions, while providing some insights on what we think should be the most important future research directions.

2.1 Traditional passive forgery detection methods

We now briefly discuss some of the "conventional" passive image forgery detection approaches that have been proposed since the early 2000s. Of course, what we present here is not an exhaustive, nor in-depth review of these methods. For a more comprehensive review, see [46], [63], and [143].

Conventional passive methods leverage techniques from the fields of signal processing, statistics, physics, and geometry, and are usually also referred to as "classic" or "traditional" approaches. In fact, they come from the pre-DL era that we are currently in and, as such, they require little or no data to perform an eventual training phase. Those that still require data for training are typically based on traditional machine learning techniques, such as clustering, support vector machines (SVM), linear/logistic regression, random forests, and so on. Here, we still consider those as belonging to the classic methods, because they rely on models that have a relatively small number of parameters, and therefore do not require a great amount of data for training.

We think it is useful to briefly describe some of the traditional approaches, for the following two reasons:

1. As mentioned above, they typically do not require much data for training (or none, even). Of course, this is an advantage when it is hard or impossible to collect a good amount of labelled images to train a high parameterized deep learning model. Also, most of these methods are not as computationally expensive, and thus can be easily deployed on commercial low-power hardware, like smartphones or tablets;

2. Some of the core ideas and principles these methods rely on can also be used in conjunction with deep learning models, in order to accelerate the training phase or to achieve better performance. For example, in [120], a SVM model is used as final classification phase applied on the output of a CNN. In [118], a YCbCr color space conversion and a DCT transform are used as pre-processing stages before a CNN. In [136], a CNN takes as input the Laplacian filter residuals (LFR) computed on the input images rather than the images themselves. All of these methods, among several others, are discussed in detail in Chapter 2.

Passive traditional methods can be usually grouped into five main categories. We discuss each separately in the remainder of this Section.

2.1.1 Pixel based

These methods rely on the fact that certain manipulations introduce anomalies that can affect the statistical content of the image at the pixels level. Some of these anomalies can be detected in the spatial domain, while others in the frequency domain or in a combination of both.

For copy-move attacks, it is common to observe a strong correlation between copied regions in the image but, due to the fact that these can be of any size and shape, it is computationally infeasible to explore the whole space of possible shape/size combinations. The authors of [50] have proposed a method based on the Discrete Cosine Transform (DCT). In particular, they divided the image into overlapping blocks and applied a DCT on each block. The DCT coefficients were used as feature vectors that describe each block. Duplicated regions then were detected by lexicograpycally ordering the DCT block coefficients and grouping the most similar ones. Another approach, proposed in [114], consisted in applying a Principal Component Analysis (PCA) on image blocks' features, and then comparing blocks representation in this reduced-dimension space. These approaches have been shown to be robust when minor variations in the copied regions are performed, like additive noise or lossy compression. However, in general these methods do not perform well in the case of geometric transformations like rotation or scaling.

Thus, let us consider now a situation in which a geometric transformation is used in order to make a copy-move attack more convincing. Geometric transformations usually involve some form of interpolation between neighbouring pixels, in particular, the most common techniques are bilinear or cubic interpolation. Depending on the chosen technique, a specific correlation pattern between these pixels is created. Statistical methods are then employed with the aim of finding these patterns in order to detect regions in which a geometric manipulation has been employed. An example of this approach is described in [113].

An example of frequency-based forgery detection is [45]. To detect spliced regions, the authors observed that, even if the boundary between the spliced region

and the original image can be visually imperceptible, high-order Fourier statistics are affected by this kind of manipulation and thus can be used for detection.

Another common type of methods, specifically designed for copy-move attacks detection, are the key-point based methods. They typically require the following steps:

1. Key-points extraction. Key-points are variously defined as "points of interest" of the image, for example, local minima or maxima, corners, blobs, *etc.* Some of the most commonly employed key-points extraction processes include the well-known Scale Invariant Feature Transform (SIFT) [94], Speeded Up Robust Features (SURF) [18], or Features from Accelerated Segment Test (FAST) [123];

2. Descriptors extraction. One or more feature vectors (descriptors) are extracted from each key-point. Usually, these vectors are a compact description of the region in the vicinity of the key-point. In addition to the SIFT/SURF feature values, Histogram of Gradients (HOG) and the FAST-based ORB [123] are other common ones;

3. Descriptors matching. In this step, descriptors are compared according to a distance (or a complementary similarity) function. If the distance of two or more descriptors is below a certain threshold, a match between these descriptors is declared;

4. Filtering step. In this phase, some form of filtering of the matching results is done in order to rule out weak matches. This can be done by different criteria, such as Lowe's ratio [94], in which a match is considered valid only if the distance between the two most similar descriptors is considerably smaller than that between the two next-best ones. Other criteria can be employed, for instance, based on the spatial relationship between the key-points.

One of the most cited key-point based methods for copy-move detection was proposed by Amerini *et al.* in [14]. The authors showed that these methods are quite robust even against rotation and scaling, but the performance are not as good when the copy-moved

regions are too uniform. In fact, in this case only few key-points can be extracted, and consequently the matching phase provides weak results.

2.1.2 Format based

Usually, images captured by a digital camera are encoded in JPEG format. This means that the image is divided into 8×8 pixel blocks, which are then DCT transformed and quantized. As a consequence, specific artefacts are generated at the border of neighbouring blocks. The authors of [96] observed that image manipulations like copy-move or splicing result in alterations in the JPEG artefact pattern, and proposed a method in which they used a sample region (which is supposed authentic) of the target image to estimate the JPEG quantization table. Then, they divided the image into blocks, and a "blocking artefact" measure is computed for each block. A block is considered tampered if the score given by this measure is sufficiently distant from the average value on the whole image.

Obviously, a key limitation of these methods is that they are based on specific assumptions on the format of the stored image (*e.g.*, JPEG), and therefore they are not universally applicable.

2.1.3 Camera based

The basic idea exploited by these methods is that every digital camera leaves a particular "footprint" or "signature" on each image they generate. This fact can also be useful to tie an image to a specific capturing device. In [49], the authors used a set of images taken by a known camera to estimate the parameters of the already mentioned PRNU, which is a camera specific multiplicative term that models the result of in-camera processing operations. These PRNU parameters are also extracted from the target image, which is supposed to be taken with the same camera, and compared with the previously estimated ones. The idea is that, if a splicing operation from a different camera type has been made, this results in a discrepancy between the estimated parameters.

One of the obvious limitations of this method is that it is camera-specific: this means that a different training set of images must be used for each type of camera in order to build its specific PRNU model. Also, this method is effective just for those splicing attacks in which the spliced region is extracted from a source image taken with a different camera with respect to the one used to acquire the target image, which is not always the case.

The authors of [65], instead, leveraged chromatic aberration to detect image forgeries. The phenomenon of chromatic aberration arises from the fact that photographic lenses are not able to focus light of different wavelengths on the same point on the camera sensor. In fact, from Snell's Law, the refraction index of a material depends on the wavelength of the incident light too. As a consequence, each point of the physical scene is mapped, in the RGB color channels, into points that are spatially slightly shifted one from another.

So, the authors of [65] built a model that approximates the effect of the chromatic aberration and estimated its parameters. Forged regions usually show inconsistencies with the estimated model, and can thus be detected. In this case as well, the main drawback is that this method is camera-specific. In fact, different cameras have different chromatic aberration levels (that typically depend on the kind of lenses), and consequently it is hard to set a specific threshold for the anomalies detection, if the camera from which the target image was taken is not known *a priori*.

2.1.4 Lighting based

Typically, when an attacker performs a copy-move or splicing attack, it is hard to ensure that the lighting conditions of the forged region is consistent with that of surrounding image. Compensating for this effect can be hard even using professional software like Adobe Photoshop. Therefore, the basic idea of lighting (or physics) based techniques is to build a global lighting model from the target image, and then to find local inconsistencies with the model as evidence of forgery.

Different lighting models were proposed, such as those in [64] and in [68], for which least squares error approaches are usually employed for parameters estimation. Techniques like Random Sample Consensus (RANSAC) [48] are sometimes used in order to make the model more robust to outliers. The positive aspect of these methods is their wide applicability. In fact, they are not based on assumptions on the type of camera that generated the image, and they can be used to detect both copy-move and splicing attacks. However, a downside of these methods is the fact that they are dependent on the physical context present in the image. In particular, if the lighting conditions are quite complex (for example, an indoor scene), a global lighting model cannot be estimated, and thus the method cannot be applied.

2.1.5 Geometry based

Geometry-based methods rely on the fact that a copy-move or a splicing attack usually results in some anomalies in the geometric properties of the 3D scene from which the image is obtained.

The authors of [67] proposed a method to estimate the so-called principal point through the analysis of known planar objects, and observed that this is usually near the center of the image. They also showed that a translation of an object in the image plane results in a shift of the principal point, and thus this fact can be used as evidence of forgery.

Another interesting approach was proposed in [66]. The idea was to consider specific known objects such as billboard signs or license plates and make them planar through a perspective transformation. Once the reference objects are viewed in a convenient plane, it is possible, through a camera calibration, to make real world measurements, which can then be used to make considerations on the authenticity of the objects in the image.

Of course, these methods are based on strong assumptions on the geometry of the 3D scene. They also require a human knowledge of the real world measures taken from specific objects in the image. Consequently, their applicability is quite limited.

2.2 Deep Learning based methods

Deep learning methods have gained a huge popularity over the past decade, and indeed they have been applied to a great variety of scientific problems. This is due to the fact that they were shown to perform particularly well for classification problems, as well as regression and segmentation ones. For certain tasks, these methods can even outperform humans in terms of accuracy and precision. Another crucial factor that contributed to the spread of deep learning techniques is that, in contrast to conventional machine learning approaches, they do not require the researcher to manually create (craft) meaningful features to be used as input to the learning algorithm, which is often a hard task that requires domain-specific knowledge. Deep learning models, such as Convolution Neural Networks (CNN), are in fact capable of automatically extract descriptive features which capture those facets of the input data that are well tailored to the task at hand.

For image forgery detection too, deep learning techniques have been explored in the recent literature in order to achieve better accuracy than previously proposed, traditional methods. The techniques that we are considering can be grouped in distinct categories according to different criteria, in this case:

A) Type of detected forgery: copy-move, splicing, or both;

B) Localization property, *i.e.*, if the considered algorithm is able to localize the forged areas. In the case of copy-move detection, an additional question is whether the algorithm is able to distinguish between the source region and the target one, *i.e.*, the region on which the source patch is pasted. This property is useful, for example, in a scenario in which a forensic expert is asked to analyze a tampered image in order to interpret the semantic meaning of a copy-move attack;

C) Architecture type, that is, the algorithm is an end-to-end trainable solution, *i.e.*, without parameters that need manual tweaking, or not.

As discussed in Section 1.2, DeepFakes can be regarded as a particular case of splicing attack. However, given the fact that the vast majority of DeepFake forgeries

involve face manipulations, methods that aim to detect these attacks can leverage domain-specific knowledge (*e.g.*, face detection algorithms) that cannot be used by generic splicing detection algorithms. As such, different datasets need to be used for evaluating and comparing these methods. Therefore, DeepFake forgery detection performance cannot presently be directly compared with generic splicing detection algorithms. Consequently, in this analysis, the discussion on the former methods is conducted separately, both in regard to employed datasets and experimental results.

For our analysis, we have selected some papers among the most recent ones that we think are particularly representative of those that can be categorized into at least one of the distinct groups that we have outlined above. A further principle that we have used for this selection is performance driven, with the added objective of being able to do a meaningful comparison (when possible), given in Section 2.8. These papers are described in some detail in this Section, with the further objective of identifying if any trend in the DL overall architecture choice is emerging.

In particular, we have used the criteria A) and B) above to sort the presentation order of the papers. Methods [13], [41], [11], [107], and [146] are copy-move-only specific, and are presented first in Section 2.5. Then, methods [120], [156], [118], [97], [136], [146], [36], and [29], that are for both splicing and copy-move detection, are discussed next in Section 2.6.

Besides this first separation through criterion A), we sort the techniques in each subset using criterion B), namely, [13], [11], and [146] in the first subset possess the localization property and are discussed first. For the second subset, such property is verified by [156], [118], and [29], which are thus described before the others. Note that methods [11] and [146] are also able to distinguish the source from the target regions.

Regarding criterion C), which is not used for sorting the methods, we remark here that end-to-end architectures can be found in [146], [97], [41], and [107]. The reader is referred to Table 2.5 for a summary of the characteristics of the described techniques.

Finally, DeepFake specific methods are discussed in Section 2.7.

For each described method, we also discuss:

- which datasets, whether public benchmark or custom ones, were used for the experimental validation;
- the performance on one or more of the above datasets: metrics like accuracy, precision, localization accuracy, *etc.*.

Therefore, before diving into a detailed overview of the deep learning based approaches, we proceed to first briefly describe in Section 2.3 some of the benchmark datasets that are typically used in the most recent literature for evaluation of the considered forgery detection methods, and summarize the employed performance metrics.

Finally, we mention that there are several other interesting works that involve deep learning as a means for forgery detection, which are however not analyzed here because their characteristics are a mixture of the representative works that we have selected. Some examples are [100] and [145]. In the former, a copy-move-only method is presented that leverages a pre-trained AlexNet (on ImageNet) as a block feature extractor and a subsequent feature matching step that allows to localize the copy-moved regions. In [145], instead, a technique for both copy-move and splicing detection is discussed, which is built upon the formulation of the forgery detection and localization task as a local anomaly detection problem. In particular, a "Z-score" feature is designed that describes the local anomaly level and is used in conjunction with a LSTM (long short term memory) model that is trained to assess local anomalies. Note that both of these methods satisfy criterion B), *i.e.*, they give the localization of the forged areas.

As a further remark regarding the property of being able to distinguish between source and target regions, we refer the reader to the recently published work in [16], in which a DL-based method is presented as a post-processing phase to distinguish between source and target regions, starting from the localization mask of any copy-move forgery detection technique.

Table 2.1: Benchmark copy-move/splicing datasets overview.

Dataset	Ref.	Manipulations	pristine/forged	Resolution	Format
CASIA1	[38]	copy-move, splicing	750/975	384 × 256	JPG
CASIA2	[38]	copy-move, splicing	7491/5123	320 × 240 – 800 × 600	JPG, BMP, TIF
DVMM	[140]	splicing	933/912	128 × 128	BMP (grayscale)
MICC-F220	[14]	copy-move	110/110	480 × 722 – 1070 × 800	JPG
MICC-F600	[14]	copy-move	440/160	722 × 480 – 800 × 600	JPG, PNG
MICC-F2000	[14]	copy-move	1300/700	2048 × 1536	JPG
SATs-130	[27]	copy-move	10/120	various	JPG
CMFD	[28]	copy-move	0/48	various	JPG, PNG
CoMoFoD	[139]	copy-move	4800/4800	various	JPG, PNG
DS0-1	[31]	splicing	100/100	2048 × 1536	PNG
Korus	[76]	copy-move, splicing	220/220	1920 × 1080	TIF
DFDC	[37]	DeepFake	1131/4113 (videos)	various	MP4
FaceForensic++	[125]	DeepFake	1000/1000 (videos)	various	MP4
Celeb-DF	[87]	DeepFake	590/5639 (videos)	various	MP4
VIPCup 2022 Test1/2	[1]	general AI-synthesis	2500/2500	various	JPG, PNG, TIF
TrueFace	[21]	Face synthesis	100k/110k	1024 × 1024 / 720 × 720	JPG, PNG
R-SMUD	[116]	Social media up/download	900/-	various	JPG (6 comp. levels)
V-SMUD	[116]	Social media up/download	512/-	various	JPG

2.3 Datasets description

We now provide a list of the benchmark datasets used by a majority of the discussed copy-move, splicing, and DeepFake detection methods. For completeness, we also mention some datasets related to more general AI-synthesis detection and for tracking the social media sharing of images. In fact, most of the deep learning methods that are presented in what follows are trained and/or tested on either one of these datasets or a custom one built upon the datasets themselves. The main characteristics of each dataset are summarized in Table 2.1. Evaluation metrics are discussed next in Section 2.4.

2.3.1 CASIA v1.0 (CASIA1)

[38]: it contains 1725 color images with resolution of 384×256 pixels in JPEG format. Of these, 975 images are forged while the rest are original. It contains both copy-move and splicing attacks;

2.3.2 CASIA v2.0 (CASIA2)

[38]: it contains 7491 authentic and 5123 forged color images with different sizes. The image formats comprise JPEG, BMP, and TIFF. This dataset is more challenging than CASIA1 because the boundary regions of the forged areas are post-processed in order to make the detection more difficult. It contains both copy-move and splicing attacks;

2.3.3 DVMM

[140]: it is made of 933 authentic and 912 spliced uncompressed grayscale images in BMP format, with fixed size of 128×128;

2.3.4 MICC-F220

[14]: it is composed by 110 copy-moved and 110 original JPEG color images. Different kinds of post-processing are also performed on the copied patches, such as rotation, scaling, and noise addition;

2.3.5 MICC-F600

[14]: it contains 440 original and 160 tampered color images in JPEG and PNG formats. The tampered images involve multiple copy-moved regions, which are also rotated. The image sizes vary between 722×480 and 800×600 pixels;

2.3.6 MICC-F2000

[14]: it consists of 700 copy-moved and 1300 original JPEG images, each one with a resolution of 2048×1536 pixels;

2.3.7 SATs-130

[27]: it contains 130 images, generated by 10 source authentic images, with copy-moved regions of different sizes. Various JPEG compression levels are applied, therefore the images are stored in JPEG format;

2.3.8 CMFD

[28]: it is composed of 48 source images in which a total of 87 regions (referred by the authors as "snippets"), with different sizes and content (from smooth areas, *e.g.*, the sky, to rough ones, *e.g.*, rocks, to human-made, *e.g.*, buildings) are manually selected and copy-moved. The authors also provide a software that allows to apply different post-processing steps on the forged images in a controlled way. The images are given in JPEG and PNG formats;

2.3.9 CoMoFoD

[139]: this dataset contains 4800 original and 4800 forged images, with copy-move attacks and post-processing operations such as JPEG compression, noise adding, blurring, contrast adjustment, and brightness change. The images are stored in PNG and JPEG formats;

2.3.10 DS0-1

[31]: it contains 200 images, 100 of which are pristine and 100 are forged with splicing attacks. All the images are in PNG format at a resolution of 2048×1536 pixels. Color and contrast adjustment operations are applied as counter-forensic measures;

2.3.11 Korus

[76, 74]: this dataset is composed of 220 pristine and 220 forged RGB images in TIFF format. The dataset contains both copy-move and splicing attacks, performed by hand with professional editing software. The resolution of the images is of 1920×1080.

2.3.12 DFDC (DeepFake Detection Challenge on Kaggle)

[37]: it contains 4113 DeepFakes videos created from a set of 1131 original ones, involving 66 subjects from various ethnicity and both genders. The video resolution varies from 180p to 2160p. All the videos are in MP4 format and the employed codec is H.264;

2.3.13 FaceForensic++

[125]: it is an extension of the previous dataset FaceForensic, with a total of 1.8 millions images created with 4 different DeepFake state-of-art generation methods (*DeepFakes* [4], *Face2Face* [138], *FaceSwap* [77], and *NeuralTexture* [137]), starting from 4000 videos downloaded from *YouTube*. Compared to other previously proposed datasets, it is bigger by at least one order of magnitude. The dataset contains videos of different sizes, such as 480p, 720p, and 1080p. The videos are in MP4 format, and the codec used is again H.264.

2.3.14 Celeb-DF

[87]: the authors of this dataset specifically created it in order to overcome the lack of realism of a large portion of DeepFake videos in previously published datasets (such as the original FaceForensic). It comprises a total of 5639 DeepFake videos and 590 pristine videos in MPEG4.0 format (H.264 coded), with different resolutions and a standard frame rate of 30 fps. The average length is about 13 seconds (corresponding to a total of more than 2 million frames). Another feature that sets this dataset apart from previously proposed ones is how it includes a pronounced variety of ethnicity and equilibrium among genders.

2.3.15 TrueFace

TrueFace [21] is a dataset containing real and synthetic human faces generated by StyleGAN and StyleGAN2 generative models and shared on three popular social networks (Facebook, Telegram, and Twitter), for a total of 210k images at a resolution of 1024×1024 or 720×720.

2.3.16 VIPCup2022 dataset

The IEEE Video and Image Processing Cup [1] is a competition organized by the University of Napoli and Nvidia to develop methods for detecting AI-generated content in images. In this competition, two test sets were used to evaluate the performance of

the candidate methods. The first one comprised 2500 real images (selected from FFHQ [70], Imagenet [33], COCO [90], and LSUN [148] datasets) and 2500 AI-generated with different techniques (such as StyleGAN2 [71], StyleGAN3 [69], Inpainting with Gated Convolution [149], GLIDE [102], and Taming Transformers [42]). The second test set was also composed of 2500 real images and 2500 fake ones, but in this case, the techniques used to generate the synthetic images are not disclosed.

2.3.17 V-SMUD and R-SMUD

The R-SMUD (RAISE Social Multiple Up-Download) and V-SMUD (VISION Social Multiple Up-Download) [116] are two large-scale datasets designed to aid research on tracking the social network origin of online images. Both datasets consist of images shared up to three times through three social network platforms: Facebook, Flickr, and Twitter. The R-SMUD dataset contains 50 RAW images extracted from the RAISE dataset [30], which were cropped to three different sizes and compressed under six quality factors. The V-SMUD dataset contains 510 JPEG images extracted from the VISION dataset [130], with 15 images selected from each of the 34 cameras (excluding one camera due to exceeding the size limit).

2.4 Evaluation metrics

Performance metrics in the considered forgery detection applications are the same used for binary classification problems. There are two classes, authentic or forged, that can be attributed either to the whole image or at the pixel level (through appropriate masks).

Table 2.2 recaps the terminology for binary classification evaluation using the so-called confusion matrix. Starting from ground-truth classes and the labels output by the detection system, the 4 outcomes given as TP, FP, TN, and FN can be counted according to the concordance or discordance of the labels with the corresponding classes.

The sum of every element in Table 2.2 is equal to the total number of queries T, namely the population (or the number of objects in the ground-truth). Among these

T queries, P have a positive ground-truth class and N have a negative ground-truth class, therefore $T = P + N$. In forgery detection, as in many other binary classification problems, each element in Table 2.2 is suitably divided by P or N, and thus express the corresponding fraction, or rate, as follows:

$$\begin{aligned} TPR &= TP/P & FPR &= FP/N \\ FNR &= FN/P & TNR &= TN/N \end{aligned} \tag{2.1}$$

Please note that in some papers the R (rate) part can be omitted, however, there is no possible confusion as the given number is in the [0,1] interval. Given the outcomes in Table 2.2 and the rates in Eq. (2.1), additional metrics can be obtained as follows:

$$\begin{aligned} \text{precision} &= \frac{TP}{TP+FP} \\ \text{recall} &= TPR = \frac{TP}{TP+FN} \\ \text{accuracy} &= \frac{TP+TN}{T} \\ \text{balanced accuracy} &= \frac{TPR+TNR}{2} \\ \text{F}_1\text{ score} &= 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \end{aligned} \tag{2.2}$$

An additional metric is the AUC (Area Under the ROC curve). The AUC is the two-dimensional area under the whole Receiver Operating Characteristic (ROC) curve, that plots FPR versus TPR varying the decision threshold of the detection algorithm.

These measures, or slight variations thereof, are extensively used in the papers described in what follows. There are commonly used synonyms for some of them, for example, the false alarm rate or fallout is the same as FPR and sensitivity is a synonym for recall. Such occurrences have been adjusted for clarity's sake.

2.5 Copy-move specific methods

According to the grouping and sorting criteria of the DL-based techniques discussed in this work, we begin in this Section by introducing copy-move only forgery

Table 2.2: Confusion matrix and outcomes.

		Ground-truth classes	
		Positive (P)	Negative (N)
Output labels	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

detection methods.

2.5.1 R. Agarwal et al.

The authors of [13] proposed a method specific for copy-move detection that uses deep learning in conjunction with a segmentation step and further feature extraction phases. First, the $M \times N$ input image is segmented with the Simple Linear Iterative Clustering (SLIC) procedure [12]. In order to do so, a 5-D feature vector is built for each pixel, by concatenating its RGB color values and spatial x, y coordinates. A clustering is then performed on these features, and the segmented patches (referred to as "super pixels") are given as output.

Then, multi-scale features are extracted from each super-pixel S_k with a VGGNet [132] network. This process involves the following steps:

- Given the segmented image, a binary mask BM for each super-pixel is obtained as:

$$BM_k(i,j) = \begin{cases} 1 & \text{if pixel } (i,j) \in S_k, \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

- Let $f \in \mathbb{R}^{M' \times N' \times D}$ be the output of the first convolutional layer, where M', N' are the spatial dimensions, and D is the number of output channels. $RF(l,m)$ denotes the receptive field on the input image in the (l,m) position. A continuous value mask $MConv_k \in \mathbb{R}^{M' \times N'}$ is then computed as follows:

$$MConv_k(l,m) = \frac{1}{|RF(l,m)|} \sum_{(u,v) \in RF(l,m)} BM_k(u,v). \tag{2.4}$$

The super-pixel-level feature map g_k is obtained by multiplying the output of the convolutional layer with the mask:

$$g_k(l, m, c) = f(l, m, c) \cdot MConv_k(l, m), \quad c = 1, .., D \tag{2.5}$$

- The previous steps are repeated for each convolutional stage of the VGGnet. By using Max-pooling after each convolutional layer, increasingly high-level features are extracted for each super-pixel (see Fig. 2.1).

Next, a "relocation" phase of the higher-levels features (with lower spatial resolution) is employed in order to find a pixel-level position of the features themselves in the input image. In this way, a set of key-points, with the corresponding multi-level features, is obtained for each patch.

Finally, a key-points matching phase is performed by comparing their associated features, and the copy-moved patches are identified by a further comparison of the super-pixels to which the key-points belong. This procedure is referred to as ADM (Adaptive Patch Matching) by the authors.

The VGGnet is trained on the MICC-F220 dataset. The same dataset is used for testing, though it is not specified which portion of it is used for training and which one is used for testing. The metrics used for evaluation are TNR, FNR, FPR, precision, TPR (recall), and accuracy. The reported results are:

- TNR: 97.1 %;
- FNR: 9.2 %;
- FPR: 55 %;
- Precision: 98 %;
- TPR: 89 %;
- Accuracy: 95 %.

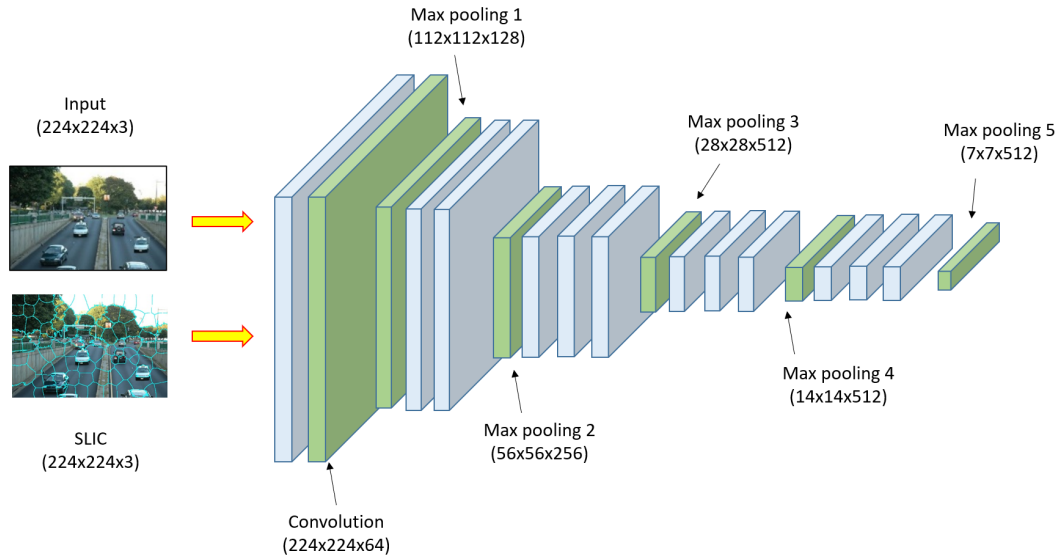


Figure 2.1: In [13] the super-pixel segmentation map is given, along with the target image, as input to a VGGNet. Features at different levels are extracted for each of the input super-pixels. Finally, high-level features undergo a so called "relocation phase" to obtain a localization mask at the original resolution.

Therefore, the reported accuracy of the method is high, but at the cost of a large number of false positives.

Also, it should be noted that the reported performance is relative to the MICC-F220 dataset, that only has 220 images, with a limited number of types of copy-move attacks. For these reasons, results obtained on just this dataset are not as statistically relevant as methods tested on other, more populated copy-move datasets, such as MICC-F2000 or CoMoFoD.

2.5.2 Y. Abdalla et al.

The authors of [11] proposed a 3-branches method for copy-move detection. An overview of the considered architecture is shown in Fig. 2.2, which is in the end based on a GAN model. To recap, the GAN is composed of two different deep learning modules: the Generator (G) and the Discriminator (D).

- The generator is a Unet that takes as input an image I and gives as output a forged version of the image itself $I' = G(I)$;
- The discriminator is a CNN network that takes as input either an original image I or a generated image $I' = G(I)$. The output is a binary mask, in which each pixel is labelled as either authentic or forged.

The purpose of D is to discriminate between original pixels and pixels that were manipulated by G. Instead, G aims to generate forgeries $I' = G(I)$, with $I' \simeq I$, in order to fool the discriminator into wrongly classify the forged areas of I' as authentic. The training of the two modules can be seen as a competitive game between them, at the end of which the generator is able to create forgeries that are difficult to detect, and the discriminator is able to correctly classify them.

In addition to the described GAN network, the authors used a custom CNN model specifically designed to detect similarities between regions (*i.e.*, copy-moved areas). This CNN is composed of different convolutional layers as well as custom ones that perform a self-correlation operation on the input features. Then, different pooling steps are used to extract more compact features that are fed to fully connected layers. Finally, a mask-decoder layer is used to reconstruct, from the extracted features, a binary mask that represents the similar regions in the image.

As a final decision step in the forgery detection pipeline, a linear SVM model is used for classification. The SVM is fed with an input vector that combines the output of the GAN and the output of the similarity detection CNN. If the image is classified as copy-moved by the SVM model, an additional mask is given as output by comparing the two input binary masks obtained by the GAN and the custom CNN, in which not only the forged areas are labelled, but also the source region used for the copy-move attack is identified (with a different label).

Two datasets unrelated to forgery detection, namely, the CIFAR-10 [78] and MNIST [84] datasets, were used to pre-train and test the GAN network. In detail, the CIFAR-10 dataset contains 60 000, 32×32 color images categorized as 10 distinct

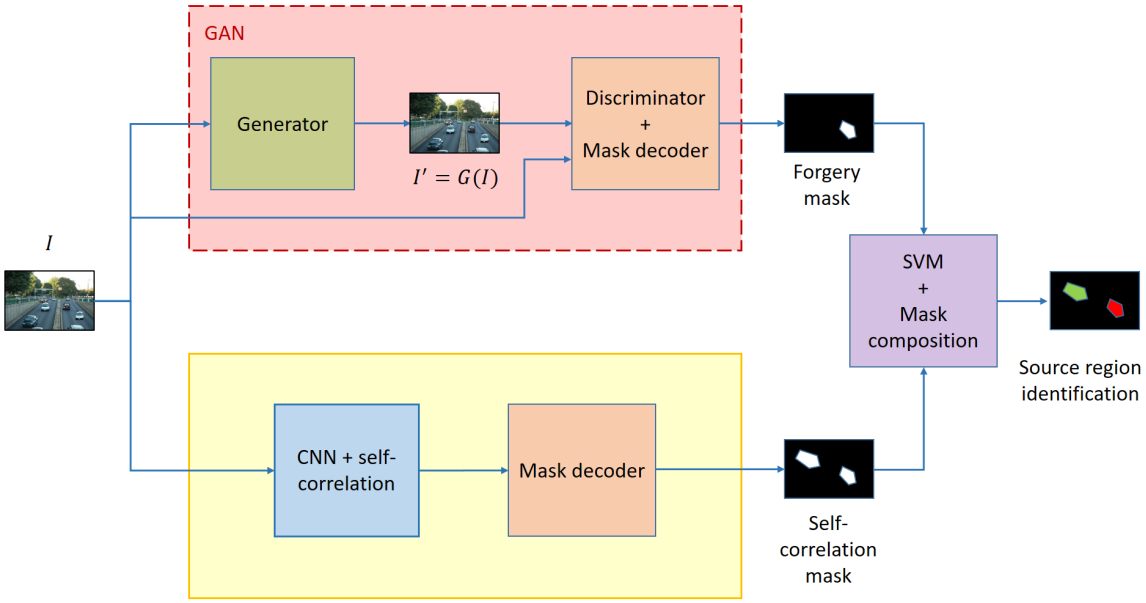


Figure 2.2: Architecture of the GAN-based method in [11]. The upper branch implements a per-pixel binary classificator (forged/pristine), while the bottom one is used to find similarities between regions. The outputs of these branches are then combined to obtain the final output mask in which, if the image is considered forged, source and target regions can be distinguished.

classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), while MNIST is composed of 60 000 grayscale images depicting handwritten digits. After the pre-training phase, the other two modules of the detection pipeline were trained and validated on a custom dataset composed of a total of 1792 pairs of forged and corresponding authentic images, sampled from MICC-F600 and two other datasets, the "Oxford Building Dataset" [110] and the "IM" [22].

The obtained detection performances on this composite dataset are as follows:

- F_1-score: 88.35 %;
- Precision: 69.63 %;
- Recall: 80.42 %.

In conclusion, it would have been interesting if the authors evaluated the performances of their method on one of the public benchmark datasets (such as MICC-F2000, or CASIA2) rather than a custom, composite one. One aspect of this method that should be further noted is that it is one of the few that gives as output not only a localization of the forged areas, but also the source regions of the copy-move attacks.

2.5.3 Y. Wu et al.

In this paper, a pure end-to-end deep neural network pipeline (referred to as *BusterNet* by the authors) is presented as a copy-move forgery detection solution. A key aspect of this method, such as in [11], is the fact that it is able not just to give a pixel-level localization of the copy-move attacks, but it also distinguishes between the source and the target region.

The detection pipeline is composed of two branches and a fusion module (see Fig. 2.3):

- The first branch, called *Mani-det*, is responsible for the detection of manipulations in the image, and it is composed of the following modules: a feature extractor, a mask decoder, and a binary classifier.

The feature extractor is a standard CNN that coincides with the first 4 blocks of the VGG16 network [132].

The mask decoder is used in order to restore the input resolution of the image, via a de-convolution process, and it uses the *BN-inception* and *BilinearUpPool2D* layers [144].

The binary classifier, which is implemented as a convolutional layer followed by a sigmoid activation function, produces a binary manipulation mask, in which the pasted patches of the copy-move attacks are localized;

- The second branch, referred to as *Simi-det*, is used in order to generate a copy-move binary mask, in which similar regions in the input image are detected. In particular, the detection process can be summarized as follows: first, a CNN is used as feature extractor. Then, a self-correlation module is used to compute all-to-all feature similarities. These are given, as input, to a percentile pooling unit, which collects useful statistics. A mask decoder is used to up-sample the obtained tensor to the size of the input image. Finally, a binary classifier is applied in order to obtain the copy-move mask;

- The fusion unit takes as input the computed features from the two branches. It is constituted by a convolutional layer followed by a soft-max activation, that gives as output a three-class prediction mask: pristine, source region, and target region.

Note that the CNN networks used in the *Simi-det* and in the *Mani-det* branches have the same architecture, but they have different weights, since they are trained independently. The same applies for the mask-decoder and the binary classification modules.

In order to train their model, the authors built a dataset of 100 000 images by automatically performing copy-move operations from source pristine images. For each tampered image, they built three ground-truth pixel-level masks:

- A three-class mask $M_{s,t}$ with the following labels: pristine, source copy-move, and target copy-move;
- A binary mask M_{man} with the following labels: pristine and manipulated. Note that the source region here is considered pristine;
- A binary mask M_{sim} with the following labels: pristine and copy-move. Note that the source and target regions are both labeled as copy-move.

The authors adopted the following three-stage strategy for training:

1. Each branch is trained independently. In order to do so, the copy-move mask M_{sim} and the manipulation mask M_{man} are used, as ground-truth, for the *Simi-det* and *Mani-det* branches, respectively;
2. The weights of each branch are frozen and the fusion module is trained with the three-class mask $M_{s,t}$ as ground-truth;
3. A fine-tuning step is performed by un-freezing the weights of the two branches and training the whole network end-to-end.

The performances of the method were evaluated on CASIA2. As CASIA2 contains both copy-move and splicing attacks, the authors selected a total of 1313 copy-move-only images along with their authentic counterparts, thus obtaining a test-set of 2626 images. The authors used the following metrics: precision, recall, and F_1 score, and they computed them both at image level and at pixel level. For the latter, the authors used two different approaches: (i) aggregate TPR, FPR, and FNR over the whole dataset, and (ii) compute precision, recall, and the F_1 score for each image and then average the results over all of them. The obtained results are reported in Table 2.3.

2.5.4 M. Elaskily et al.

In [41], a method for copy-move forgery detection is presented. It is purely DL-based, that is, no separate features are pre-computed. In detail, the authors built a CNN with the following architecture:

Table 2.3: Performances of *BusterNet* [146] on CASIA2.

Eval. method	**Prec. %**	**Recall %**	**F_1 %**
image level	78.22	73.89	75.98
pixel level (i)	77.38	59.15	67.05
pixel level (ii)	55.71	43.83	45.56

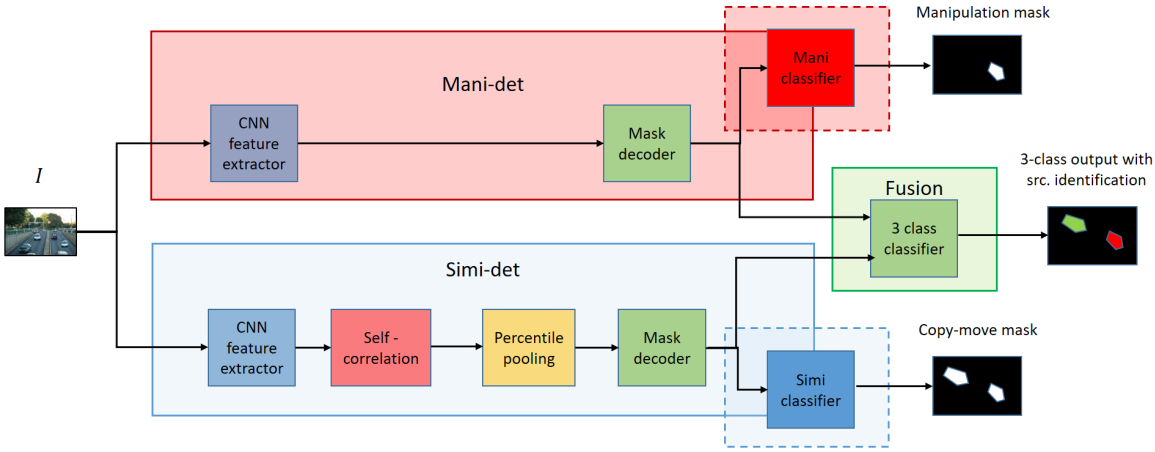


Figure 2.3: Architecture of *BusterNet*. [146].*Mani-det* branch is used to obtain a classification of each pixel of the input image as forged or pristine. *Simi-det* branch instead, aims to find similarities between pixels in the input image. Finally, a fusion module is employed that takes as input the outputs of the two branches and outputs a classification for each pixel: source, target or pristine.

- Six convolutional layers, each one followed by a max pooling layer;
- A Global Average Pooling (GAP) layer, used to reduce the number of parameters of the network and to limit the probability of overfitting. This layer acts as a fully-connected dense layer;
- A soft-max classification with two classes: authentic or forged.

Therefore, the method does not give as output the localization of the forged regions, but only a global classification of the image. It has been evaluated on 4 benchmark datasets: MICC-F220, MICC-F600, MICC-F2000, and SATs-130. Since each of the listed dataset is quite too small to train a CNN, the authors merged them into a new one that could be more suitable for the training phase. The obtained dataset is thus composed of 2916 images: 1010 tampered and 1906 original.

The authors used the following metrics in order to evaluate the performance of the method: accuracy, TPR, TNR, FNR, and FPR. The metrics were evaluated by a k-fold (with $k = 10$) cross-validation. To elaborate, for each validation a random split of the composed dataset is performed: 90% for training and 10% for testing. Here, the 10% testing images is selected all from one of the 4 constituting sets of the composed dataset.

The obtained metrics are presented in Table 2.4, and they are actually really high. However, we observe that the testing was performed on a small percentage (10%) of the composed dataset, which contains images from all the 4 benchmark datasets themselves. As a consequence, test and training images are possibly highly correlated. Hence, they likely have similar kind of forgeries, that is, with similar dimensions and types of post-processing operations. It could have been interesting if the authors trained their model on one dataset, like MICC-F2000, and evaluated it on another one, such as MICC-F600, in order to better assess the robustness and generalization capability of the model.

Table 2.4: Performance metrics of [41].

Dataset	TPR %	TNR %	FPR %	FNR %
MICC-F220	100	100	0	0
MICC-F600	100	100	0	0
MICC-F2000	99.24	100	0	0.76

2.5.5 J. Ouyang et al.

The method presented in [107] is an end-to-end deep learning approach that features a CNN for binary classification (forged *vs.* authentic) of the whole image. The crucial aspect of this approach is the use of the transfer learning technique, as follows:

1. A CNN with the same architecture as AlexNet [79] is used as base-model;
2. The classification layer is changed in order to have two classes as output: authentic or forged;
3. The weights of the AlexNet model trained on the ImageNet dataset [33] are used as initial weights for the training step;
4. A first training phase is carried out by freezing the weight values of the first levels of the network;
5. A second training phase (which is often referred to as "fine tuning") is performed by de-freezing all the network weights, and by using a smaller learning-rate value than the one used in the first training step (such as 10^{-5}).

Since, as already mentioned before, these public forgery detection datasets are not extensive enough for training a CNN without introducing overfitting issues, the authors artificially created copy-move operations by randomly selecting rectangles from an image and pasting them in different locations on the same image. By adopting this approach, they built the following datasets:

- "data1", that contains (i) all the 1338 color images from the UCID dataset [126], and (ii) a total of 10 000 forgeries obtained by applying the above discussed copy-move operations to the original images;

- "data2", that contains (i) all the 8189 color images from the Oxford flower dataset [105], and, again, (ii) a total of 10 000 forgeries obtained with copy-move operations on the original images.

The training of the network was done on both the "data1" and "data2" datasets. Data-augmentation with flipping and cropping operations was performed on the authentic images in order to balance the distribution of the two classes.

For the model performance evaluation, the "data1", "data2", and CMFD datasets were used. The obtained results are reported in terms of test detection error (which is the measure complementary to accuracy). They are as follows: 2.32%, 2.43% and 42% for "data1", "data2" and CMFD, respectively.

From these results it is clear that, even if the model performs well on the custom datasets, it has poor generalization capabilities for real-scenario forgeries, such as the ones contained in CMFD, likely due to its basic approach in generating forgeries. However, this simple approach could still be useful if richer copy-move datasets were available, or a more sophisticated algorithm could be used to build synthetic forgeries, such as a GAN network (see Section 2.5.2).

2.5.6 Amit Doegara et al.

The authors of [36] proposed a simple yet effective method for copy-move detection.

A pre-trained AlexNet model [79] on MICC-F220 dataset is used to extract deep feature vectors of 4096 elements from the input images (note that, in order to obtain the feature vector, the classification layer of the AlexNet network is removed).

An SVM model is then fed with the extracted features and used to obtain a binary classification on the whole image: either pristine or forged.

The training process is carried out in two phases (see Fig. 2.4). First, the pretrained AlexNet CNN is used to extract features both from the pristine images and from the forged ones. As a pre-processing step, the images are resized to match the input dimension required by the AlexNet model, which is 227×227 pixels. Then, the SVM classifier is trained on the obtained dataset of features and corresponding binary labels.

The authors evaluated their method on the MICC-F220 dataset, and it obtained the following results:

- FPR: 12.12%;
- TPR: 100 %;
- Precision: 89.19 %;
- Accuracy: 93.94 %.

Even if the accuracy is quite high, there is still room for improvement as the number of false positives is not really low, especially if compared with other approaches, such as [14], in which the reported FPR ratio was of 8 %, along with a TPR of 100 %.

A final note on the choice of MICC-F220 dataset for performance evalutation is in order. This dataset is also used for pre-training the AlexNet model used by the authors. In the paper, it is not clear which portion of the dataset is used for training and which for testing. Therefore, it is not possible to evaluate if and how much the reported results are affected by bias due to correlation between training and testing sets. In order to clear up these issues, the authors could have used different datasets for either phase instead, such as MICC-F2000 or MICC-F600.

2.6 Copy-move and splicing methods

We now move on to discuss those methods designed to detect both copy-move and splicing forgeries.

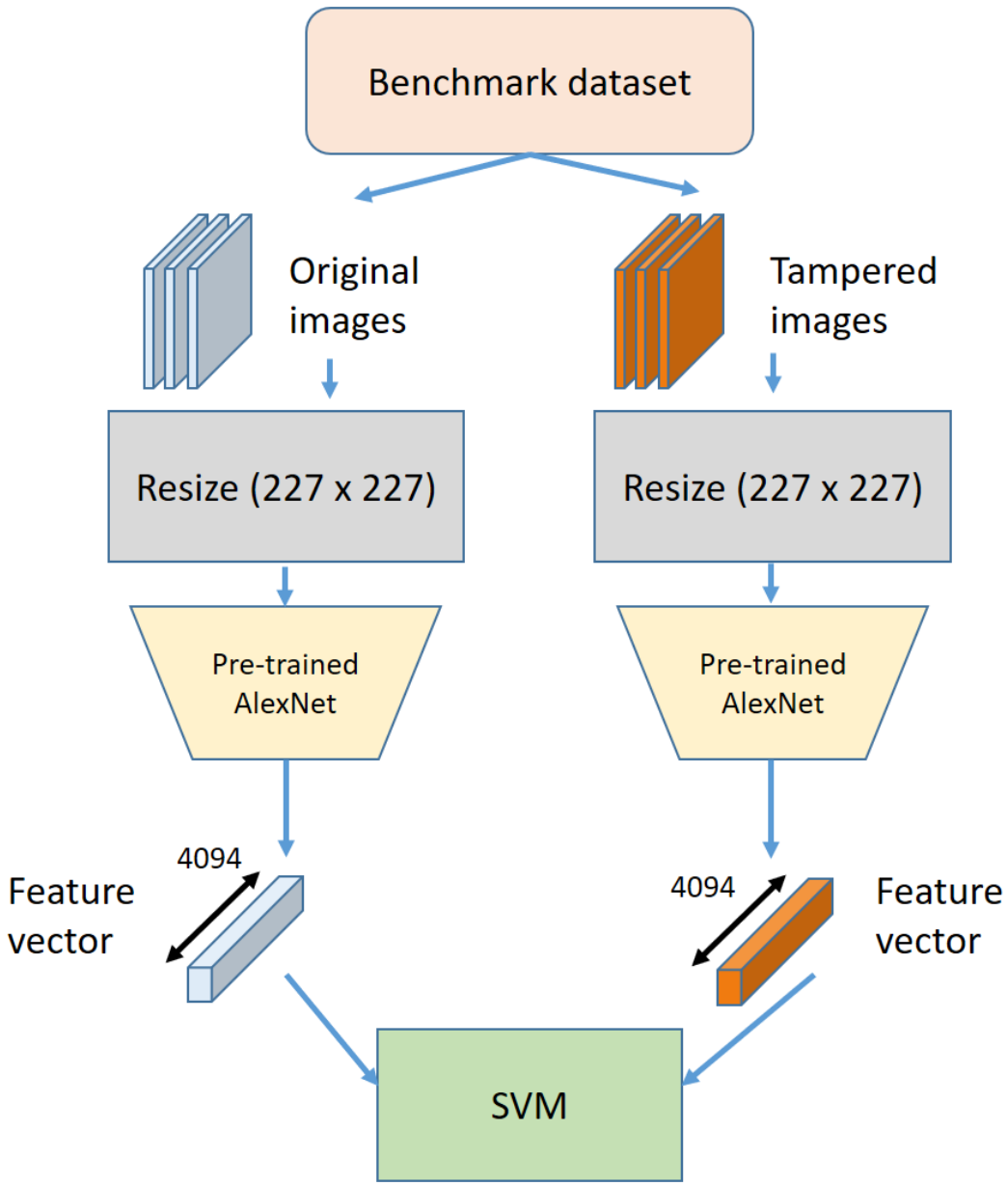


Figure 2.4: Detection approach of [36]. A pre-trained AlexNet is used as feature extractor. The extracted features, either from pristine or forged images, are then used to train a SVM classifier to obtain the final decision on the input image: forged VS pristine.

2.6.1 Cozzolino and Verdoliva

In this work, the authors presented a deep learning approach that aims to extract a camera model noise pattern (referred to as "noise print") as a means to detect forgeries.

A digital camera, due to the on-board processing operations carried out on the signal received from the sensor, leaves on the generated picture a distinctive pattern of artifacts that are model-specific. This can be exploited, in a forensic scenario, to estimate from which camera model a certain picture was taken from. This idea can also be applied for the purpose of forgery detection. For instance, in the case of a spliced image, if the patch used to create the composition was extracted from a photo taken by a different camera model, then inconsistencies between the camera model artifacts could be leveraged in order to detect the tampering.

A useful property of the camera noise pattern is that it is not space invariant. This means that two patches extracted at different locations from the same image are characterized by different noise artifacts. By exploiting this property, this method can also be used for copy-move detection, as the camera noise pattern at the target location of the copy-move attack is hardly consistent with the expected one at that particular location.

The authors used the pre-trained denoising CNN presented in [154] as the starting point for their approach. This network was trained with a great number of paired input-output patches, where the input is a noisy image and the output is its corresponding noise pattern.

In order to estimate the camera model noise print, a further training of the previous architecture was performed. Since a mathematical model describing the camera noise pattern is not available, it is not easy to build a dataset with pairs of an input image and its corresponding desired camera noise print. In order to overcome this problem, the authors used the following key idea: patches extracted from images taken with the same camera model, and at the same location, should share similar camera noise print, while this should not be true for patches coming from different camera

models or from different spatial locations. Following this insight, the authors built a Siamese architecture, in which two identical Residual CNNs (initialized with the optimal weights computed in the first training phase) are coupled and the prediction of one network is used as desired output for the other one and *vice-versa*. The overall architecture is shown in Fig. 2.5.

In the training phase, the two CNNs are fed with patches x_i^a and x_i^b, respectively. These patches can be:

1. extracted from images taken from different camera models;
2. extracted from images taken from the same camera model, but at different spatial locations;
3. extracted from images taken from the same camera model, at the same location.

The input pair (x_i^a, x_i^b) is assigned, as expected output, a positive label $y_i = +1$ ("similar camera noise print") in the third case, while a negative label $y_i = -1$ ("different camera noise print") in the first and second cases. The output of the Siamese architecture is obtained by means of a binary classification layer that takes as input the noise print extracted by the two CNNs. This output is then compared to the expected label y_i and the error is back-propagated through the network. This way, the network is pushed towards generating a similar noise print for patches from the same camera model (and at the same location), and different ones for patches corresponding to different camera models and/or locations. As a result, the network learns to enhance the specific model artifacts and discard the irrelevant features, while reducing the high level scene content of the images. Once the network is trained, the noise print can be obtained as output of one of the two CNNs from an input target image.

In order to detect and localize forgeries, the authors used the EM (Expectation - Maximization) algorithm. With the assumption that the pristine and manipulated parts of the target image are characterized by different camera noise models, the algorithm searches for anomalies with respect to the dominant model. This is done by

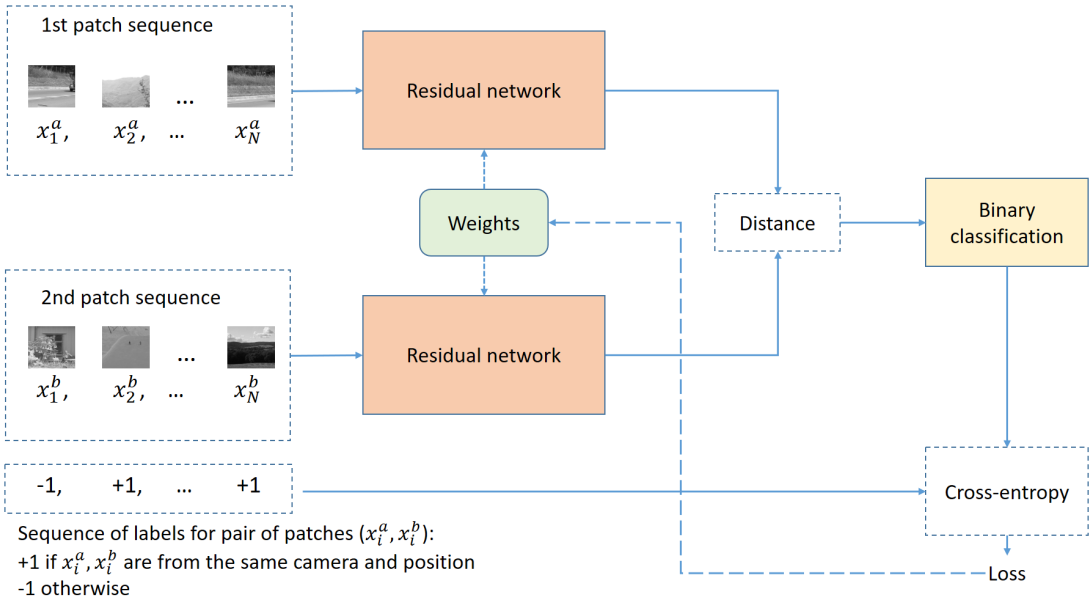


Figure 2.5: Architecture of the Siamese network proposed in [29]. Two residual networks (with shared weights) are trained to extract noise patterns that are given as input to a binary classificator. The model learns to extract similar noise patterns for positive labels (patches from same cameras) or different ones for negative labels (patches from different cameras and/or different spatial locations).

extracting features from the noise print image at a regular sampling grid, that are then used to train the EM algorithm. A heat-map with the probability of manipulation for each pixel is given as output.

The authors tested their method on 9 different datasets for forgery detection, containing many kind of tampering, such as copy-move, splicing, inpaiting, face-swap, GAN generated patches, and so on. Here, we only report the results on the DS0-1 [31] and Korus [76] datasets, as they contain only splicing and copy-move attacks (with possible post-processing operations). The obtained F_1-score is 78% for DS0-1 and 35% on Korus. The authors also computed the AUC score, which is 82.1% and 58.3%, respectively.

2.6.2 Y. Zhang et al.

The authors of this paper proposed the following approach for image forgery detection:

1. Feature extraction and pre-processing. The image is first converted into the YCbCr color space, then it is divided into 32×32 overlapping patches. For each component of the YCbCr space a total of 450 features are extracted from each patch by leveraging the 2-D Daubechies Wavelet transform;

2. The extracted features from each patch are used to train a 3-layers Stacked AutoEncoder (SAE), which is an unsupervised model. On top of the SAE, an additional MLP (Multi-Layer Perceptron) is employed for supervised learning and fine tuning;

3. Context learning. In order to detect forged regions that span across multiple 32×32 patches, each patch-level prediction from the MLP is integrated with the predictions of the neighboring patches. Specifically, for each patch p, a neighbouring patch set $\mathbf{N}(p)$ with cardinality $k+1$ is defined as:

$$\mathbf{N}(p) = [y_p^0, y_p^1, \ldots, y_p^k] \tag{2.6}$$

   where y_p^0 is the output feature of the SAE for the patch p, and y_p^i, with $i \geq 1$ is the feature of its i-th neighbouring patch;

4. Finally, a binary output $Z(p)$ (forged/authentic) is obtained by computing the average of the MLP predictions of the neighbouring patches and comparing it to a threshold, as follows:

$$Z(p) = \begin{cases} 1 & \text{if} \frac{1}{k+1} \sum_{y_p^i \in \mathbf{N}(p)} \text{MLP}(y_p^i) >= \alpha \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

   where the authors set $k = 3$ and $\alpha = 0.5$.

For the training and testing stages of the model, a total of 1000 images were randomly extracted both from the CASIA1 and the CASIA2 datasets. In particular, 770 images were used for training and the remaining 230 for testing. The authors manually built a pixel-wise ground-truth mask for each image in order to train their model at the patch level. Likewise, a patch-level ground-truth mask for each of the test image was also built, as shown in Fig. 2.6.

In order to evaluate the performance, the authors used the following metrics: accuracy, FPR (fallout), and precision, where the usual rates are again defined at patch-level.

The method can be applied for copy-move detection, as well as splicing detection. Note that this method gives a coarse localization of the forged areas (at patch-level).

The reported performance is 43.1%, 57.67% and 91.09% for fallout, precision, and accuracy metrics, respectively. Even if these performance are not quite satisfactory at a first glance, it should be considered that these metrics are evaluated at patch level, and hence are most restrictive than the the same metrics evaluated at image level.

2.6.3 N. H. Rajini

This technique involves two separate CNN models that are used for different purposes in the forgery detection pipeline. It is able to detect both splicing and copy-move attacks. A schematic view of the method is shown in Fig. 2.7, and it can be summarized as follows:

1. Pre-processing stage. The image is first converted into the YCbCr space. Then, a Block DCT is applied on each Y, Cb, and Cr component. In order to reduce the effect of the actual image content, horizontal and vertical de-correlation is computed from the DCT coefficients. Finally, a set of features are extracted from these values by means of a Markov Random Chain model;

2. Forged/authentic decision. The extracted features are given as input to the first CNN model, which gives a binary classification of the image as either forged or

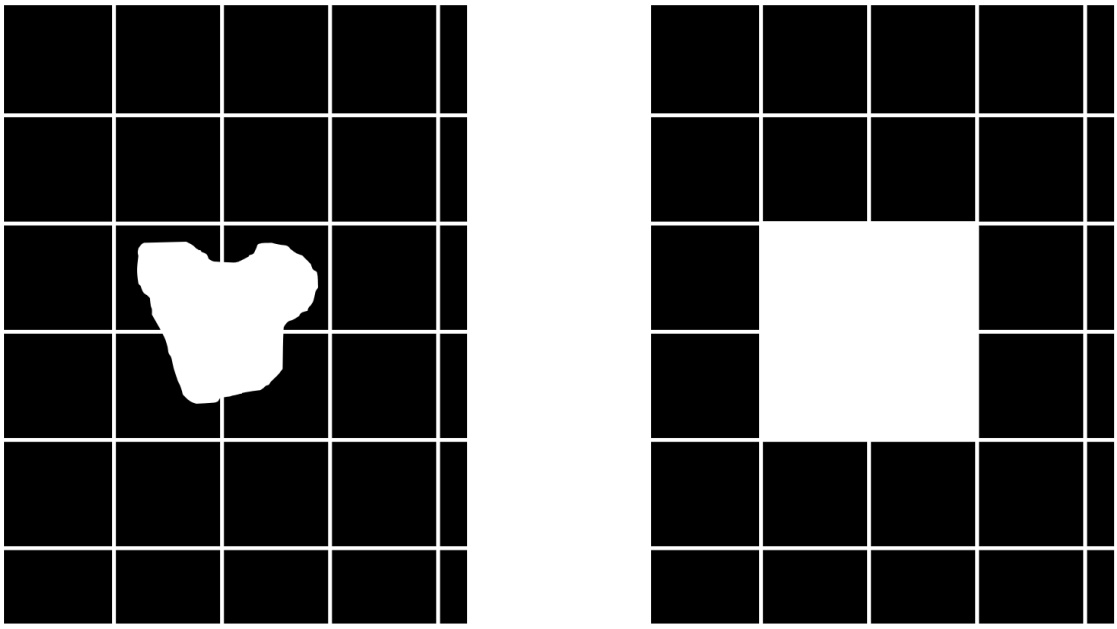


Figure 2.6: Construction of patch-wise ground-truth from the pixel-level mask as in [156]

authentic;

3. Type of attack recognition. In the case that the image is recognized as forged, a second CNN is then employed to classify the type of attack: copy-move or splicing;

4. Post-processing. If a copy-move attack is detected by the second network, further features are extracted and used in order to localize the forged regions.

The authors evaluated their method on the CASIA2 dataset. In particular, they used 80% of the images for training and the remaining 20% for testing. The procedure was repeated 50 times with differently extracted training and testing sets, and the reported performance were computed as an average between all the experiments. The TPR, TNR, and accuracy are used as evaluation metrics.

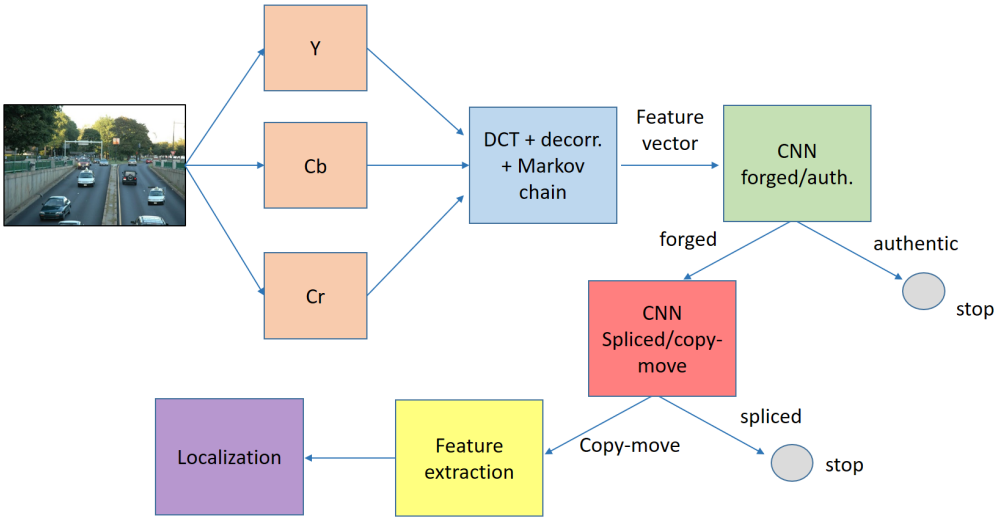


Figure 2.7: Multi-step strategy proposed in [118]. First, features are extracted from the YCbCr converted image to classify the image as authentic or forged. If the image is classified as forged, a CNN is used to distinguish between copy-move and splicing attacks. Finally, in the case of copy-move attack, another feature extraction and localization procedure is employed to obtain a map of the forged regions.

Although the described method can provide as output the localization of the forged areas, the authors only reported performance at a global level (that is, the forged *vs.* non forged image assessment). The obtained results are the following:

- 98.91%, 99.16%, and 99.03% for TPR, TNR, and accuracy, respectively, in the case of copy-move attacks;
- 98.98%, 99.24%, and 99.11% for TPR, TNR, and accuracy, respectively, in the case of splicing attacks.

The reported performance metrics are really high. In addition, they are meaningful from a statistically point of view, as they are evaluated on the sizable CASIA2 dataset. It would have been interesting, though, if the authors evaluated the localization accuracy of their method too, in a similar manner to [156].

2.6.4 F.Marra et al.

The authors proposed a full-resolution, end-to-end deep learning framework for forgery detection.

Typically, due to limited memory resources, deep learning models, such as CNNs, are designed to take as input images with small sizes. So, in order to process high resolution images, either a resize to match the network input size or a patch-level analysis (with possible overlapping) is needed. For computer-vision tasks in which only a high level understanding of the image content is required, such as object recognition, this is usually not an issue. But, for the purpose of forensic analysis, resizing is not recommended, as it tends to destroy important information that is usually stored at high frequencies. Patch-level analysis can also be a limiting factor, as usually the context of the whole image is important as well for forgery detection purposes.

In order to address these problems, the authors built a deep learning framework that takes as input full-resolution images and perform image-level predictions: "forged" or "pristine". The framework is composed of three consecutive blocks:

1. Patch-level feature extraction. This is a CNN that takes as input a patch extracted from the target image and gives as output a feature vector;

2. Future aggregation module. This block takes as input the extracted feature vectors from the overlapping patches and aggregate them together in order to obtain an image-level feature. The authors considered different methods for feature aggregation, such as average pooling, min/max pooling, and average square pooling;

3. Decision step. It is a binary classification process, that was implemented with two fully-connected layers.

The whole framework is trained end-to-end. This is not the case for other similar approaches, in which the patch feature extractor, the feature aggregation module, and the classification layers are trained independently one from the others.

Note that, when an input large size image is processed during training, a great amount of memory is required to simultaneously store all the overlapping patches and to compute their corresponding feature vectors. Also, in the forward pass, the activations in all the intermediate layers need to be memorized for the computation of the loss gradients (needed to update the network weights) in the subsequent back-propagation pass. In order to solve this issue, the authors exploited the gradient check-pointing strategy [24]. This technique consists in saving the activations only at certain check-point layers during the forward pass. In the back-propagation phase, the activations are re-computed up to the next check-point layer and used to compute the gradients. As a consequence, less memory is required at the cost of an increased computational time during the back-propagation.

The authors evaluated their method on the DSO-1 and Korus datasets, obtaining an AUC score of 82.4% and 65.5%, respectively.

2.6.5 Y. Rao et al.

An overview of the architecture of this method is shown in Fig. 2.8. It starts by taking an input RGB image of size $M \times N$ and dividing it into $p \times p$, $p = 128$, overlapping patches X_i, $i = 1, \ldots, T$, where T is the total number of patches. Each patch X_i is given as input to a 10-layer CNN that gives a softmax binary output Y_i, as follows:

$$Y_i = f(X_i) \in \mathbb{R}^2 \tag{2.8}$$

The Y_i vector represents a compact feature that describes the patch i. A global feature vector is then obtained by concatenating the Y_i of each image patch:

$$\mathbf{Y} = [Y_1 ... Y_T] \in \mathbb{R}^{T \times 2} \tag{2.9}$$

A pooling function (either mean or max) is then applied for each of the 2 dimensions:

$$\hat{Y}(k) = \text{Pooling}\left\{Y_1(k) ... Y_T(k)\right\}, k = 1, 2 \tag{2.10}$$

Finally, $\hat{Y}$ is given as input to a SVM classifier that performs a global two-class prediction on the whole image: authentic *vs.* forged.

A key aspect of this technique is the following: in order to suppress the image perceptual content and instead focus the detection phases on the subtle artefacts introduced by the tampering operations, the authors initialized the first CNN layer weights with a set of high-pass filters that are used for residual maps computation in SRM (Spatial Rich Models). This step also has the benefit of speeding up the training phase of the network.

The CNN was trained on the CASIA1, CASIA2, and DVMM datasets. This method can be applied both for splicing and copy-move detection, because the CNN and the SVM are trained on the aforementioned datasets, which contain both type of forgeries. Note that the SVM classification step is only used for the CASIA datasets.

The detection performance, in terms of accuracy, is 98.04%, 97.83%, 96.38% on CASIA1, CASIA2, and DVMM datasets, respectively. These accuracy values are objectively high. This is true in particular in the case of CASIA2, which is the dataset

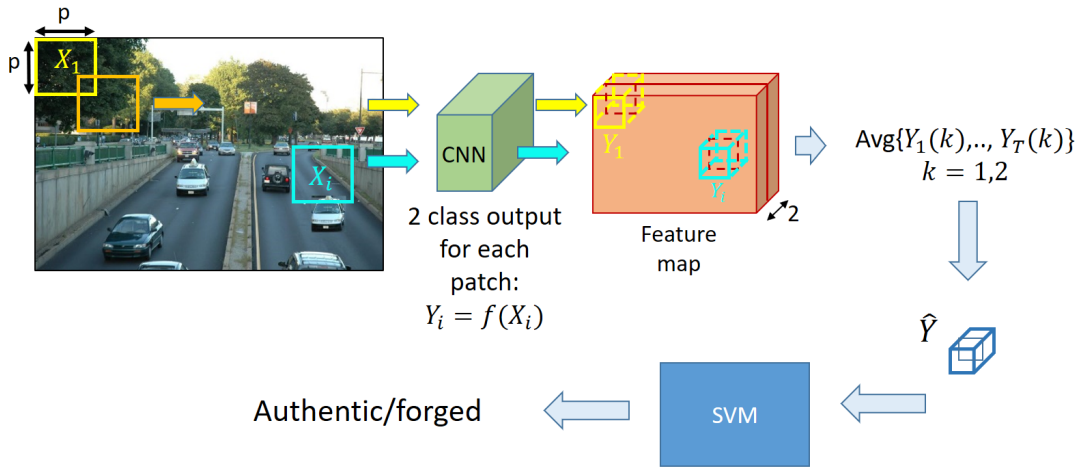


Figure 2.8: Architecture of the technique in [120]. Overlapping patches are extracted from the input image and feature vectors are extracted from each of them. A global feature, computed by averaging along the spatial dimension, is then fed to an SVM model, which is used to obtain the final global classification: forged VS authentic.

with not only the most images (and consequently it is the most statistical relevant, as we said before), but it also contains both splicing and copy-move attacks. It should be noted, though, that this method only gives a global binary prediction on the image, and no localization of the forged areas is performed.

2.6.6 M. T. H. Majumder et al.

The approach described in [97] is also based on a CNN to classify an image as authentic or forged. In contrast to the previously discussed methods, however, in which deep learning networks were composed of a high number of layers, in this case a shallow CNN model, composed of just two convolutional layers, was employed. Also, no max-pooling steps were used for dimensionality reduction, as this goal was achieved by exploiting large convolutional filters, with size of 32 by 32 and 20 by 12 for the first and the second layer, respectively.

This strategy is based on the following idea: in deep neural networks, complex high-level features are learnt at deeper levels, while more simple visual structures, such as edges and corners, are learnt at the first ones. Hence, in order to detect the artefacts

introduced by forgery operations, low-level features are more likely to be useful. As a consequence of this choice, the number of parameters of the network is limited, thus allowing for training with less over-fitting risk.

The CASIA2 dataset was used both for training and testing. The authors trained their shallow network multiple times in an independent fashion, using different pre-processing strategies, such as: raw input (that is, no pre-processing), DCT-based transformation, and YCbCr space conversion. They showed that the best results were obtained without any kind of pre-processing.

To further reduce the risk of overfitting, real-time data augmentation was applied during training, with transformations such as shearing, zooming, and vertical and horizontal flipping. An accuracy of 79% was obtained with this training strategy, and, as we said, without pre-processing.

As a comparative experiment, the authors also applied the aforementioned transfer learning technique, by using two deep learning models with a high number of layers that were pre-trained on the ImageNet dataset: the VGG-16 [132] and the well-known ResNet-152. Despite the fact that these models perform well on standard image classification problems, they were not able to transfer the acquired knowledge to this specific task, and a substantial underfitting issue was observed in the training phase. The outcome of this test validated the choice of a shallow model instead of a deep one.

The main contribution of this work is therefore the usage of a shallow network, in which low-level features are exploited as a mean to detect subtle artefacts generated by tampering (rather than high-level ones), which thus can be used for the forgery detection task. Also, the authors showed that large convolutional filters can be exploited in place of max-pooling layers to reduce the number of network parameters, therefore reducing the risk of overfitting. Despite this, the obtained accuracy still leaves room for improvement.

2.6.7 R. Thakur et al.

In [136], a filtering scheme based on image residuals is exploited. Therefore, the residuals, rather than the raw images, are fed as input to a CNN network for classification (as usual, original/forged). This approach is tailored to pursue high frequencies in the image data, which, as often assumed even by the other approaches, carry most of the possible tampering traces. The image residuals are computed as follows:

1. The image is resized at the 128×128 size, and converted to grayscale;

2. The second-order median filter residuals (SDMFR) are then calculated as follows. Given an image, a first median filtering is applied:

$$y(i,j) = \text{med}_w[x(i,j)] \tag{2.11}$$

where w is a 5×5 window and $x_{i,j}$ is the (i,j) pixel intensity. Then, a second median filtering is applied to the median-filtered image:

$$z(i,j) = \text{med}_w[y(i,j)] \tag{2.12}$$

Finally, the residuals are obtained by subtracting the second order median filtered image from the first order filtered image:

$$MFR(i,j) = z(i,j) - y(i,j) \tag{2.13}$$

3. Laplacian filter residuals (LFR) are also computed, with the following algorithm. Let:

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.14}$$

be the Laplacian kernel filter. The Laplacian-filtered image is obtained by convolving the original image with K, that is:

$$L(i,j) = (x * K)(i,j) \tag{2.15}$$

The residuals are then calculated as the difference between the filtered image and the original one:

$$LFR(i,j) = L(i,j) - x(i,j) \tag{2.16}$$

Both the SDMFR and the LFR residuals are fed to the CNN classification network as a combined input. The CNN model comprises 6 convolutional layers, each one followed by a max pooling step (except the first one). Two fully connected layers are then used before the final binary softmax classifier.

The authors trained and tested their network on two different datasets: the CoMoFoD and the BOSSBase [17]. In the case of the first dataset, a split of 70% and 30% has been made for training and validation, respectively. In the case of the second one, as it is composed of 10 000 raw pristine images, the authors applied median filtering to each image in order to simulate a tampering operation, thus obtaining a total of 20 000 images (half authentic and half filtered). Then, they split the obtained dataset into 70% for training and 30% for validation.

The accuracy obtained on both datasets is high: 95.97% for the CoMoFoD dataset, and 94.26% for the BOSSBase. However, it could have been interesting if the authors tested their method, without retraining, also on other benchmark datasets for forgery detection, such as CASIA2, MICC-F2000 or MICC-F600, in order to assess its generalization capability.

2.7 DeepFake methods

We now present a few of the most recent DeepFake-specific detection methods, that achieved the best results on the previously introduced datasets for DeepFakes detection evaluation (see Section 2.3). The selection has been made according to the criteria previously outlined, namely, suitability for the still images case.

2.7.1 A. Rössler et al.

In [125], the authors developed a method to detect image DeepFakes that is based upon the *XceptionNet* architecture proposed by *Google* in a previous paper [26].

The main peculiarity of this model is the employment of a custom layer, called *SeparableConv*, whose purpose is to decouple the depth-wise convolution from the spatial one, thus reducing the number of weights of the model itself.

The detection pipeline can be summarized as follows: a state-of-art face detection/tracking method [138] is used to extract the face region from the image/frame, which is cropped as a slightly larger rectangle than the size of the face in order to include some contextual information.

The obtained bounding box is then fed to a modified XceptionNet for binary classification. In order to do this, the final fully-connected layer of the original XceptionNet is substituted with a fully-connected layer with binary output.

The authors adopted the following transfer-learning strategy to train the model:

1. The weights of each layer from the original XceptionNet are initialized with the ImageNet ones, while the fully-connected layer is random initialized;

2. The network is trained for 3 epochs, with all the weights freezed except the ones in the fully-connected layer;

3. All the weights are un-freezed and the network is trained for other 15 epochs (fine-tuning step).

The authors released three different versions of their model: the first one is trained on uncompressed videos, while the second and the third one were trained on videos compressed with H.264 codec at quantization levels of 23 and 40, respectively. We denote these variants as Xception_a, Xception_b, and Xception_c, respectively.

While Xception_a achieved the best results on FaceForensic++ dataset, with a detection accuracy of 99.7%, its performance dropped when evaluated on DFDC and CelebDF, with accuracy scores under 50% in both cases. Xception_b achieved the best accuracy on DFDC (72.2%), while Xception_c performed better on CelebDF, with an accuracy of 65.5%.

2.7.2 Huy H. Nguyen et al.

In this paper [101], a novel forgery detection framework, called *Capsule-Forensic* was proposed. Its main feature is that it uses a particular kind of neural network, *Capsule Network* (first introduced in [57]), as the binary detector, instead of the more usual convolutional neural networks.

Capsule Networks were designed in order to efficiently model hierarchical relationships between objects in an image, and to infer not only the probability of observation of objects, but also their pose estimation.

The main idea behind Capsule Networks is the concept of "capsule". A capsule is an ensemble of neurons that describe a set of properties for a given object. In contrast to single neurons, in which the scalar output represents the probability of observation of a certain feature, the output of a capsule is an activation vector, in which each element represents the activation of one of the capsule's neurons, *i.e.*, the value corresponding to the associated feature.

Capsules are arranged in different layers in a hierarchical fashion: a parent capsules receives, as input, the output of different child capsules. The connections between child and parent capsules (*i.e.*, which outputs are kept and which are discarded for the next layer) are not fixed at the beginning, such as for max/average pooling layers (usually employed in standard CNNs), but they are dynamically computed by means of a routing by agreement algorithm.

Thanks to this procedure, child capsules whose predictions are closest to the predictions of certain parent ones become more and more "attached" to these parents, and a connection can be considered established. The interested reader is referred to the original paper for a more detailed explanation on how the hierarchical connections are built.

Among the advantages of Capsule Networks compared to CNNs, a remarkable fact is that they have less parameters, as neurons are grouped in capsules and the connections between layers are between capsules and not directly between neurons. Also, thanks to the presence of pose matrices, they are robust against viewpoint changes

under which objects are seen in the image. This is not true for CNNs, that need to be trained on lots of possible rotations and transformations in order to generalize well to unseen transformations.

The proposed method is designed for different forensics tasks, such as (i) DeepFakes detection, and (ii) computer-generated frame detection, both for image and video content.

The detection pipeline (shown in Fig. 2.9) comprises the following elements:

- Pre-processing phase. It depends on the specific forensic task at hand, *e.g.*, for DeepFakes detection it involves a face detection algorithm in order to extract the face region, while for CGI detection it consists in patch extraction from the input image. For video content the frames are separated and fed one by one to the subsequent steps;

- Feature extraction. This is done by using the first layers of a VGG_19 network pre-trained on ILSVRC dataset [124]. These weights are fixed during training;

- Capsule Network. It is the core of the detection method, involving three primary capsules (children) and two output capsules "Real" and "Fake" (parents). The predicted label is computed as in Eq. (2.17):

$$\hat{y} = \frac{1}{M} \sum_{i=1}^{M} \text{softmax} \left(\begin{bmatrix} \mathbf{v}^1 \\ \mathbf{v}^2 \end{bmatrix}_{:,i} \right), \tag{2.17}$$

  where $\mathbf{V}^1 \in R^M$ and $\mathbf{V}^2 \in R^M$ represent the output capsules, and M is their dimension;

- Post-processing phase. As the pre-processing step, this is task-specific: the scores are averaged among patches for computer generated image detection, or among frames for video input.

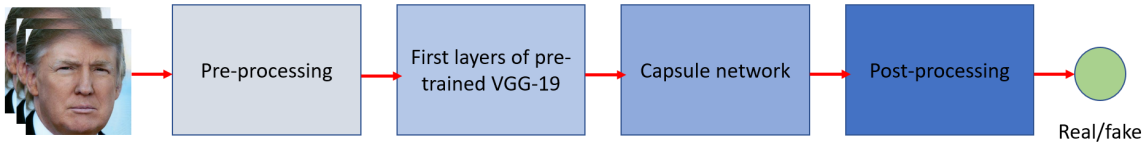


Figure 2.9: Overview of method [101]. Note that pre-processing and post-processing stages are task-dependent, *e.g.*, for DeepFake detection in the former a face tracking algorithm is used to extract and normalize the face region, while for CGI detection this step consists in the extraction of overlapping patches.

The achieved detection accuracy is very high on FaceForensic++, with a score of 96.6%, but it is lower on the more challenging datasets DFDC and CelebDF, with accuracies of 53.3% and 57.5%, respectively.

2.7.3 Y. Li et al.

In [86], the authors proposed a deep learning method to detect DeepFakes based on the following observation: typically, DeepFakes generation algorithms tend to leave distinctive artifacts in the face region due to resolution inconsistencies between the source image/video and the target one. In particular, GAN-synthesized face images are usually of a fixed low resolution size and, in order to be applied to the target video, an affine warping needs to be performed in order to match the source face to the facial landmarks of the target face. If the resolutions of the source and target videos do not match, or if the facial landmarks of the target person are far from the standard frontal view, these artifacts are more and more evident.

The authors trained four different CNNs, namely a VGG-16, a ResNet50, a ResNet101, and a ResNet152 to detect these kinds of artifact. In particular, they used a face-tracking algorithm to extract regions of interest containing the face as well as the surrounding area, which are then fed to the networks. The reason why also a portion of the surrounding area is included is to let their model learn the difference between

the face area, that contains artifacts in the case of positive (fake) examples, and the surrounding one, which does not contain artifacts.

The authors used the following training strategy. Instead of generating positive examples by means of a GAN-syntesization algorithm, which in turn requires a good amount of time and computational resources to train and run, they generated positive examples by simulating the warping artifacts with standard image processing approaches, starting from negative (real) images. The processing steps are summarized as follows:

1. The face region is extracted with a face tracking algorithm;

2. The face is aligned and multiple scaled versions are created by down/up-sampling the original one. Then, one scale is randomly selected and Gaussian-smoothed. This has the effect of simulating the mismatch in resolutions between source and target videos;

3. The smoothed face is then affine-warped to match the face landmarks of the original face;

4. Further processing can be done in order to augment the training data, such as brightness change, gamma correction, contrast variations, and face shape modifications through face landmarks manipulation.

The detection accuracy obtained are: 93.0% for FaceForensic++, 75.5% for DFDC and 64.6% for CelebDF.

2.8 Performance comparison

In this Section we proceed to compare the previously described forgery detection methods from a performance perspective.

We begin by comparing techniques specific for copy-move and splicing, while DeepFake detection algorithms are discussed in a separate section. In fact, even if the DeepFake methods that we previously discussed can be seen as a particular kind

of splicing attack, they are mostly performed on faces. As a consequence, DeepFake detection techniques must be evaluated with datasets specialized on face manipulations, while the standard splicing datasets, such as CASIA, contain pictures of generic scenes. Furthermore, these methods can successfully exploit domain specific knowledge, such as face landmarks, mouth/eyes-based features, and so on, while of course this is not the case for generic splicing detection algorithms.

2.8.1 Splicing and copy-move methods

In Table 2.5 the performance of all previously discussed copy-move and splicing detection techniques are reported. For each method, we also indicate the type of detected attacks (splicing, copy-move, or both) and the capability or lack thereof to give as output the localization of the forged areas.

As a first comment, from the sparseness of the table it is easy to see that it is very challenging to compare the different techniques strictly in terms of performance. This is due to a number of reasons. The first and most obvious one is that approaches designed specifically for copy-move detection cannot be easily evaluated on CASIA (both v1.0 and v2.0) datasets, as these also contain splicing attacks (an exception can be made for method [146], that was evaluated on a copy-move-only subset of the dataset itself, see Section 2.5.3). In this case, copy-move specific datasets, such as MICC-F220, MICC-F600, and MICC-F2000 should be considered for evaluation.

The second reason is that the presented methods, especially in the case of copy-move specific ones, are mostly not trained nor tested on the same benchmark sets. This is due to the fact that some of the standard datasets are either too small for training a highly parameterized deep learning model, or contain only naive attacks (such as MICC-F220, in which copy-moved regions are square or rectangular patches). For this reason, different authors instead built their own custom datasets to fulfill their specific requirements, either by merging together the benchmark ones or by artificially generating them. However, the downside of this approach is the difficulty of comparing the results achieved by other techniques.

Therefore, the comparison between different techniques, when it is possible, is performed by grouping them on the basis of specific criteria, such as the type of detected attacks, the dataset used for evaluation, and the localization property.

We start by focusing our analysis on the methods designed for copy-move only forgeries, then proceed to both copy-move and splicing detection techniques, and conclude with DeepFake specific ones.

Table 2.5: Copy-move and splicing detection methods performance comparison

Method	Detected attacks	Localization	CASIA1 Acc.%	CASIA2 Acc.%	MICC-F220 Acc.%	MICC-F600 Acc.%	MICC-F2000 Acc.%	Other perf.
[13]	copy-move	yes	-	-	99.11	-	-	55% FPR on MICC-F220
[11]	copy-move	yes + source id.	-	-	-	-	-	88.35% F1-score on custom dataset
[146]	copy-move	yes + source id.	-	76.65	-	-	-	75.98% F1-score on CASIA2
[41]	copy-move	no	-	-	**100**	100	99.7	-
[107]	copy-move	no	-	-	-	-	-	43% det. error on CMFD
[36]	copy-move	no	-	-	93.94	-	-	-
[29]	Splicing copy-move	yes	-	-	- [c]	-	-	82.1% AUC on DS0-1 and 58.3% AUC on Korus
[156]	Splicing copy-move	block-wise	91.09 [a]	91.09 [a]	-	-	-	-
[118]	Splicing copy-move	yes	-	**99.07** [b]	-	-	-	-
[98]	Splicing copy-move	no	-	-	-	-	-	82.4% AUC on DS0-1 and 65.5% AUC on Korus
[120]	Splicing copy-move	no	**98.04**	97.83	-	-	-	96.38% acc. on DVMM
[97]	Splicing copy-move	no	-	79	-	-	-	-
[136]	Splicing copy-move	no	-	-	-	-	-	95.97% acc. on CoMoFoD

[a] The accuracy is computed on a dataset obtained by randomly selecting a total of 1000 images from CASIA1 and CASIA2.
[b] This value was obtained as average of the splicing detection accuracy (99.03) and the copy-move detection accuracy (99.11).
[c] The accuracy score is sub-par compared to the performance obtained on the datasets used by the authors. This was probably due to the fact that MICC-F220 is a dataset of small-sized JPEG compressed images, which are very different from the images used to train the method.

2.8.1.1 Copy-move detection methods

We start the present analysis by first comparing methods [41], [13], and [36], as they have been all tested on the MICC-F220 dataset. The first method achieved a slightly better accuracy and a considerable better FPR (see Table 2.4) than the other two, along with a considerably better accuracy. In addition, [41] has been shown to

achieve perfect results on MICC-F600 and almost perfect ones on MICC-F2000, which are more significant evaluation datasets (see Section 2.5.4). However, it should be considered that [41] only gives as output a global decision on the authenticity of the image, while [13] also provides the location of the forgery.

Regarding the forgery localization property, it is worth noting that the techniques presented in [11] and [146] allow not only to detect the copy-moved regions, but also to distinguish them from the source patches used to perform the attack. This property is useful in real forensic scenarios, in which it is important to understand the semantic aspects of an image manipulation.

A further interesting feature of [11] is the adoption of a GAN network to generate increasingly hard-to-detect forgeries, that are used to train the discriminator network. This is an original approach to address the problem of data-scarcity that plagues many different existing standard datasets. However, from a performance point of view, it is hard to compare this method to the other ones, as it was evaluated on a custom dataset and not on one of the benchmark datasets. This is not the case for [146], which was evaluated on CASIA2. Note that, even if its accuracy is slightly worse than [97], it has the source plus target localization property mentioned before, while the latter gives as output only a global classification on the image.

2.8.1.2 Splicing and copy-move detection methods

These techniques fit the best in a general application context, in which the type of attack is not known *a priori*, so it is better to cover as many attacks as possible. In particular, we consider the methods tested on CASIA2, which is likely the most significant dataset for copy-move and splicing detection evaluation, both for its sheer size and for the various applied post-processing operations.

Among the methods that we discussed, the one presented in [118] obtained the best overall accuracy. It also gives as output the localization of the forged areas, which as we mentioned is of course relevant in many application contexts. Looking at its forgery detection pipeline, it features both a pre-processing stage, in this case based

on YCbCr space conversion and DCT compression, as well as a post-processing phase that through further features extraction allows to perform localization. Therefore, the good performance that it achieved indicate that an exclusively end-to-end deep learning model, without any pre-processing or post-processing, could be indeed a sub-optimal choice for the task of forgery detection.

On the same note, another comment can be made about the method in [97]. Even if its performance are worse than the others in terms of accuracy, the proposed approach is quite interesting because it involves a "shallow" deep learning model. This allows reducing not only the number of network parameters (and consequently the training time), but also the risk of over-fitting. This idea is in contrast to the common trend in computer vision to use ever deeper networks to achieve high accuracy on specific datasets, that usually cannot be achieved on slightly different ones, which is a clear indicator of over-fitting issues.

A remark should be made on the approach proposed in [29]. This method has a wide applicability even outside the field of forgery detection. In fact, the possibility to extract the noise camera pattern and suppress the high-level scene content of a target image is of great utility in other forensic scenarios as well as for sophisticated camera-specific denoising applications. It is important to also note that the authors evaluated the performance of their algorithm on different datasets, which contain a wide set of forgery attacks such as copy-move, splicing, inpainting, GAN-synthesized content, face-swap, *etc.*, thus proving its wide applicability and robustness. Still, it would have been interesting to have the detection results on other more classic benchmark data, such as the CASIA2, thus allowing a better comparison with other existing methods.

2.8.1.3 DeepFake detection methods

In Table 2.6, the performance of DeepFake detection methods are reported.

As it can be observed from the table, there is not a method that performs better on all three considered benchmark datasets. The method proposed in [55] exhibits remarkable performance on DFDC, with an accuracy of 95.9 %, while the best results

Table 2.6: DeepFake detection methods performance.

Method	DFDC acc.%	FaceForensic++ acc.%	Celeb-DF acc.%
[125] (a)	49.9	**99.7**	48.2
[125] (b)	72.2	**99.7**	65.3
[125] (c)	69.7	95.5	65.5
[101]	53.3	96.6	57.5
[86]	75.5	93.0	64.6
[39]	81.0	94.0	**97.0**
[55]	**95.9**	94.2	-
[150]	-	97.3	-

on FaceForensic++ are achieved by the approaches in [125](a) and [125](b). It must be considered, though, that FaceForensic++ was built by the same authors of [125] (all three versions). As such, it is, to some extent, expected that these are the methods that perform better on that particular dataset. The approach in [39] outperformed all other methodologies on Celeb-DF, by a great margin. Moreover, this method is the one that best generalizes over the three considered datasets, with a discrete accuracy of 81 % and 94 % on FaceForensic++, respectively.

One important observation that can be drawn from the this table is that Celeb-DF remains the most challenging dataset. Despite the significant efforts made by various state-of-the-art methods, only one of them achieved satisfactory performance on Celeb-DF. On the other hand, all methods considered in this analysis attained accuracy scores exceeding 90% on FaceForensic++. This suggests that FaceForensic++ is relatively less challenging than Celeb-DF, and future studies should aim to compare methods on more difficult datasets.

2.9 Discussion

In this chapter we provided a survey of some of the recent AI-powered methods (from 2016 onward) for copy-move and splicing detection that achieve the best results

in terms of accuracy on the standard benchmark datasets. Several reviews and surveys have been published on this topic, but most concerned mainly traditional approaches like those based on key-points/blocks, segmentation, or physical properties. Instead, we focused our analysis on recently published, deep learning based methods, because they have been shown to be more effective in terms of performance and generalization capability than the traditional approaches. As a matter of fact, they are able to achieve really high accuracy scores on the benchmark datasets.

We separated the performance analysis between copy-move only, both copy-move and splicing, and DeepFake detection methods. In the case of copy-move only detection, the method in [41] shows an almost perfect accuracy on all three standard benchmark datasets (MICC-F220, MICC-F600, and MICC-F2000). The technique presented in [13] is able to achieve a similar accuracy, while also giving the identification of both the copied regions and the original ones used as source for the attacks. In the case of both copy-move and splicing detection, similar results were achieved on the CASIA2 dataset. In particular, method [118] shows the best accuracy and gives the localization of the forged regions as well.

Concerning DeepFake detection, from the reported performance (see Table 2.6) we infer that there is not a clearly winning approach, in particular no method is general enough for different kinds of DeepFake content. However, we can conclude that the XceptionNet-based models proposed in [125] are able to achieve better performance on at least two out of the three considered benchmark datasets.

From a general point of view, it can be easily inferred from the DL-based methods surveyed in this analysis that a clear trend has not yet emerged. Most works have been more or less independently proposed, in the sense that the vast possibilities offered by DL architectures are still being explored, without a clear winning strategy indication. Nonetheless, we showed that, in the case of splicing and copy-move detection methods, the best accuracy scores were obtained by the techniques that involve some form of pre-processing and post-processing in addition to a deep learning network. For this reason, we think that this appears to be the most promising approach, and so we

believe that further research should be conducted on algorithms that combine deep learning approaches with traditional techniques from all over the field of (statistical) signal processing.

As a further consideration, it can be noted that in the case of techniques aimed at "classic" forgery detection (splicing and copy-move), most of state-of-art methods are able to achieve good performance (on different datasets). Instead, this is not the case for newer challenges like DeepFake detection, whose methods report accuracy performance which is still not satisfactory on complex datasets, like Celeb-DF. As such, further research efforts and ideas still need to be explored in this particular direction.

Further remarks are in order on the problem of performance evaluation of deep learning based methods. Different authors built custom datasets or merged different ones in order to train and test their algorithms. While this can be a solution to overcome issues of data-scarcity (over-fitting), it makes the comparison with other methods more difficult, or even impossible. Even when the same dataset is used to evaluate different approaches, the authors do not always specify which and how many images were used as testing set.

This problem could be addressed by building a custom dataset for training, and using one or possibly more benchmark datasets in their entirety for testing. In this way, not only it would be possible to easily compare different deep learning based approaches, but also to compare them to traditional, non-learning based ones.

Of course, building a custom dataset with thousands of images, with realistic forgeries and post-processing operations on the forged areas, such as blurring, JPEG-compression, smoothing, and so on, is not a simple undertaking. For this reason, we point out that another possible future research direction could be the automation of this task, for example by leveraging a GAN network (as done in [11]), or encoder-decoder models such as a Unet.

A wholly different comment on the subject of datasets building should also be made on the meaning of the forgery attacks currently contained in the benchmark datasets. As these have always been generated in a laboratory environment (whether

manually or not), they typically contain copy-move and splicing attacks that hardly bring a particular semantic value to the altered images. For example, when a tree is copied and pasted in a wood landscape, or a cloud is pasted into a blue sky, the obtained image could hardly be used for malicious purposes. This is clearly not the case for many manipulated images that can be found on the Web. Let us consider for example the splicing shown in Fig. 1.2: the fact that the 2004 presidential election candidate John Kerry was (falsely) immortalized together with pacifist actress Jane Fonda, who was viewed by many as an anti-patriotic celebrity, could have seriously influenced the elective campaign (in this case, the image was shown to be false, but not quickly enough to avoid some damage to the candidate's reputation).

Of course, in such real-world cases, the context adds a lot to the meaning of the forgery, and thus it can hardly be taken into account by a forensic tool without a human supervision. Nevertheless, we feel that it could be interesting to build a database that collects more realistic, manually made, "in-the-wild" forgeries, like the ones that routinely spread on social media these last years, and so present potentially malicious attacks from a purely semantic point of view. Also, this database should contain, for each forgery, the associated ground-truth mask, in order to better assess and compare the forgery localization capability of the forensic tools.

We would like to conclude adding a final, more philosophical observation. As is typical in the case of security-related fields, attackers usually embody, in their attacks, ideas and "hacks" that are specifically designed to counterpoise the latest state-of-art detection methods, *e.g.*, so-called *adversarial attacks* [54, 80, 44], which are used to fool deep learning classification systems. For example, a possible strategy to achieve this confusion consists in using a certain CNN architecture as a discriminator in a GAN model, in order to produce synthesized content which is, by construction, hard to be detected as fake by that particular CNN. Another interesting example of this kind involves DeepFake detection: in [85], the authors observed that, in DeepFake videos, it was common to see unnatural eye-blinking pattern (or no blinking at all), because DeepFake generation algorithms were trained mostly on pictures of people with

open eyes. As expected, attackers immediately adapted DeepFake methods in order to generate realistic eye-blinking, either by including pictures of people with closed eyes during training, or by synthetically correct this issue altogether.

As a consequence, it is probably an illusion to consider a certain forgery detection method to be "safe" forever, even if it has been shown to achieve great detection accuracy on different datasets. For this reason, we think that continuous research efforts should be made in order to develop methods that can, at least to some extent, keep up with the attackers' pace in developing more and more sophisticated and hard-to-detect forgeries. One possible strategy, that tries to anticipate potential attacker moves, could be to actively implement new forgery techniques while developing detection algorithms, this way understanding and leveraging their flaws and thus to allow the creation of possible counter-measures.

Chapter 3

COPY-MOVE DETECTION USING SIFT KEYPOINTS MATCHING

Copy-move attacks, because of their simplicity of implementation, are more common compared to other forgeries. On the contrary, detecting these manipulations can be very difficult due to the transformations that might be done on source objects in the images.

Scale Invariant Feature Transform (SIFT) [94] is a method designed to extract distinctive invariant features from images. The invariancy of these features to scale and rotation makes them suitable to be used in copy-move detection, especially in scenarios in which post-processing transformations are used to disguise the tampering. The interested reader is referred to Chapter A for an in-depth description of the SIFT algorithm.

In this chapter, we describe a copy-move detection approach we developed that is based on SIFT keypoints matching and density-based clustering. Even if keypoints-based copy-move detection methods have been proposed before (the most famous being the work by Amerini et al. [14]), we think that our idea of combining it with a density-based clustering and filtering can be regarded as an important contribution in this field, as this allowed our approach to outperform the method in [14] on two benchmark copy-move dataset.

This chapter is organized as follows: first, an overview of the proposed approach is given and its main computational blocks are detailed, with a particular focus on the density based-clustering step and the related filtering. Then the experimental results on two benchmark datasets are reported, including a performance comparison between our approach and the one by Amerini et al.

3.1 Copy-move detection method

An important property of SIFT features is the rotation and scale invariance. Each of these features is characterized by a distinctive descriptor. These attributes make SIFT features suitable for detecting similar objects or regions within an image, which is the main challenge of copy-move detection. In [14], the authors detected copy-moved areas by extracting SIFT features and matching them based on the descriptors' distances. Then, hierarchical clustering is performed to group spatially close keypoints in order to estimate the transformation matrix between the source and target regions involved in the copy-move attacks.

In this work, we tried to improve their approach by modifying the clustering step and adding further post-processing in order to filter out weak keypoints matches. In particular, we used the DBSCAN algorithm to cluster the keypoints based on their spatial coordinates. The advantages of using this clustering algorithm are (i) the fact that we don't need to specify the number of clusters and (ii) its intrinsic capability of identifying outliers/noise points. This was crucial in our approach to filter out matches between isolated keypoints.

The steps involved in our method are outlined in Fig. 3.1. The SIFT keypoints extraction is explained in Chapter A, and the other three steps are described below.

3.1.1 Descriptors matching

This step aims to find the most similar couples of descriptors (D_i, D_j). The idea is that the distance between descriptors of keypoints belonging to copy-moved areas should be considerably smaller than in the case of pristine areas. The reason for this is that SIFT descriptors are, by construction, a compact description of the neighborhood of a keypoint, that is (to a certain extent) scale and rotation invariant. As such, similar regions - in particular copy-moved areas - will have keypoints with similar descriptors (supposing that keypoints are found in both regions).

The distances between descriptors are usually computed either with a "brute

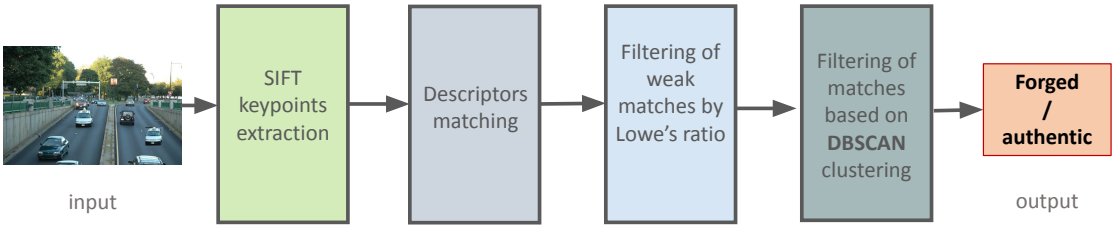


Figure 3.1: General steps of our copy-move detection method based on [14].

force" algorithm or a Flann-based algorithm. The latter is faster, but it is not deterministic, and, as such, some matches can be missed. Let us define the distance matrix between descriptors as:

$$Dist(i,j) = d(D_i, D_j) = d(D_j, D_i) \quad \in \mathbb{R}^{N \times N} \tag{3.1}$$

where N is the total number of descriptors and the distance function $d(.,.)$ is the Euclidean distance. As each element of the matrix can be computed independently, we wrote a parallelized version of the computation that can be run on the GPU, achieving a speedup of 30 to 50 times with respect to the non-parallelized version.

3.1.2 Filtering with Lowe's ratio

For each matching candidate, $M_{i,j} = (D_i, D_j)$, if the following inequality holds, the match is kept. Otherwise, it is discarded.

$$d(D_i, D_j) < \alpha \cdot (D_i, D_k), \quad \alpha \in]0, 1] \tag{3.2}$$

Where D_k is the second closest descriptor to D_i and α is the Lowe's ratio constant. This constant, and other parameters of our algorithm are optimized by means of a grid search (more details are given in Section 3.2).

3.1.3 DBSCAN based filtering

As mentioned before, the DBSCAN has the advantage of not requiring the number of clusters that need to be detected. Also, this algorithm is able to detect outlier points automatically, *i.e.*, points that should not be assigned to any cluster. This property was particularly useful in our approach, as it allows to filter out matches between isolated keypoints.

Let us consider the example copy-moved image shown in Fig. 3.2. The idea we leveraged for this filtering step is the following: if a meaningful region (such as a patch containing an object) is used as a "source" for a copy-move attack and contains enough keypoints, these will probably be spatially close and it will be possible to assign them to a unique cluster (*e.g.*, C1 - pink colored points). Similarly, the keypoints of the corresponding "target" region will be clustered together (*e.g.*, into C2 - green colored points). In this case, if the descriptors' matching step is successful, cluster C1 is correctly "mapped" to cluster C2 and vice-versa.

In the case of weak matches between keypoints, instead, the keypoints in C1 (or C2), even if clustered together, will probably be mapped to a set of different clusters or to outliers/noise points (red crossed). Also, each keypoint identified as an outlier (and the corresponding matched keypoint) should be discarded (blue-crossed).

As mentioned before, we used DBSCAN algorithm for clustering [43] instead of hierarchical clustering for two main reasons: first, it does not require as input the number of clusters to be found (such as in K-means algorithm) and second, it is able to recognize outliers/noise points.

We will now describe, in detail, the filtering algorithm:

1. Perform DBSCAN clustering on the keypoints belonging to valid matches (some of them have already been filtered out with the Lowe's condition). Note that the

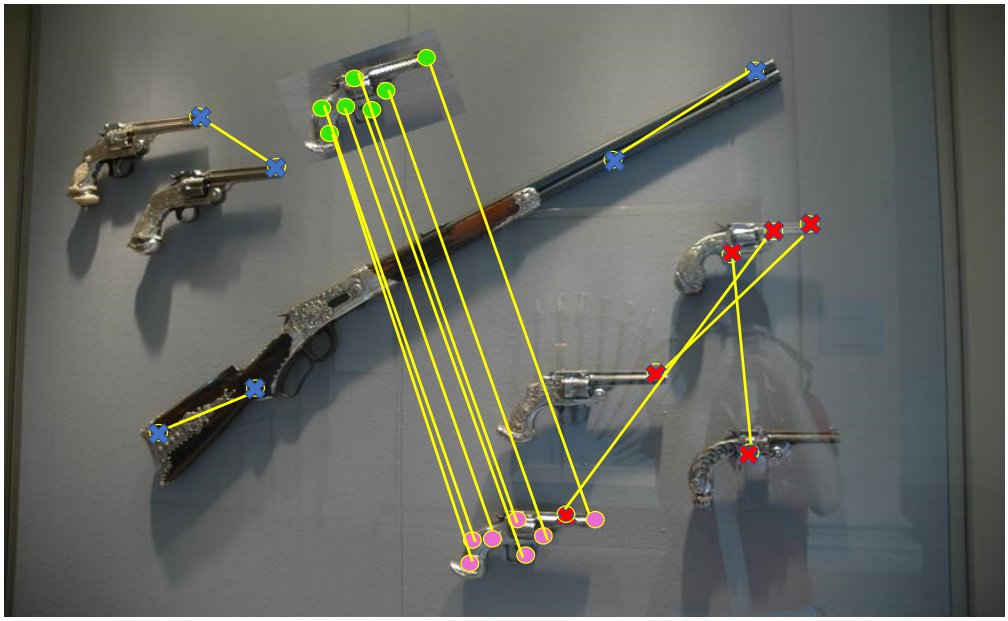


Figure 3.2: DBSCAN-based filtering. The following kind of weak matches are filtered out: (i) matches involving outliers keypoints (blue crossed) and (ii) matches that are scattered to multiple clusters/noise (red crossed).

clustering is done on the spatial coordinates (x, y) of the keypoints, not on the associated descriptor vectors.

2. Consider a valid match $m = (k_i, k_j)$.

3. If either k_i or k_j is labelled as noise m is discarded.

4. If either k_i or k_j belongs to a "small" cluster (whose cardinality is under a threshold T) m is discarded.

5. If m is discarded, both k_i and k_j are labelled as non valid. The cardinality of the clusters to which the keypoints belonged is decreased.

6. Repeat steps 2-5 until all the matches are considered and no more matches are discarded.

7. Finish.

Note that our copy-move detection approach depends on a set of hyper-parameters and thresholds, which should be fine-tuned to obtain the best performance. These parameters are:

- α: parameter for Lowe's ratio test (see Section 3.1.2).
- ϵ: DBSCAN parameter used to define each point's neighborhood size.
- N_s: minimum cluster size; each cluster containing less than N_s points is discarded and its points are labeled as noise.
- β: this parameter is used to filter out matches whose keypoints belong to the same cluster and the distance between the associated descriptors is too high: $d(D_i, D_j) > d_{min} + \beta(d_{max} - d_{min})$, where d_{min} and d_{max} are the minimum and maximum distances between all the extracted descriptors, respectively.

3.2 Experimental results

In this section, we present the detection results of our proposed method on three benchmark datasets for copy-move detection evaluation: MICC-F220, MICC-F600, and MICC-F2000 (See Section 2.3). Subsequently, we compare the performance of our method with the one proposed in [14] (Amerini et al.). We chose this method as baseline for comparison as it is one of the most cited and well performing in the category of keypoints-based techniques and also because our algorithm shares the same initial steps of keypoints computation and Lowe's filtering. As such, we wanted to demonstrate the positive effect in terms of performance of both our successive filtering strategy and density-based spatial clustering.

Similarly to what Amerini et al. did in their work, we adopted a procedure to optimize the parameters of our algorithm. In particular, they used a k-fold strategy (with $k = 4$ - three folds for training and one for testing).

We instead used a "partial" k-fold training and validation procedure, which can be summarized as follows:

1. from the considered dataset D, k non-overlapping subsets D_i, $i = 1, .., k$ are randomly extracted, each with fixed cardinality N.

2. The parameters of our algorithm are optimized on each subset D_i by means of a grid search procedure that maximizes the F1 score. We denote as $\boldsymbol{\theta}_i^*$, $i = 1, .., k$ the best parameters for each subset.

3. Each set of parameters $\boldsymbol{\theta}_i^*$ is tested on the rest of the dataset $D \setminus D_i$ and the metrics of interest are computed for each of these runs: F1_i, Acc_i, Prec_i.

4. The average between the metrics of all k runs are computed, obtaining F1-avg, Acc-avg and Prec-avg.

As mentioned before, the described parameters selection and validation procedure can be thought of as a partial k-fold validation. The reason is that the k subsets don't cover the whole dataset: $\bigcup_{i=1}^{k} D_i \subset D$. We couldn't use a complete k-fold validation procedure as the search space for our parameters is too big - 4 independent parameters need to be optimized. As such, the optimizing procedure takes several hours to run on a subset of just 50 images. Hence, we used this partial k-fold validation in which we fixed both the number of subsets and the cardinality N of each subset. In particular, for MICC-F220 we fixed $N = 44$ and $K = 5$, while we set $N = 80$ and $k = 4$ for MICC-F2000 and MICC-F600.

In Table 3.1 the obtained averaged accuracy, precision, and F1 score on the three mentioned datasets are reported, while in Fig. 3.3, Fig. 3.4 and Fig. 3.5 the average confusion matrices are shown (each entry TP, TN, FP, FN of the matrix was averaged between the 4 runs). These results demonstrate the high performance of our approach.

In Table 3.2 a comparison of the (average) F1-score obtained with our method and the one by Amerini et al. on MICC-F220 and MICC-F2000 is shown. Note that (i) in the table we used the F1 values reported in the original paper and (ii) the results on MICC-F600 were not present in [14]. Since our method shares the first steps (keypoints

	MICC-F220	MICC-F600	MICC-F2000
avg. Acc.	0.943	0.908	0.933
avg. Prec.	0.924	0.787	0.895
avg. F1	0.943	0.840	0.906

Table 3.1: Performance of proposed method on MICC-F220, MICC-F600 and MICC-F2000.

	MICC-F220	MICC-F2000
Amerini et al.	**0.960**	0.869
Proposed	0.943	**0.906**

Table 3.2: Comparison of avg. F1 score between Amerini et al. [14] and proposed approach.

extraction and matching) of the algorithm by Amerini et al, and they obtained state-of-the-art performance detection on these datasets, it is somewhat expected that our approach also performs well. While the obtained F1 was slightly worse on the MICC-F220 dataset, our approach was able to improve the F1 by a margin of 3.5% on the MICC-F2000 dataset which, comprising 2000 images is more general and representative than MICC-F220 (which only has 220 images). Also, we recall that Amerini et al. used a k-fold procedure with each subset having a dimension of three-quarters of the original dataset. This means that they trained their algorithm on sets of 1500 images on the MICC-F2000 dataset, while we only used sets of 80 images. This means that our algorithm is more robust in the sense that way fewer images are needed to fine-tune its parameters.

3.2.1 Future work

As can be seen from the results reported in the previous sections, our method achieves good performance on three benchmark datasets. Still, especially on MICC-F600 and MICC-F2000, there is room for improvement.

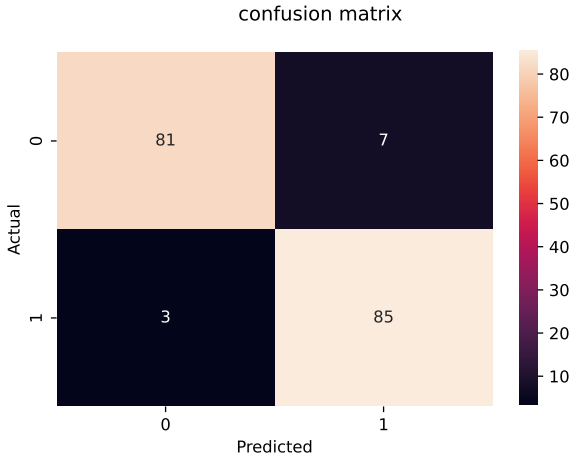


Figure 3.3: Confusion matrix of proposed copy-move detection approach on MICC-F220.

We plan to further extend our method to achieve the following goals:

1. reducing the number of false positives by adding further filtering based on the DBSCAN clustering results.

2. Adding a localization step, used to precisely segment the suspected copy-moved areas, also differentiating between the source and the target regions.

3. Reducing the number of parameters of the algorithm, trying to automatically fit them to the data.

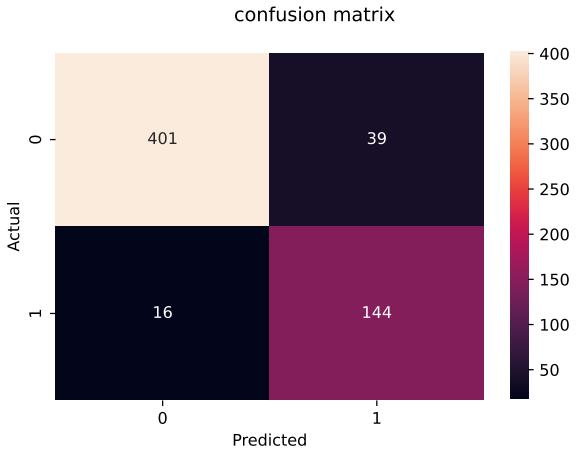


Figure 3.4: Confusion matrix of proposed copy-move detection approach on MICC-F600.

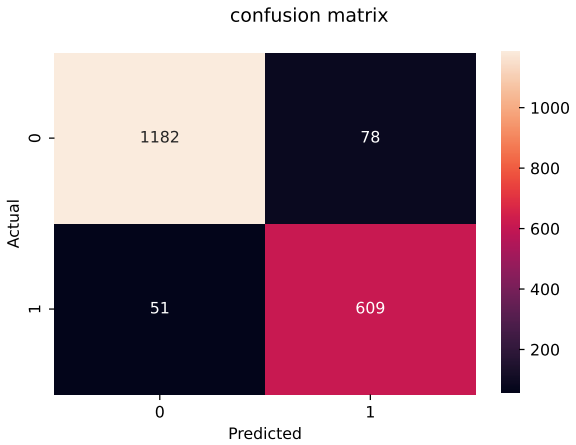


Figure 3.5: Confusion matrix of proposed copy-move detection approach on MICC-F2000.

Chapter 4

LIGHT DIRECTION ESTIMATION

In computer graphics (CG), a 3D scene, comprising meshes, normals, lighting, and material properties, is projected to a 2D plane (the screen image) by means of a *rendering pipeline*, which is based on the *pinhole-camera model*. This is the equivalent of the physical process that takes place when a picture of a real-world scene is acquired with a traditional 2D camera. clearly, part of the original information is lost in the process, as we are projecting elements from a 3D space to a 2D space. The task of understanding the inherent light conditions of a 3D scene from single images has gained a lot of attention for numerous applications, such as virtual/augmented reality, computer vision, and robotics. For example, if we want to visualize a CG building on top of a real panorama picture, the image lighting should be estimated in order to render the building in a realistic way. The task of light source estimation relates to the general problem referred to as *inverse rendering*. In this case, a set of orthogonal components of the 3D depicted scene, such as surface normals, light color and direction, albedo, and material properties are estimated from the 2D image. The inverse rendering problem, even when limited at the identification of the light source direction, is an *ill-posed* problem because, given an image, there are potentially infinite 3D scenes (with associated lighting conditions) that can produce the same 2D image through a perspective projection. In order to deal with this problem and reduce the dimensionality of the research space, prior knowledge and/or simplifying assumptions about the content of the image are usually employed, such as:

- the image contains a person or a face;
- the image is taken in an indoor/outdoor environment;

- there is a known number of light sources;
- some of the objects in the scene have known geometry/orientation/material properties. An example is when a sphere, with known reflectance properties, is inserted in the scene that is being captured, to easily infer the light direction based on the position of the spotlight on the sphere itself.

In recent years, a growing number of researchers have addressed the problem of outdoor light estimation, and have proposed both model-based and data-driven approaches. Recently, with the advent of Deep Learning (DL), several methods have been proposed that leverage this technology. Despite the fact that they have proven to achieve good performance (see Section 4.1), both for the task of light direction estimation and sky dome reconstruction, further improvements partially depend on the availability of a great amount of annotated data, as for the majority of supervised methods. Even for domain experts, it is very difficult and time-consuming to precisely annotate a large amount of 2D outdoor images with 3D light direction. In this regard, only a few datasets have been proposed in the literature to train and evaluate deep learning methods for outdoor light source estimation (see Section 4.1.3). Also, they suffer from some limitations, such as:

- limited dimension;
- lack of realism when CG generated;
- ground truth not available or incomplete.

In this chapter, we propose a framework to predict, from a single outdoor image, the corresponding 3D light direction. The key idea of our method is to combine a data-driven approach, such as a convolutional network, with a physical illumination model. Through this model, we manage to embed geometrical information about the scene that contributes to improving the light prediction accuracy. Hereafter we refer to our framework as *Fusion*.

In order to properly train and test our architecture, we developed two methods, that can be used to generate big datasets of realistic images, annotated with 3D light direction. The first one allows the creation of a dataset of CG-rendered images starting from a given outdoor 3D scene, with a great number of lighting conditions. The second one can be used to extract a dataset of limited field of view (FOV) images from 360 degrees panoramas in latitude-longitude format. With the first method it is possible to generate a CG dataset, hereafter referred to as *SynthOut*. The second one, instead, is used to generate images of limited FOV from spherical images. In particular the Laval High Dynamic Range (HDR) outdoor database [59] has been used to produce a dataset hereafter referred to as *RealOut*.

First, we trained and tested our *Fusion* network on *RealOut*. Then, in order to assess the generalization capability of our approach, we fine-tuned and evaluated our model on *SynthOut*, as well as on two other benchmark CG public datasets: SID2 [131] and VIDIT [40]. In a direct comparison, we show that our *Fusion* model outperforms a previously proposed state-of-art method on SID2, in terms of average angular error between the real and the estimated light direction vectors.

In order to evaluate the improvement introduced by the use of the physical model and the surface normal prediction in the light estimation task, we conducted an ablation study, in which we compare the performance of the *Fusion* architecture and the same network where the surface normal prediction branch and the physical illumination model have been removed, proving the superiority of the *Fusion* approach.

The remaining part of this chapter is organized as follows: in Section 4.1 we give an overview of the main contributions in the field of outdoor light estimation. In Section 4.2 we describe, in detail, our light estimation architecture, while in Section 4.3 we present the algorithms we developed to generate both our *SynthOut* and *RealOut* datasets. In Section 4.4 we discuss the performance of our deep architecture on *SynthOut*, *RealOut* and on two other CG benchmark datasets. In this section, we also present the results of the ablation study to evaluate the contribution of surface normal estimation in the proposed light estimation pipeline. Finally, in Section 4.5 we draw

some conclusions.

4.1 Related work

In this section, we review some of the research efforts made in the field of light source estimation in the specific case of outdoor scenes. First, we introduce two widely cited and known physical models, that describe the appearance of the sky hemisphere and the sun (in the daytime), in terms of luminance. These models take as input a set of parameters, such as light and view direction, sky turbidity, light, and sky color, extracted from a 2D panorama/image and they generate the corresponding 3D descriptions of the sky. They are really relevant because they are widely used both in CG and in conjunction with DL methods to estimate 3D light direction from 2D images (some of them will be presented in Section 4.1.2). Finally, we give an overview of some of the benchmark datasets commonly used for training and evaluating light estimation approaches.

4.1.1 Sky and sun models

One of the first important contributions was brought by Hosek L. and Wilkie A. in [62]. They proposed an analytical, physics-based model to describe the HDR luminance of the sky hemisphere at daytime, which can be represented by the following equation:

$$L_\lambda(\mathbf{l}) = f(\theta, \phi, t, \sigma_g) \cdot L_{M,\lambda}, \tag{4.1}$$

where:

- $f(.)$ is a function of several parameters (described as follows), and it is independent of the wavelength.
- $\mathbf{l}$ is a vector representing the 3D view direction in the sky hemisphere, Ω;
- $L_\lambda(\mathbf{l})$ is the spectral radiance along the view direction $\mathbf{l}$;
- θ is the angle between view and sun direction;

- ϕ is the angle between the view direction and a vector pointing to Zenith.
- t is a scalar value that models the sky turbidity;
- σ_g is a scalar value that models the ground albedo;
- $L_{M,\lambda}$ is the expected value of radiance in a point randomly chosen in the hemisphere Ω. Note that this term is wavelength-dependent.

This model is an improvement on the one previously proposed by Perez et al. in [109] and by Preetham et al. in [115], as it introduces the following advantages:

- it handles each spectral component independently (through the term $L_{M,\lambda}$);
- it allows to better model specific sky conditions such as sunsets;
- it allows to model the effect of the ground albedo, which was ignored in previous sky models.

A key aspect of this model is that its parameters can be fit to Low Dynamic Range (LDR) panoramas to extrapolate an HDR sun description. This is possible because the sun's appearance is embedded in the sky-dome modeling through the turbidity parameter.

Another physical approach to describe outdoor lighting conditions was introduced by Lalonde and Matthews in [82]. In particular, the authors employed two separate components in order to better describe the sky and the sun's appearance under various conditions. The Lalonde-Matthews model (LM) can be described by the following equation:

$$f_{LM}(\mathbf{l};\mathbf{q}_{sun},\mathbf{q}_{sky},\mathbf{l}_{sun}) = f_{sun}(\mathbf{l};\mathbf{q}_{sun},\mathbf{l}_{sun}) + f_{sky}(\mathbf{l};\mathbf{q}_{sky},\mathbf{l}_{sun}), \tag{4.2}$$

where:

- $\mathbf{l}$ and $\mathbf{l}_{sun}$ represent the view and sun direction, respectively, expressed as spherical coordinates;

- $\mathbf{q}_{sun}$ is a set of parameters controlling the sun shape and color;
- $\mathbf{q}_{sky}$ is a set of parameters controlling the sky turbidity and color;

Note that, in order to use this model, a set of 11 parameters (comprising the sun direction, $\mathbf{l}_{sun}$) must be estimated. An advantage provided by this model, compared to the one proposed by Hosek and Wilkie (HW model), is its greater expressiveness, as the sun and sky components are described independently through the $f_{sun}(.)$ and $f_{sky}(.)$ functions, respectively. Besides, a limitation of this approach is that, in contrast to the HW model, the HDR sun parameters cannot be fit directly to LDR panoramas.

4.1.2 Light estimation approaches

In [60], Hold-Geoffroy et al. trained a Convolutional neural network to regress, from a single LDR image, a set of parameters (including sun position, camera field of view, elevation and sky turbidity) to be fed to the Hosek-Wilkie sky model in order to obtain the corresponding HDR sky-dome description. This approach combines the advantages of the HW model, such as having few parameters that can be fit to LDR images, and the ones provided by DL, *i.e.*, its ability to automatically learn the features needed to infer the parameters themselves. This is in contrast to previous approaches, in which the parameters were manually estimated or picked from fixed tables.

In [59] an end-to-end deep learning approach was proposed to solve both the light modeling and estimation problems. In particular, the authors trained a deep encoder-decoder model (ED) on the Laval HDR sky database [81] to learn a compact representation $z \in \mathbb{R}^{64}$ of HDR sky panoramas (this is also called latent space representation). Then, they used the approach in [152] to convert the panoramas in SUN360 dataset [147] from LDR to HDR. The ED was then employed to extract latent space representations for each panorama in SUN360. Then, they extracted crops from each panorama, creating a dataset of limited FOV images and corresponding ground truth sky latent space descriptions (each crop extracted from a given panorama P is assigned with the same z vector previously predicted from P with the ED model). Finally, they

used this dataset to train two CNNs to predict, from an input limited FOV image, the sky encoding z and the sun azimuth, respectively. The 360-degree sky panorama is then retrieved by giving the predicted z vector as input to the decoder branch of the ED. Note that, in contrast to the method in [60], no physical models were used, as the light information is automatically encoded in the latent space representation. Even if the proposed method outperforms the previous state-of-the-art approaches, it still suffers from some limitations, such as low-quality shadows and texture on the reconstructed sky.

A similar approach to [59] was proposed by Zhang et al. in [153]. First, an encoder-decoder network, referred to as *PanoNet* by the authors, was introduced to estimate the HDR Lalonde-Matthews model parameters from LDR panoramas. This network aims to reconstruct 360-degree panoramas, while at the same time, learning the Lalonde-Matthew model parameters in the bottleneck layer of the network (latent space vector). In order to constrain the network to encode as much lighting information as possible in the latent space, a "rendering loss", measuring the difference between a synthetic scene rendered with the predicted lighting parameters and the real one (ground truth), was added to the basic reconstruction loss (between the input panorama and the reconstructed one). The errors between the estimated LM parameters and the ground truth ones are also directly included in the loss as a separate term. A second network, named *CropNet*, was designed to regress the LM model parameters from limited FOV LDR images (Note that the previously discussed architecture, *PanoNet* could only regress LM parameters from 360-degree panoramas). This network was trained on a novel dataset, that was created as follows:

1. each panorama in the SUN360 dataset was labeled by regressing its LM parameters with *PanoNet*;

2. limited FOV cropped LDR images are extracted from each panorama and the corresponding LM parameters (from the "parent" panorama) are saved.

The trained model, *CropNet*, is then used to predict the LM parameters directly from LDR images and outperforms previous state-of-the-art approaches. However, this approach has still some limitations, such as the fact that it frequently predicts LM parameters that lead to gray skies.

In [131], the authors designed a simple deep learning architecture to predict, from a single outdoor RGB image, both light source direction, expressed as tilt and pan angles (between light and camera direction) and light color (as RGB values). The network architecture involves five sequential blocks, comprising both convolutional and inception layers, followed by three fully connected branches to output pan, tilt and color information, respectively. In order to train their model, the authors leveraged a rendering software (based on *Blender*) to build a dataset, SID2, of 48k+ synthetic images along with the corresponding light source information (source direction and color). The rendered scenes were built by placing objects in the center of surreal rooms with variously textured walls and backgrounds. The authors showed that their approach performed well on the generated synthetic dataset but, when evaluated on real scenes, extracted from the *Multi-illuminant* dataset [19], its performance was not as good.

4.1.3 Benchmark datasets

In this section, we analyze the most cited and important datasets that have been adopted in the field of light source estimation.

- The **Laval HDR** Sky dataset [81] contains a total of 1850 sky-domes pictures. As such, it cannot be directly used to train models to predict light directions in a standard outdoor scene (containing buildings, trees, people, etc..).

- The **Laval HDR Outdoor** dataset [59] is instead composed of 205 HDR high resolution (7768 x 3884) outdoor 360-degree panoramas, in different lighting conditions (with sky varying from sunny to cloudy). The main limitation of this dataset is the fact that it doesn't include any ground truth data regarding the

light direction. As a consequence, the sun's position in the sky must be manually identified, which can be hard in the case of occluding buildings or cloudy sky. Also, the number of panoramas in this dataset is still quite limited to train highly-parameterized deep learning models.

- **VIDIT** [40] is a CG dataset containing 390 (1024 x 1024) outdoor scenes rendered with *Unreal Engine*, each one lit with a total of 40 lighting conditions, combining five possible color temperatures (2500K, 3500k, 4500k, 5500k, 6500k) and eight possible light directions (N, NE, E, SE, S, SW, W, NW), for a total of 15600 images. This dataset includes a great variety of lighting conditions and environments, with a high degree of realism. However, light conditions are quantized and, especially for the task of light direction estimation, having only 8 possible directions is limiting.

- **SID2** [131] is a CG dataset containing 48138 rendered images, with a resolution of (256 x 256) pixels. The depicted scenes contain multiple objects having different shading and reflectance properties. As a consequence, complex light interactions are present. The associated ground truth contains both information about light direction and color. The major limitation of this dataset is the low resolution of the images.

- Finally, the **SUN360** database [147] contains a total of 67583 360-degree real panoramas of both indoor and outdoor scenes. It includes annotations and labels of objects, and consequently, it can be used for general scene understanding tasks, such as object segmentation and recognition. Unfortunately, this dataset is no longer publicly available online. We stress the fact that this dataset was one of the most widely used for the evaluation and comparison of light estimation methods, both for its size and heterogeneity. As a consequence, it has now become hard to compare new light source estimation methods with previous state-of-art approaches, such as [59] or [153].

Considering the limited number of publicly available datasets and their limitations in terms of the number of images and/or the resolutions we decided to design new tools for dataset generation. The methods are described in Section 4.3 and, due to the fact that they will be released as open-source software, in our opinion, these methods are an important contribution to both comparison and development of light estimation methods.

4.2 Light prediction model

In this section, we describe our deep *Fusion* architecture for 3D light direction estimation. The main ideas we leveraged in the design of our method are two:

- the information about the 3D orientation of objects in the target image could help improve the light direction estimation accuracy;
- the use of a simplified rendering model which takes as input the estimated light direction and surface normals of objects can be used as feedback to improve the model performances.

The complete architecture is shown in Fig. 4.1: the target RGB image is fed to the surface normals prediction network. The output of this network, *i.e.*, the surface normals map $\hat{N}$, is concatenated to the input image and fed to the Light estimation network, which outputs the estimated 3D light vector $\hat{\mathbf{L}}$. This vector is given as input to a physical illumination model, along with the currently predicted surface normals map $\hat{N}$, in order to obtain an approximation of the luminance map, $\hat{Y}$, of the input image.

The whole architecture is trained end-to-end through the optimization of a weighted sum of the following two terms:

$$\mathcal{L}_1 = \frac{1}{N}\sum_{k=1}^{N}\frac{1}{3}||\mathbf{L}_k - \hat{\mathbf{L}}_k||_1, \tag{4.3}$$

$$\mathcal{L}_2 = \frac{1}{N}\sum_{k=1}^{N}\sqrt{\frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}(Y_k(i,j) - \hat{Y}_k(i,j))^2}, \tag{4.4}$$

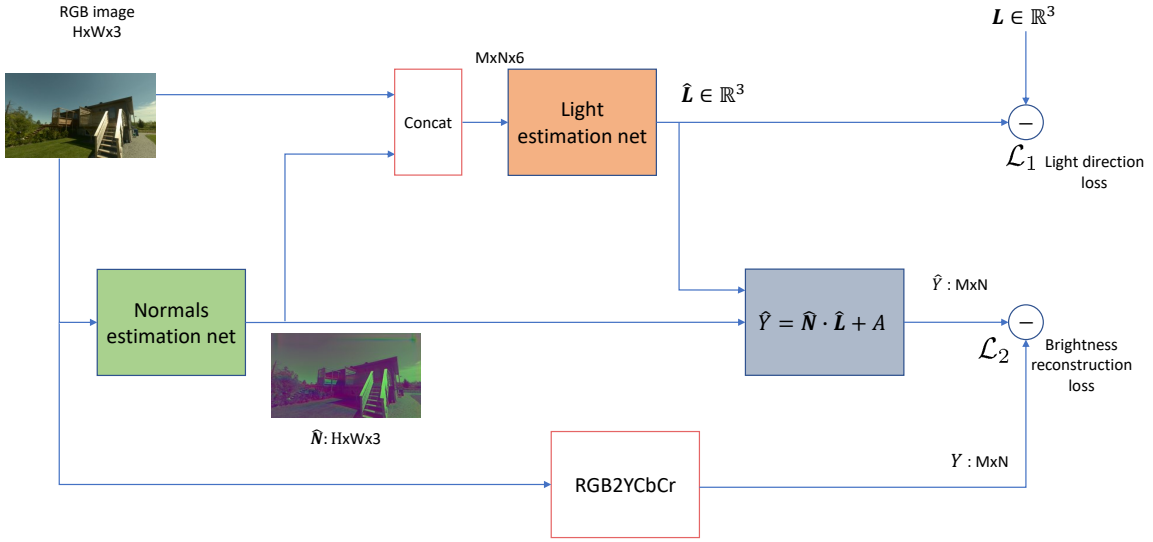


Figure 4.1: Our deep light estimation architecture.

where N is the batch size, and H and W are the height and width of the input image, respectively. $\mathcal{L}_1$ measures the error between the predicted light direction $\hat{\mathbf{L}}$ and the real one $\mathbf{L}$, while $\mathcal{L}_2$ measures the reconstruction error of the luminance channel Y in the YCrCb space: $Y_k(i,j)$ and $\hat{Y}_k(i,j)$ are, respectively, the real and predicted luminance intensity values at pixel (i,j) for image k in a given batch. $\mathcal{L}_2$ is crucial, as it allows to back-propagate the error of the reconstructed luminance map to train the weights of the surface normals estimation network. In this way, the normals estimation network is trained in a self-supervised manner (see Section 4.4.1 for more details).

As mentioned before, the loss used to train our architecture is the weighted sum of the two terms $\mathcal{L}_1$ and $\mathcal{L}_2$:

$$\mathcal{L} = \alpha\mathcal{L}_1 + \beta\mathcal{L}_2, \tag{4.5}$$

where the α and β coefficients are used to weigh differently the two loss terms.

In the next paragraphs, we describe in more detail the processing blocks used in our estimation pipeline: normals estimation network, light estimation network, and physical illumination model.

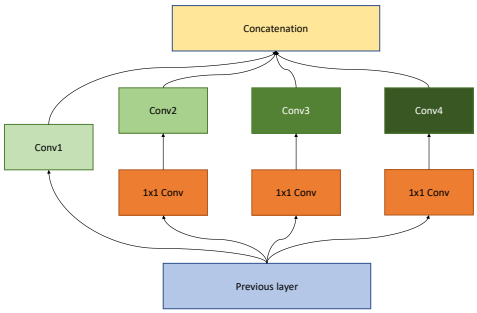


Figure 4.2: Modified inception module used by the authors in [25]. Conv1-4 are Convolutional layers with different kernel dimensions and/or output depths.

4.2.1 Normals estimation net

For the normals estimation network, we used the architecture derived from [10]. This is the implementation of the 3D normals estimation network proposed in [25], which was designed to predict depth fields from single, 2D RGB images, in an indoor environment.

This architecture involves a series of connected modules, each one composed of different inception layers [135] (described in detail in Fig. 4.2). The main advantage of inception layers, compared to standard convolutional ones, is the reduced number of parameters, which allows for building deeper networks, without increasing too much the dimension of the required training dataset and consequently the computational cost. The input resolution is (144 x 256) pixels.

Features at different resolution scales are extracted and propagated through the network by using multiple downsampling layers (Max Pooling), while upsampling layers are employed to construct the output normals map at the same resolution as the input image. Hence, the normals are encoded as a (H x W x 3) matrix. The complete architecture scheme is shown in Fig. 4.3.

4.2.2 Light estimation net

For our light direction estimation network, we adopted a CNN architecture, which consists of 4 convolutional layers, each one followed by a ReLu activation and Max-Pooling downsampling, used to halve the spatial resolution of the input features.

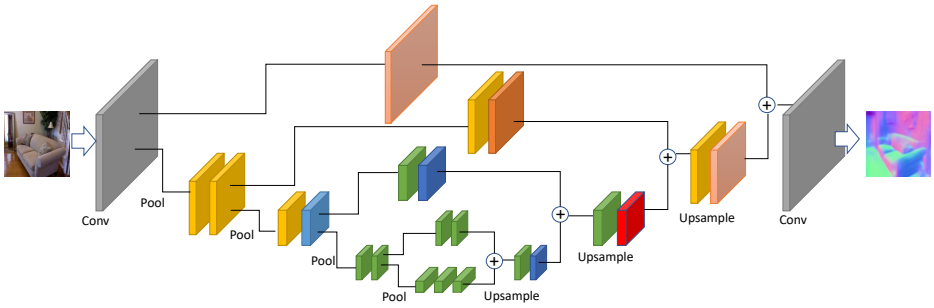


Figure 4.3: Architecture of normals estimation network, as proposed in [25]. The two gray blocks represent convolutional layers, while all the colored ones represent modified Inception modules (Fig. 4.2) with different parameters for the internal convolutional layers.

After the convolutional blocks, we used a global Average Pooling layer in order to eliminate the spatial dimension. This is followed by a dense layer with 64 units, connected to the 3 output neurons, for which we chose a linear activation function. Each of these outputs corresponds to one of the 3D components of the predicted light vector $\hat{\mathbf{L}}$.

The spatial input resolution is (144 x 256) pixels. This is done in order to match the output's dimension of the surface normals map (which is concatenated with the input image).

In order to avoid overfitting, we also employ a Dropout layer between the Dense layer and the output neurons. We chose a probability $p = 0.5$ to drop each neuron in the training phase.

We tested different modalities to encode the light direction prediction, such as $(\sin(\theta), \cos(\theta))$ and $(\sin(\varphi), \cos(\varphi))$ (where θ and φ represent the horizontal and vertical angles of the sun, such as in Fig. 4.8), but the performance was slightly worse than with the three neurons outputs strategy discussed before.

As shown in Fig. 4.4, the light prediction network takes as input both the RGB image and the 3D normal prediction map obtained with the model presented in

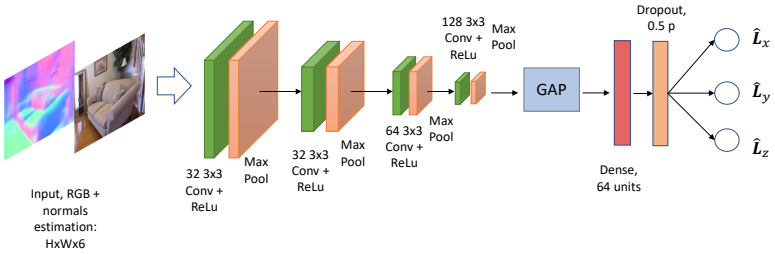


Figure 4.4: Architecture of our light estimation network.

Section 4.2.1.

4.2.3 Physical Illumination model

In this section, we describe the illumination model employed in our architecture with the aim of training our normals estimation network in a self-supervised manner, *i.e.*, without a ground truth of surface normal maps (see Section 4.4.1).

The model is described by the following equation:

$$I_b(i,j) = R(\mathbf{N}(i,j) \cdot \mathbf{L}) + A, \tag{4.6}$$

where:

- $I_b(i,j)$ is the radiance at pixel (i,j),
- $\mathbf{N}(i,j)$ is the 3D surface normal evaluated at pixel (i,j),
- $\mathbf{L}$ is the 3D light vector, supposed constant across the image,
- A represents the ambient light contribution.

The Eq. (4.6) is a simplified version of Phong's empiric illumination model [111]. In fact, here we only consider diffuse light (described by the $\mathbf{N}(i,j) \cdot \mathbf{L}$ term), ignoring

the specular contributions. In addition, we are working under the assumption of a single light source. Also, we further simplify the model by supposing R to be constant for each material in the scene. As a consequence, the only learnable parameters of this model are R and A.

4.3 Dataset generation methods

One of the most widely cited benchmark datasets for evaluating methods for image intrinsic decomposition and, more specifically, light estimation is SUN360 [147]. Currently, this dataset is not available due to IP issues.

Moreover, because it is quite difficult to build high-quality datasets with the correctly annotated light information, most of the existing and available ones are synthetic, and often they don't have complete ground truth data (*e.g.*, VIDIT only has light direction expressed in the form of cardinal points).

For these reasons, we built two new datasets and the corresponding precise ground truth light information. The algorithms we developed to generate them allow for reproducing the exact same data used in this work. The algorithms are released as open source so that it will be possible for other researchers to either generate new annotated datasets or reproduce those used in this work.

4.3.1 Synthetic Dataset

In this section, we describe our synthetic dataset, *SynthOut*, and the algorithm we designed to generate it. This dataset contains a total of 20k RGB CG-rendered images with the corresponding ground truth lighting information.

First, we used the Unity 3D Engine to create a simple outdoor scene composed of realistic objects, such as pans, sofas, f1 cars, chess pieces, glasses, cups, etc. These objects were placed at random locations, with random orientations on a textured tarmac surface. The sky was created with a high-resolution texture (a sunny sky containing some sparse clouds).

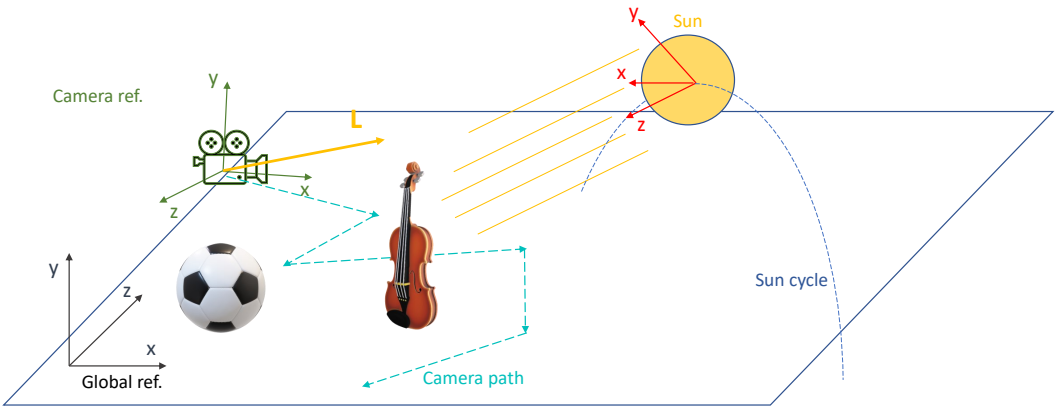


Figure 4.5: Approach used to create our synthetic dataset, *SynthOut*: both the camera and the sun are moved across a fixed path. **L** is the direction of the sun expressed in the camera reference frame.

The 3D scene involves a great variety of shading and reflectance properties, as the objects are composed of different materials, such as metal, wood, and glass, with different opaqueness levels. Some of the objects have transparent shading, as well as specular surfaces.

The only light source present in the scene is the sun, which is modeled as a directional light, assumed to be infinitely far away.

Then, leveraging the C# scripting API available in Unity we created a software application to automatically generate a dataset of rendered images from the 3D scene. Its main components are the following:

- A light controlling module, which is responsible to move the sun across the sky dome, with the purpose of simulating light conditions at different hours in the day. Different RGB colors are randomly chosen for the light as well, with the aim of creating a greater variety of lighting conditions.

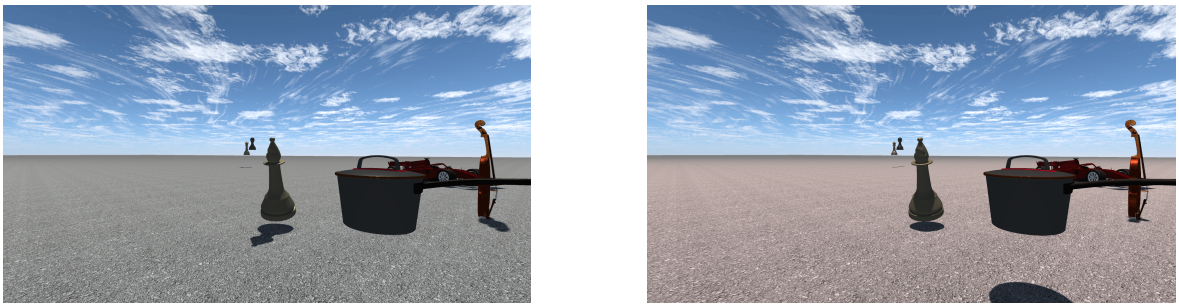
- A camera controller module, which is responsible to change the camera position and orientation, by following a "path" that runs across the scene. In this way, we further augment (i) the range of relative orientations between light and camera and (ii) the variety of combinations of objects present in each captured picture. With this approach, we obtained multiple rendered images of the same objects viewed from different angles and lit with different light conditions (direction and color).

- A script which renders the images viewed from the camera at fixed time intervals. The corresponding illumination conditions are also computed. In particular, the relative orientation of the light with respect to the camera $\mathbf{L} = [L_x, L_y, L_z]^T$ and the color of the light $\mathbf{L}_{col} = [L_R, L_G, L_B, L_\alpha]$ are saved. When an image is rendered, the corresponding segmentation map is also generated, assigning to each object a spatial mask with a different label. In this way, the generated dataset can be used for segmentation and object recognition tasks as well.

- By using a seed, we are able to control a set of pseudo-random parameters, ensuring that each run of our algorithm on the same 3D scene gives as output the same set of rendered images.

In order to generate images with enough different lighting conditions, we designed the sun cycle not to be synchronized with the time needed for the camera to travel across the fixed path through the scene. In this way, the software could iterate through multiple runs on the path and obtain images of the same objects lit with different lighting conditions. An overview of the dataset generation approach is shown in Fig. 4.5, while an example of the same scene under different lighting conditions is shown in Fig. 4.6.

All the images are rendered at a (1080 x 1920) pixels resolution.



(a) (b)

Figure 4.6: Two sample images extracted from our *SynthOut* dataset, depicting the same scene under different light conditions (direction and color).

4.3.2 Real dataset

In this section, we describe the algorithm we designed to automatically generate a dataset of 40k real outdoor images along with the corresponding ground truth light directions. We named this dataset *RealOut*.

Our algorithm is inspired by the work of the authors in [59]. In fact, they also extracted limited FOV images from their Laval HDR Outdoor database, without releasing the produced dataset, nor providing details about how the images were generated.

Note that we do not release *RealOut*, as it is derived (through our algorithm) from the Laval HDR Outdoor database. Nevertheless, the researchers who want to use our dataset can re-create it by requesting the input data from the authors in [59] and running our code.

4.3.2.1 Input panoramas

The Laval HDR Outdoor database [59] contains 205 HDR outdoor panoramas in the latitude-longitude format, with a horizontal view of 360 degrees and a vertical one of 180 degrees, with a resolution of (3884 x 7768) pixels. The dataset involves a wide variety of environments, including parks, forests, buildings, roads, and cars. Several light and sky conditions, from sunny to cloudy are present as well.

4.3.2.2 Sun position estimation

Since no metadata regarding the sun position is available in the Laval Outdoor database, we designed an algorithm (Algorithm 1) to estimate it as accurately as possible from an input panorama I. The idea is to find the brightest connected component in the panorama and identify the sun's position as the centroid of this area.

Algorithm 1 Sun position estimation in panorama

Input: HDR panorama I
Output: Sun position(x_{sun}, y_{sun})

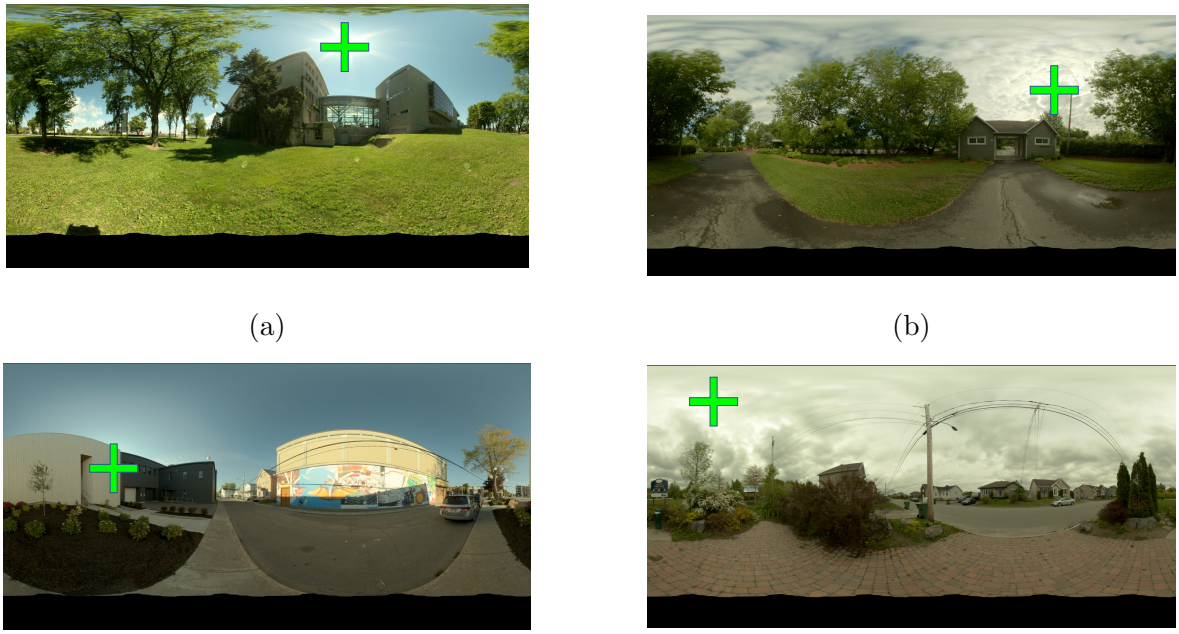
▷ *We work with Luminance channel in YCbCr space*
$Y, Cb, Cr \leftarrow$ **RGB_to_YCbCr**(I)
$M \leftarrow$ **max**(Y)
$Y' \leftarrow \exp(\frac{Y}{M*10})$ ▷ *Augment contrast on light channel*

▷ *Threshold image to obtain brightest regions*
$Th \leftarrow$ **find_percentile**$(Y', 0.999)$
$Y_B \leftarrow$ **bin_thresholding**(Y', Th)

map, stats $\leftarrow$ **connected_components**(Y_B)
$id\_max \leftarrow$ **find_max_area_comp**$(stats)$
white_pixels $\leftarrow$ **where**(map $==$ id_max)

▷ *Compute weighted centroid*
$x_c \leftarrow 0$
$y_c \leftarrow 0$
$sum \leftarrow 0$
for each $(i, j) \in$ white_pixels **do**
    $w \leftarrow Y(i, j)$
    $sum \leftarrow sum + w$
    $x_c \leftarrow x_c + w * j$
    $y_c \leftarrow y_c + w * i$
end for
$x_{sun} \leftarrow x_c / sum$
$y_{sun} \leftarrow y_c / sum$

We found that only in 4 cases out of 205 the estimated sun positions were not correct (from a qualitative point of view). This was due to buildings/objects occluding



(a) (b)

(c) (d)

Figure 4.7: Sun position estimation examples. In cases c and d the position was manually corrected.

the sun or because the sky was too cloudy. In these cases we manually corrected the estimated position (see Fig. 4.7).

The position of the sun in the panorama is then converted into horizontal and vertical angles (see Fig. 4.8), by using the following formulae:

$$\theta_{sun} = \left(1 - \frac{x_{sun}}{W_{pan}}\right) 2\pi \tag{4.7}$$

$$\varphi_{sun} = \left(\frac{1}{2} - \frac{y_{sun}}{H_{pan}}\right) \pi, \tag{4.8}$$

where (x_{sun}, y_{sun}) is the sun position, in pixels, estimated with Algorithm 1 and (H_{pan} x W_{pan}) is the panorama resolution.

4.3.2.3 Limited FOV extraction from Panorama

We developed a ray-casting approach to render a limited FOV portion of a panorama. In order to do that, we leveraged the pinhole camera model, as it is a

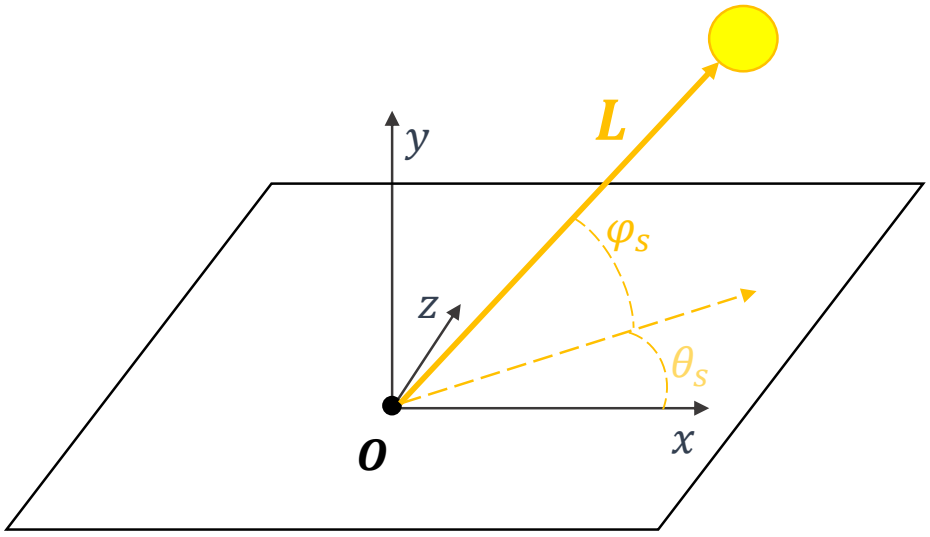


Figure 4.8: Horizontal and vertical angles of sun direction.

standard model in CG. In particular, we modeled the panorama as a texture wrapped around a sphere with a fixed radius ρ. The observer (*i.e.*, the camera) is positioned in the center of this sphere and its orientation is expressed by two angles: θ_c and φ_c, representing the counter-clockwise rotation in the x, z plane, and the elevation, respectively (as in Fig. 4.9).

With this assumption, a point (or a vector) $\mathbf{P}_c$ in the camera reference system can be converted in global coordinates, $\mathbf{P}$ with the following equation:

$$\mathbf{P} = R_{\theta,\varphi}\mathbf{P}_c, \tag{4.9}$$

where $R_{\theta,\varphi}$ is a pure rotation matrix:

$$R_{\theta,\varphi} = \begin{bmatrix} \cos(\theta) & \sin(\varphi)\sin(\theta) & -\sin(\theta)\cos(\varphi) & 0 \\ 0 & \cos(\varphi) & \sin(\varphi) & 0 \\ \sin(\theta) & -\sin(\varphi)\cos(\theta) & \cos(\varphi)\cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

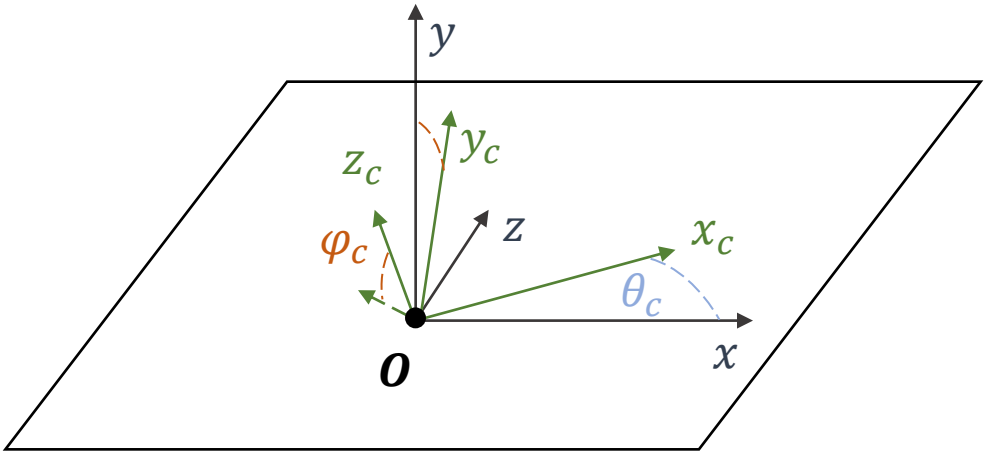


Figure 4.9: Camera and global reference system. the camera orientation is identified by the two angles θ and φ.

Note that both $\mathbf{P}_c$ and $\mathbf{P}$ are expressed as homogeneous coordinates.

The image plane is centered on the z axis of the camera, at a distance of f from the origin (focal length), and it has physical dimensions h by w. If we set the image pixel size equal to H by W, then each pixel will have a physical size equal to $\mu_y = \frac{h}{H}$ by $\mu_x = \frac{w}{W}$.

Each pixel (i, j) in the image plane needs to be assigned the correct source color from the panorama wrapped on the sphere. In order to do that, we cast a ray r from the camera origin $\mathbf{O}$ through the center of pixel (i, j) itself and we compute the intersection between this ray and the sphere. The points where r intersects the image plane and the sphere are $\mathbf{Q}_c$ and $\mathbf{P}_c$, respectively (see Fig. 4.10). $\mathbf{Q}_c$ has the following

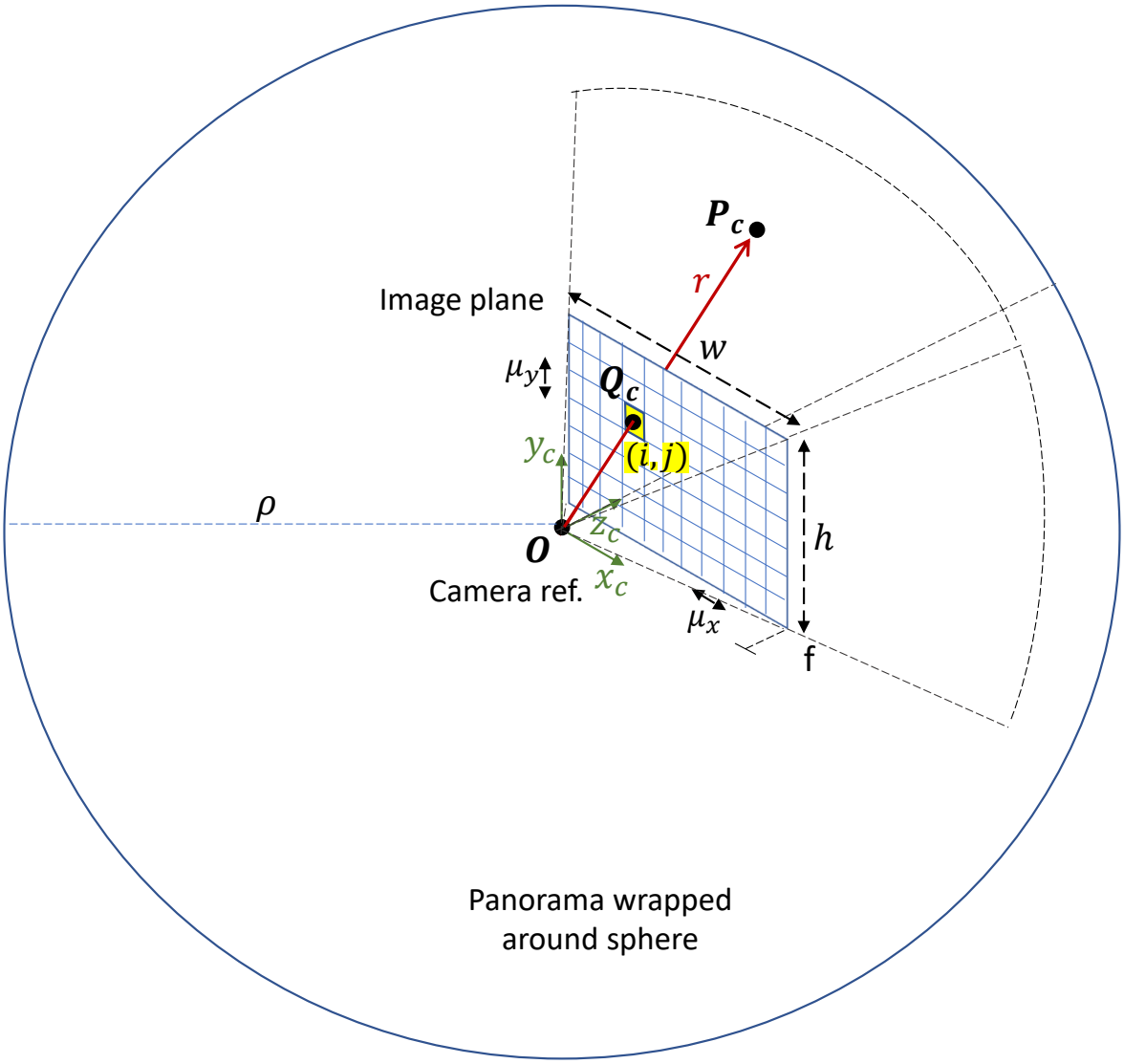


Figure 4.10: Pinhole camera model and ray-tracing approach to rendering a portion of the panorama on the image plane.

coordinates:

$$\mathbf{Q}_c = \begin{bmatrix} \mu_x(j+\frac{1}{2}) - \frac{w}{2} \\ \frac{h}{2} - \mu_y(i+\frac{1}{2}) \\ f \\ 1 \end{bmatrix}, \tag{4.11}$$

while r is described by the following equation:

$$r : \mathbf{S}_c = \mathbf{O} + t\frac{\mathbf{Q}_c - \mathbf{O}}{||\mathbf{Q}_c - \mathbf{O}||}, \quad t > 0, \tag{4.12}$$

where $\mathbf{S}_c$ denotes a generic point belonging to r and t is a positive real value.

The intersection point between r and the sphere can be found by setting $t = \rho$, leading to:

$$\mathbf{P}_c = \mathbf{O} + \rho\frac{\mathbf{Q}_c - \mathbf{O}}{||\mathbf{Q}_c - \mathbf{O}||}. \tag{4.13}$$

We then convert this point from camera to global coordinates through Eq. (4.9), obtaining:

$$\mathbf{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = R_{\theta,\varphi}\mathbf{P}_c \tag{4.14}$$

In order to obtain the corresponding pixel in the panorama, we need to retrieve the horizontal and vertical angles:

$$\theta_{\mathbf{P}} = \text{atan2}(P_z, P_x) \tag{4.15}$$

$$\varphi_{\mathbf{P}} = \arcsin(\frac{P_y}{\rho}). \tag{4.16}$$

The pixel coordinates (u, v) in the panorama can then be computed as:

$$v = \lfloor \left(1 - \frac{\theta_{\mathbf{P}}}{2\pi}\right) W_{pan} \rfloor \mod W_{pan} \tag{4.17}$$

$$u = \lfloor \left(\frac{1}{2} - \frac{\varphi_{\mathbf{P}}}{\pi}\right) H_{pan} \rfloor \mod H_{pan}, \tag{4.18}$$

where (H_{pan}, W_{pan}) is the panorama resolution.

Finally, by denoting as I the rendered image and I_{pan} the input panorama, we can assign each pixel of I as:

$$I(i,j) = I_{pan}(u,v). \tag{4.19}$$

This procedure is repeated by casting a ray through each pixel (i,j) of the image plane.

4.3.2.4 Implementation

In order to generate our *RealOut* dataset, our algorithm iterates through each panorama from the Laval Outdoor Database and, for each one, it renders 200 images, by randomly varying the camera orientation. As for our other dataset generation algorithm, the pseudo-random parameters (such as the camera orientation) are controlled by a seed. In this way, the reproducibility of the generated data is guaranteed.

For each image, we compute the ground truth light direction angles $\varphi_{s,c}$ and $\theta_{s,c}$ as:

$$\varphi_{s,c} = \begin{cases} \varphi_s - \varphi_c, & \text{if } \varphi_s - \varphi_c \in [-\frac{\pi}{2}, \frac{\pi}{2}] \\ \arcsin(\sin(\varphi_s - \varphi_c)), & \text{otherwise} \end{cases} \tag{4.20}$$

and

$$\theta_{s,c} = \begin{cases} (\theta_s - \theta_c) \mod (2\pi), & \text{if } \varphi_s - \varphi_c \in [-\frac{\pi}{2}, \frac{\pi}{2}] \\ (\theta_s + \pi - \theta_c) \mod (2\pi), & \text{otherwise} \end{cases} \tag{4.21}$$

In order to speed up the dataset generation, we parallelized the ray casting procedure on GPU (with CUDA framework), by rendering each pixel (i,j) independently on a separate computing unit. In this way, we were able to render each image in an average time of 8 ms.

We also created an interactive version of our software to load panoramas, navigate through them and render the viewed portion from the camera, along with the corresponding ground truth light direction.

4.4 Experimental results

In this section, we present and analyze the performance of our *Fusion* light estimation approach.

First, we dive into some details about the training strategy of our model. Then, we discuss the performance obtained on four datasets: *SynthOut*, *RealOut*, VIDIT, and SID2. Finally, we perform an ablation study, showing how our architecture performs better, on all these four datasets with respect to the same model that doesn't include the surface normals prediction branch and the illumination model.

4.4.1 Network Training

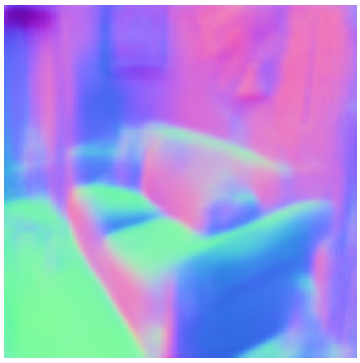
We randomly split our *RealOut* dataset (Section 4.3.2) by a proportion of 70%, 10%, and 20% for training, validation, and testing, respectively.

We trained our model from scratch on the training subset for 200 epochs, with Adam optimizer and an initial learning rate of 0.1, with a batch size of 24. The images were resized to match the input resolution of our architecture, that is (144 x 256) pixels.

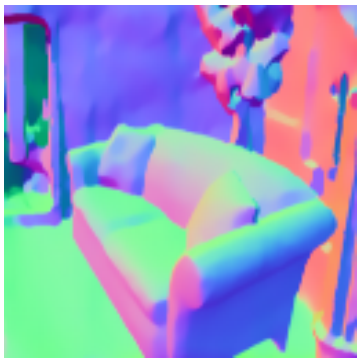
We tested different combinations of the weights α and β of the two terms $\mathcal{L}_1$ and $\mathcal{L}_2$ of the loss function, but it turned out that there were no substantial differences in performance. A possible explanation for the lack of impact of changing the weights α and β on the performance of the model could be that the model is not very sensitive to the precise values of these weights, as long as both $\mathcal{L}_1$ and $\mathcal{L}_2$ components are included in the loss function. Another possibility is that *RealOut* is still not a diverse or complex enough dataset to reveal the effects of different weight values for the loss function. Consequently, we opted to set $\alpha = 0.5$ and $\beta = 0.5$.

Note that, as no ground truth data about 3D normals is available in our datasets, we couldn't separately train our normals estimation network. Also, we performed surface normals estimation tests with the pre-trained network proposed in [10] as a fixed processing unit, but the predictions were not satisfactory from a qualitative point of view. In fact, our tests showed that the pre-trained network is able to obtain satisfactory predictions of normals in the case of indoor scenes (Fig. 4.11), but not in the outdoor case (Fig. 4.12), which is the scenario of our interest. This is the reason why we opted for a self-supervised approach that leverages the physical illumination

(a) Input image



(b) Predicted normals map



(c) Ground truth normals map

Figure 4.11: 3D normals prediction obtained with the code in [10], which is an implementation of the model in [25], in the indoor case.

model (described in Section 4.2.3). The weights of our normals prediction network are, in fact, indirectly learned by back-propagation of the $\mathcal{L}_2$ term of the loss (Eq. (4.4)), which measures the error between the luminance map Y of the input image and the one computed according to the simplified Phong's model, $\hat{Y}$ (Eq. (4.6)).

4.4.2 Notation and metrics

In order to evaluate the performance of our model, we measure the error between the predicted light direction $\hat{\mathbf{L}}$ and the real one $\mathbf{L}$ as the angular distance between the two directions, defined as:

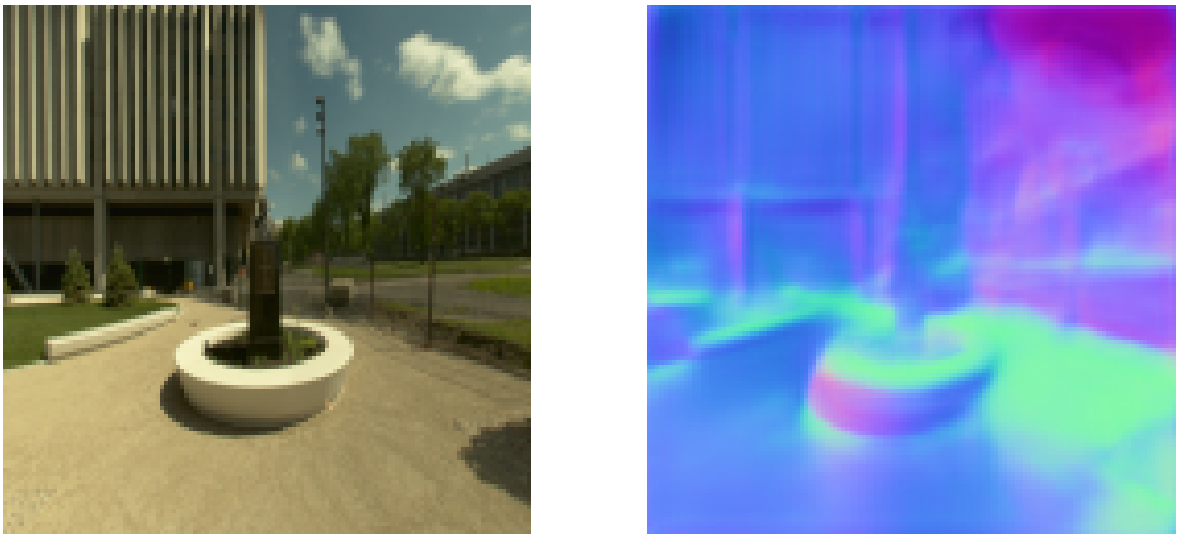
$$\alpha = \arccos\left(\frac{\mathbf{L}\cdot\hat{\mathbf{L}}}{||\mathbf{L}|| * ||\hat{\mathbf{L}}||}\right)\frac{180}{\pi}, \quad \in [0, 180] \tag{4.22}$$

Then, we compute the cumulative error distribution on all the images from the target test set T as:

$$F(x) = |S|, \quad S = \{i \in T | \alpha_i \leq x\}, \quad x \in \{0, 1, .., 180\}. \tag{4.23}$$

In our evaluation we also consider the AUC metric relative to the cumulative error curve:

$$\text{AUC} = \frac{1}{|T| \cdot 180}\sum_x F(x), \quad \in [0, 1], \tag{4.24}$$



(a) Input image (b) Predicted normals map

Figure 4.12: 3D normals prediction obtained with the code in [10], for outdoor scenes. Even if no ground truth normals map is available for validation, it can be seen that the result is not qualitatively satisfactory.

4.4.3 Results on *RealOut*

As can be seen in Fig. 4.13, our model achieves really good performance on the test subset of our *RealOut* dataset, with an AUC score of 0.976. The average angular error is 4.19 and the standard deviation is 4.17 (both values expressed in degrees).

We observe that, in [59], the authors tested their approach on a limited FOV dataset derived from Laval HDR Outdoor, in a similar way to what we did, obtaining the cumulative error distribution curve shown in Fig.5 (b) of their paper. Unfortunately, they have not released the aforementioned limited FOV dataset nor an in-depth description of the method used to generate it. For this reason, we don't know its characteristics, such as image resolution, vertical/horizontal FOV, anti-aliasing filtering, statistical distribution of camera orientations, etc. Nevertheless, assuming their dataset to be "similar" to ours, we could state that our method performs better, as our cumulative error curve leans more towards the ideal one. Note also that the code of their approach is not available and as such, we cannot test their method on our dataset for a direct comparison.

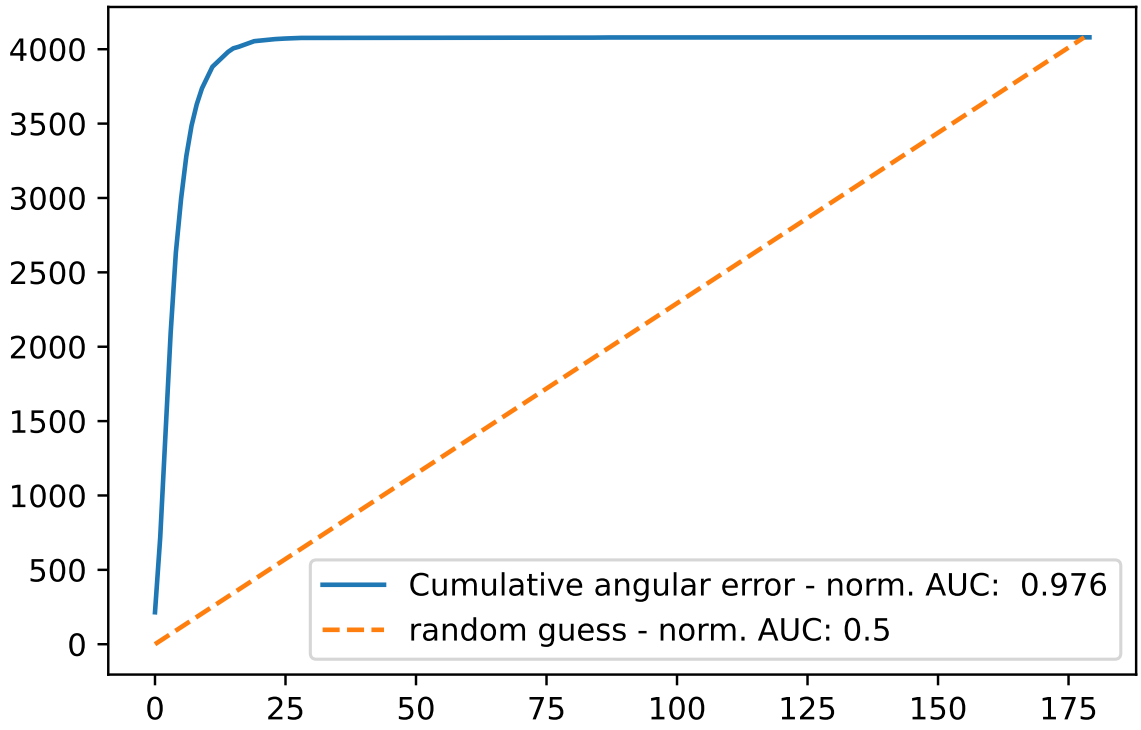


Figure 4.13: Cumulative angular error distribution on our *RealOut* dataset.

4.4.4 Results on *SynthOut*

We also evaluated our method on our CG-generated dataset, *SynthOut*, mainly to assess its capability to generalize on different kinds of images.

First, we tested our model directly on this dataset without any transfer learning. In this case, we only obtained an AUC score of 0.389, which is under the random guess of 0.5 (corresponding to a uniform error distribution). This is somewhat expected, as our model was trained from scratch on *RealOut* dataset, which is profoundly different from *SynthOut*, both for the nature of the involved images (real vs. CG rendered) and the semantic content.

Nevertheless, we show that, even with a few epochs of transfer learning and fine-tuning, our model manages to learn the features of this dataset very well. In particular, we adopted the following strategy:

1. initialize the network with the weights trained on *RealOut*;

2. freeze all layers except the Dense ones in the light network branch (shown in Fig. 4.4). In this way, only 8453 weights remain trainable;

3. train for 20 epochs (transfer learning step);

4. unfreeze all weights;

5. train for 10 (or 50) epochs, with Adam optimizer and small starting learning rate (10^{-3}) (Fine-tuning step).

With only 10 epochs of fine-tuning, our model achieves an AUC score of 0.890, while this score increases to 0.926 after 50 epochs.

4.4.5 Results on other CG datasets

We also evaluated our model on two other CG datasets: VIDIT and SID2 (see Section 4.1.3).

Note that, in the case of VIDIT, the ground truth direction of light is available only for the training subset, and it is in the form of one of eight possible cardinal directions for the horizontal angle, θ. On the other hand, the vertical angle, ϕ, is not specified, but it is constant in each image. Hence, in order to be able to do our tests, we proceeded as follows: first, we divided the training subset into two parts with a proportion of 70% and 30%. The first one was used for training and the second one for testing. Then, we set $\phi = \frac{\pi}{4}$ in our ground truth, for each image, as it seemed a reasonable value upon visual inspection.

In a similar way to what we did for *SynthOut*, we tested three levels of transfer learning and fine-tuning, *i.e.*: (i) no fine-tuning nor transfer learning, (ii) transfer learning (20 epochs) + 10 epochs of fine-tuning (FT-10) and (iii) transfer learning + 50 epochs of fine-tuning (FT-50). The obtained AUC scores for these two datasets and on *SynthOut* are summarized in Table 4.1,

From the table, it can be seen that, similarly to the case of *SynthOut*, without fine-tuning the performance obtained is not satisfactory, with just an AUC of 0.658 for

Table 4.1: AUC scores of our approach on four datasets against different fine-tuning levels: (i) no transfer learning/fine-tuning, (ii) transfer learning + 10 epochs of fine-tuning (FT-10), (iii) transfer learning + 50 epochs of fine-tuning (FT-50).

	SynthOut	**VIDIT**	**SID2**
None	0.389	0.658	0.639
FT-10	0.890	0.887	0.862
FT-50	0.926	0.944	0.917

Table 4.2: Angular error (in degrees) of our model on different datasets. On *SynthOut*, VIDIT, and SID2 the reported performance is obtained with the 50 epochs fine-tuned model (FT-50).

	SynthOut	**VIDIT**	**SID2**	***RealOut***
Mean	13.80	8.79	15.46	4.90
Std. Dev.	10.96	10.52	14.92	4.17

VIDIT and 0.639 for SID2. Nevertheless, we can observe that, even with a few epochs of transfer learning and fine-tuning our model is able to achieve good predictions on these datasets, which we recall, are CG generated and, as such, are very different from *RealOut*, which was used to train our model.

In Table 4.2 we report the mean angular error and standard deviation, obtained in the FT-50 case on the three CG datasets, compared to the ones obtained on *RealOut*.

Finally, we compare our approach with the one proposed in [131], which achieved a mean angular error of 14.22 degrees on SID2. The result obtained by our fine-tuned model is 15.46 (Table 4.2), which is slightly worse. However, by training our model from scratch on SID2, as the authors in [131] did, our approach is able to outperform theirs, achieving an average angular error of 12.92 degrees.

4.4.6 Ablation Study

In this section, we discuss the analysis we did to verify that our *Fusion* architecture performs better than a simplified one that does not exploit the information

given by the prediction of surface normals. We will show that including a dedicated branch for indirectly learning the task of surface normals estimation allows for better light prediction results.

We compare the prediction performance of the following two models:

1. our complete architecture, *Fusion* (Fig. 4.1).

2. A deep neural network with the same architecture described in Section 4.2.2 (and shown in Fig. 4.4), but modified in order to take as input only an RGB image (without concatenation with the normals prediction map). We refer to this model as *Light Only*.

We tested both models on the four datasets: *SynthOut*, VIDIT, SID2 and *RealOut*. In order to have a fair comparison, we adopted, for both *Light Only* and *Fusion*, the same hyper-parameters and training scheme of the FT-50 case.

The cumulative prediction error curves obtained are shown in Fig. 4.14: while in the case of *SynthOut* *Fusion* only slightly outperforms *Light Only*, with an AUC score of 0.926 against 0.893, on VIDIT the performance difference increases, with AUC scores of 0.954 and 0.871 for *Fusion* and *Light Only*, respectively. This is true also for SID2, for which the AUC obtained are 0.878 for *Light Only* and 0.917 for *Fusion*. In the case of *RealOut* dataset, the performance gap between the two architectures increases, with an AUC of 0.976 of *Fusion* against 0.886 of *Light Only*. The same conclusions can also be drawn by analyzing the values of the mean angular errors obtained (shown in Table 4.3): on all four datasets the *Fusion* architecture achieves a significantly lower mean error.

These results can be interpreted as follows: in the case of *SynthOut*, although the objects in the images are realistic, the layout and in general, the complexity of the scene is still limited in comparison with real outdoor photos. As a result, even *Light Only* model, which does not use information about surface normals, is able to learn the features needed to determine the direction of light. This is no longer true in the case of more complex and realistic outdoor scenes, such as those contained in VIDIT, SID2,

Table 4.3: Comparison of mean angular error (in degrees) of *Fusion* and *Light Only* models on four reference datasets. For *SynthOut*, VIDIT and SID2 the reported results are obtained with both *Fusion* and *Light Only* models fine-tuned with the FT-50 strategy.

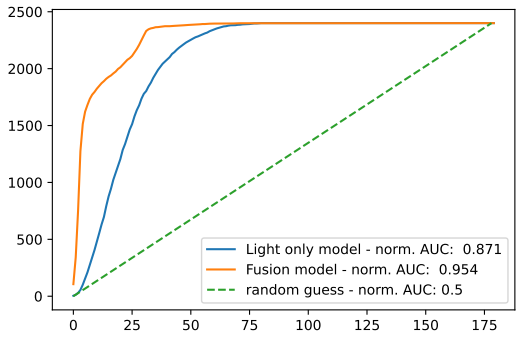
	SynthOut	**VIDIT**	**SID2**	***RealOut***
Fusion	**13.80**	**8.79**	**15.46**	**4.90**
Light Only	19.77	23.75	22.42	21.07

and even more the ones from the *RealOut* dataset. In these cases, *Fusion*, through the additional information on surface normals, is able to achieve considerably better prediction accuracy, confirming our intuition.

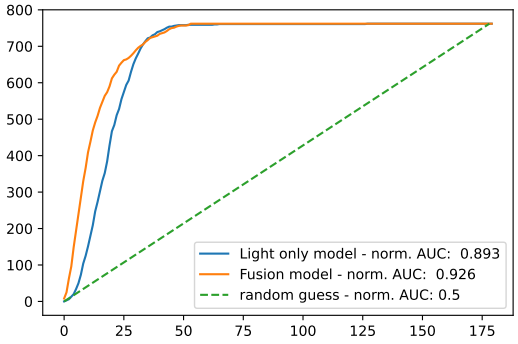
4.5 Discussion

In this chapter, we presented a novel deep-learning approach for the prediction of 3D light direction, from outdoor images, in a single light source scenario. The key aspect of our method is the joint use of a data-driven strategy with a physics-inspired model to exploit additional geometric information, *i.e.*, 3D surface normals, in order to achieve better prediction accuracy. The idea we leveraged is that geometric information about the depicted scene can aid the task of light source direction prediction, as it allows the model to disambiguate cases where different light directions may give rise to the same reflections/shading on objects.

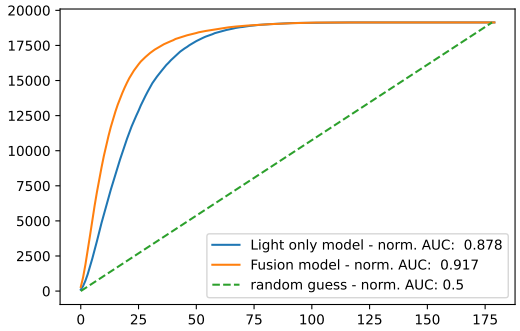
Since we couldn't find any pre-trained network for outdoor 3D surface normal estimation, neither we had a dedicated ground truth with surface normal maps, we adopted the following self-supervised training strategy (see Fig. 4.1): we input the current surface normals and light direction predictions to a simplified version of the Phong's model (Eq. (4.6)). Then, we compare the model's output with the luminance channel of the input image, through a reconstruction term (Eq. (4.4)) embedded in the loss function. In this way, our network indirectly learns to predict the surface normals (or at least an approximation of them) without the need for the associated ground



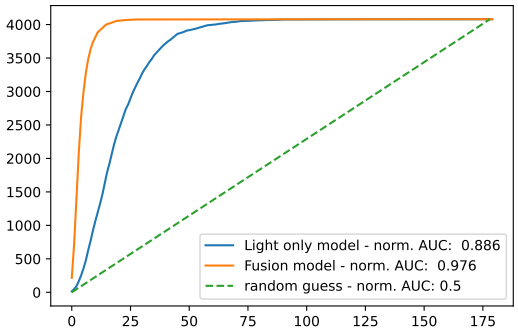
(a) Comparison on VIDIT



(b) Comparison on *SynthOut*



(c) Comparison on SID2



(d) Comparison on *RealOut*

Figure 4.14: Comparison between *Fusion* and *Light Only* architectures on 4 datasets. On *SynthOut* the two models achieve similar performance, while on all the other datasets *Fusion* achieve significantly better AUC scores, showing how the inclusion of the surface normal estimation branch helps to improve the light estimation performance. Note: for *SynthOut*, VIDIT and SID2 we compare both models fine-tuned with the FT-50 strategy.

truth data. Also, both the task of surface normals and light direction estimation are learned simultaneously and in a synergistic manner, as one helps improve the other.

We observed how the datasets available in the literature are either too small, not realistic, or simply don't contain (reliable) ground truth light information. Also, SUN360, which is one of the most widely used datasets for the evaluation of light source estimation methods, is not available anymore, due to IP issues. For this reasons, in order to train and evaluate our deep light estimation model, we developed two algorithms (available as open-source code), that allow for the creation of both synthetic and real image datasets. With these algorithms, we built two datasets, namely: *SynthOut* and *RealOut*. *SynthOut* is composed of 20k images, containing realistic objects, such as cups, plates, sofas, chairs, musical instruments, etc.. which are placed in an outdoor scene with a tarmac textured floor and a sunny sky. A high number of different light directions and colors is present in the dataset. *RealOut* contains 40k real, limited FOV outdoor images extracted from the 360 degree panoramas contained in the Laval Outdoor HDR database. This dataset involves a great number of outdoor scenes, with buildings, trees, cars, and streets, lit with different sky conditions, ranging from sunny to cloudy. We believe that our data generation algorithms constitute a valuable contribution that can be used to tackle the problem of data scarcity in this field. In fact, they can be used to automatize the process of building light estimation datasets starting from either CG scenes or 360 degree panoramas.

RealOut was used as a base dataset to train from scratch and validate her proposed architecture. We tested the generalization capability of our method, by making predictions also on *SynthOut*, VIDIT, and SID2 datasets, by adopting and comparing different transfer learning and fine-tuning strategies. We showed that even with a few epochs of fine-tuning our model achieves satisfactory performance on all these datasets. We also showed that our architecture, if trained from scratch, is able to outperform the method proposed in [131], in terms of mean angular error on the SID2 dataset.

Finally, in order to evaluate the contribution of surface normals estimation in the light direction prediction task, we performed an ablation study, in which we compared

the performance of our complete architecture (*Fusion*) and a simplified one (*Light only*) that does not incorporate the normal estimation branch and the illumination model. When evaluated on *SynthOut* VIDIT, SID2, and especially on *RealOut*, the *Fusion* model achieved significantly better AUC scores and angular errors than the *Light Only* version. These results confirmed that, for real, complex, outdoor images, the surface normals information helps to achieve better light direction predictions.

Chapter 5

LIGHT-BASED FORGERY DETECTION

In this chapter, we describe two forgery detection methods based on the identification of inconsistencies in the illumination field of the image under examination. Indeed, under appropriate assumptions, one of the clues that may indicate the presence of photomontage (*e.g.*, splicing attacks) is an inconsistent illumination between various parts of the image under examination.

Consider, for example, the image in Fig. 5.1: at a close look, it can be seen that the direction of the light of the two people involved in the scene is not consistent. This is evident, for example, by focusing on the shadows of the face and clothes. As mentioned earlier, this kind of analysis is thinkable only under certain assumptions. If the input image has "simple" illumination, for example, due only to sunlight in the outdoor environment then we are in the condition of directional light. This means that the light source can be considered infinitely far from the scene, *i.e.*, can be completely described by a vector in 3D space. In such a situation it is reasonable to assume this

Figure 5.1: Splicing example. The light direction is not consistent between the two people in the image.

Figure 5.2: An example of an indoor scene with multiple light sources creating a complex lighting field.

constant direction within the whole image. Consequently, if the objects/patches used to perform the photomontage (*e.g.*, splicing) are derived from images with sufficiently different light directions, this method can identify the forgery.

Let us consider an indoor scene with multiple lighting sources inside a room (Fig. 5.2). In this case, the illumination varies greatly within the image itself and it is clear that in such a situation this type of analysis is not applicable. It is important to emphasize that this method is not to be considered the ultimate solution with regard to forgery detection. However, in certain situations, it could be used in conjunction with other, more general methods.

Consider, in fact, that an attacker will typically try to make it more difficult to identify the photomontage created, through counter-forensic measures, for example by applying Gaussian-smoothing to smooth the edges of the spliced/copy-moved area, or by adding noise to make a synthetic fake image more realistic. It should also be considered that ad-hoc methods can be developed with the aim of limiting/eliminating artifacts generated by a photomontage, which could be detected by forgery detection methods published in the past. In this sense, the use of the light field as a discriminant in a forgery detection method makes it at least complicated to apply counter-forensic measures. In fact, even with professional software, it is difficult to modify the lighting

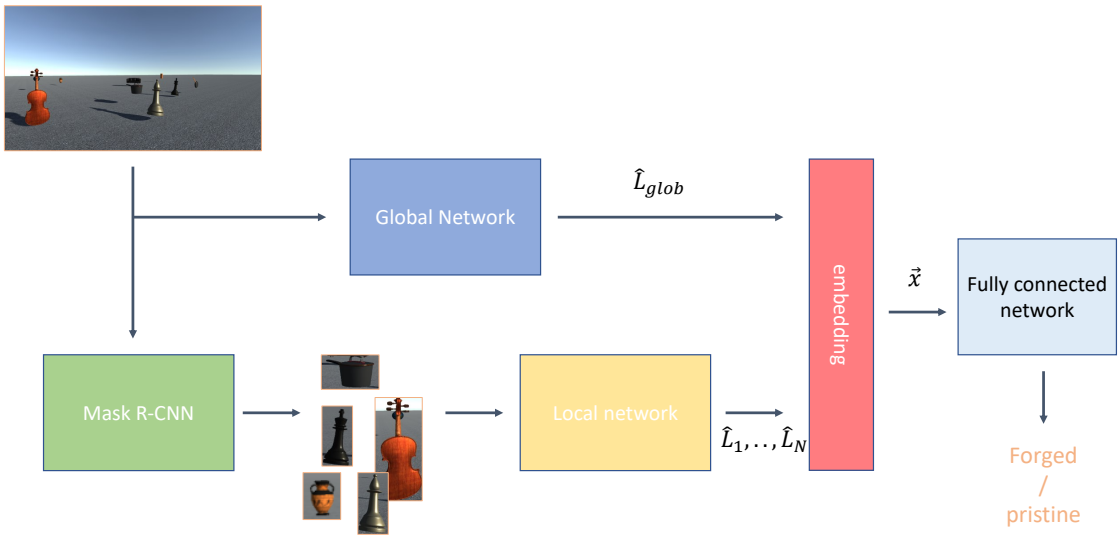


Figure 5.3: Architecture of proposed light-based forgery detection method.

of an object to make it consistent with the other objects/patches present in the image.

The chapter is organized as follows: Section 5.1 presents the architecture of our first light-based splicing detection method, while Section 5.2 presents the experimental results. Section 5.3.2 describes a second light-based splicing detection approach and Section 5.4 presents the experimental results of the second approach. Finally, in Section 5.5 the obtained results are discussed and some conclusions are drawn.

5.1 Method architecture

In this section, we describe our light inconsistency-based forgery detection method, which we denote as *light1*, for clarity. The whole architecture is described schematically in Fig. 5.3.

First, a network is used to perform instance segmentation on the input image. Instance segmentation consists of performing a segmentation that assigns different labels to different instances of an object of the same class. This is in contrast to semantic segmentation, where a single label is typically assigned to all objects of the same class. The architecture we have chosen is the "MaskR-CNN", proposed in [56].

This model is an extension of the popular Faster R-CNN network [121] for object detection, which adds an additional branch to the network to predict a binary mask for each object in the image.

The Mask R-CNN model is composed of a base network, a Region Proposal Network (RPN), and a head network. The backbone network is typically a pre-trained Convolutional Neural Network (CNN) such as ResNet or VGG, which is responsible for extracting features from the input image. The RPN is a fully convolutional network that takes the features from the backbone as input and generates a set of proposal boxes, which are regions of the image that may contain objects.

The head network takes the features from the backbone network and the proposal boxes from the RPN as input and performs three tasks: object classification, bounding box regression, and mask prediction. The object classification task involves predicting the class of the object (*e.g.*, person, car, chair) for each proposal box. The bounding box regression task involves refining the proposal boxes to enclose the objects in a more accurate way. The mask prediction task involves predicting a binary mask for each object, which outlines the object's shape in the image. We chose this model as it was shown to achieve state-of-the-art results on several benchmarks, such as COCO dataset [90].

The architecture proposed in Chapter 4 is then used to predict the 3D direction of the light, both on the whole image and on the segmented patches (the "global network" and "local network" blocks in Fig. 5.3). In this way, both local and global light information is available and can be combined to obtain a feature vector x (embedding step), which is in turn fed as input to a binary classifier to label the image as forged or pristine. The idea is that x should carry information about eventual inconsistencies in the light field of the image.

It should be noted that the number of patches that are extracted through the segmentation is not known a priori. If we created x simply as a concatenation of the light predictions its dimension would vary image by image. As a consequence, the classifier should allow for non-fixed size inputs, such as a Recurrent neural network

(RNN) or, more specifically, a Long short-term memory model (LSTM). These architectures, in fact, include the concept of memory/state and are capable of correlating information at different "times" in the input sequence. This is useful when the input signals are audio, video, temporal series, and text. In our case, however, it is not appropriate to identify a sequential relationship between our input data, *i.e.*, the local light predictions $\hat{L}_i$. Rather, we could say that there is a spatial relationship between them, assuming that the prediction $\hat{L}_i$ is valid in all pixels belonging to patch i.

To obtain a fixed-size feature vector x we opted for the following embedding strategy. First, we convert each local light prediction into polar coordinates, $\theta_i = 180 + \frac{180}{\pi} atan2(\hat{L}_z, \hat{L}_x)$ and $\varphi_i = 90 + \frac{180}{\pi} arcsin(\frac{\hat{L}_y}{||\hat{L}||})$ and analogously for the global prediction, obtaining θ_{glob} and φ_{glob}. Then, we compute the frequency histograms of the local prediction angles θ and φ as follows. Let

$$S_i = \{\theta_j \mid \theta_j \in [i\alpha, (i+1)\alpha[, \quad j = 1, .., N\}, \quad i = 0, 1, .., k \tag{5.1}$$

be the set of θ angles that fall in the i-th bin of the frequency histogram, where N is the number of extracted patches, and $\alpha = 10$. As θ ranges in the interval $[0, 360[$, k is fixed to 36. Analogously, we define:

$$T_i = \{\varphi_j \mid \varphi_j \in [i\beta, (i+1)\beta[, \quad j = 1, .., N\}, \quad i = 0, 1, .., k \tag{5.2}$$

as the set of φ angles that fall in the i-th bin of the histogram. In order to obtain the same number of bins as for θ, we set $\beta = \frac{\alpha}{2} = 5$. The histograms for θ and φ are computed, respectively, as:

$$h_i^{\theta} = |S_i|, \quad i = 0, 1, .., k \tag{5.3}$$

and

$$h_i^{\varphi} = |T_i|, \quad i = 0, 1, .., k \tag{5.4}$$

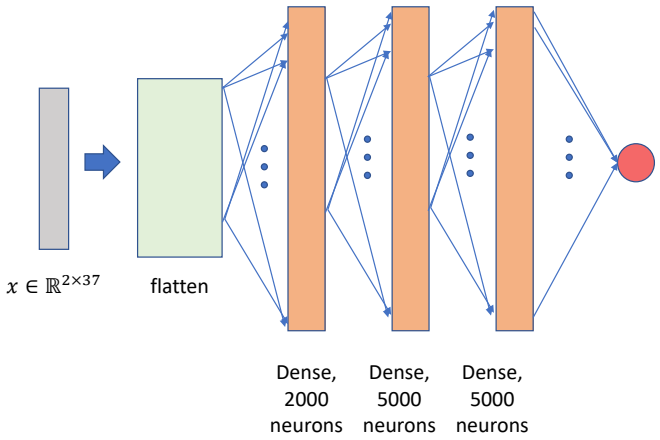


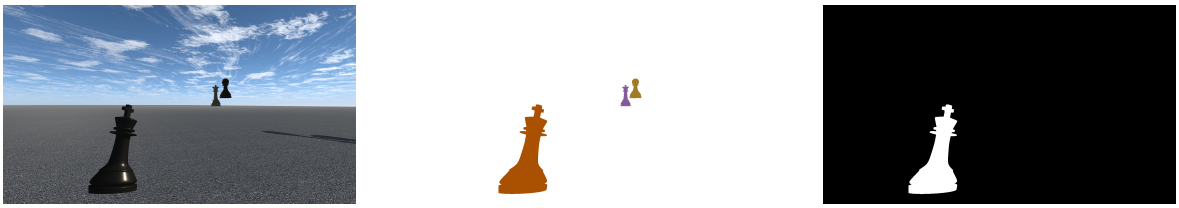
Figure 5.4: Binary classifier used in our proposed approach, that takes as input the feature vector x, which is an embedding of the local and global light predictions.

Finally, we construct the feature vector, $x \in \mathbb{R}^{2\times(k+1)}$ as:

$$\begin{aligned} x_{1,i} &= h_i^{\theta}, \quad i = 1, .., k, \\ x_{1,k+1} &= \theta_{glob} \\ x_{2,i} &= h_i^{\varphi}, \quad i = 1, .., k, \\ x_{2,k+1} &= \varphi_{glob} \end{aligned} \tag{5.5}$$

Note that we chose $\alpha = 10$ as it is compatible with the deviation standard (~ 5 degrees) of the prediction angular error of our light-estimation network.

A fully connected neural network was used as a classifier. This network was trained on a dataset composed of couples of feature vectors x extracted with the already-trained local and global light prediction networks and corresponding binary labels (forged/pristine). The architecture is shown in Fig. 5.4



(a) Spliced image (b) Objects mask (c) Splicing ground truth mask

Figure 5.5: An example spliced image and the corresponding masks from our synthetic splicing dataset.

5.2 Experiments

This section describes the experimental setup we used to validate our method and the results obtained on two datasets. First, we will describe these datasets, then we will give an overview of the training strategy and the hyper-parameters used to train our models. Finally, we will present the results obtained on both datasets.

5.2.1 Synthetic splicing dataset

As a first experiment, we tested the proposed architecture on a synthetic splicing dataset, derived from our *SynthOut* dataset (described in detail in Chapter 4). *SynthOut* is a dataset of images rendered with Unity starting from an outdoor 3D scene containing realistic objects, such as chairs, balls, cups, sofas, etc. The rendered images are lit with a great variety of lighting conditions, obtained by simulating the movement of the sun during the day and by changing the color of the light. Each spliced image was obtained by cutting an object from an image and inserting it into another one. Being the object segmentation masks available as additional meta-data in *SynthOut*, we have been able to automate the segmentation process with the certainty of correctly extracting the objects. We only selected objects that were entirely contained in the image. Also, we made sure that the copied objects were not occluding other objects in the target image. An example of a spliced image, along with the corresponding ground truth mask is shown in Fig. 5.5.

The resulting dataset contains a total of 7566 images, of which 3783 are spliced,

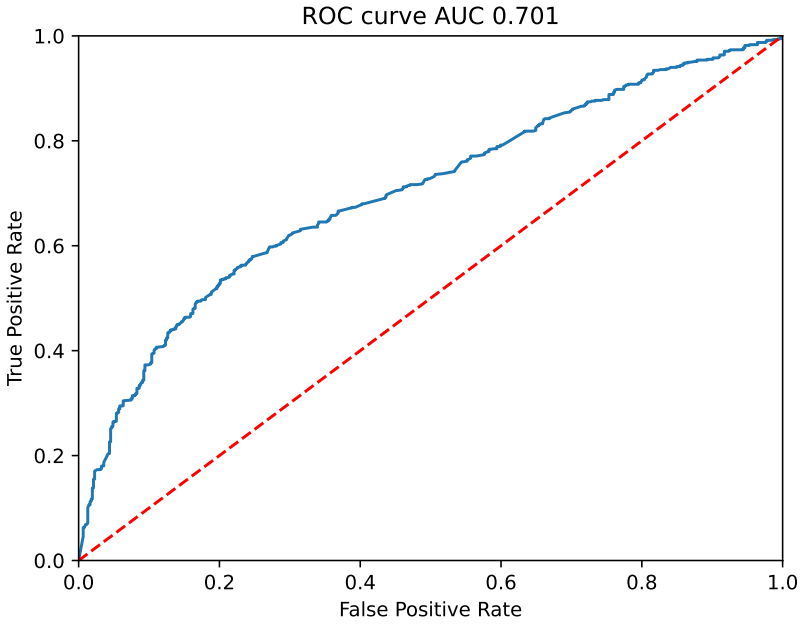


Figure 5.6: ROC curve and corresponding AUC obtained on synthetic splicing dataset.

with a resolution of (1080 × 1920) pixels. From now on, we will refer to this dataset as "synthetic splicing dataset".

As mentioned, we used the same deep-learning architecture for local and global light predictions. However, we trained one instance of it on *SynthOut* for the global light direction prediction task and a second instance on a dataset of patches extracted from *SynthOut* for the local light direction prediction.

We then computed the x feature vectors (as in Eq. (5.5)) starting from the local and global predictions obtained with the previously discussed networks and stored them along with the ground truth binary labels. This newly constructed dataset was then used to train and test the classifier network. In order to assess the splicing detection performance of our approach, we measured the F1-score and the ROC curve on a portion of the dataset (20%), used as a test set. We obtained an accuracy of 0.655, and an F1-score of 0.627, while the ROC curve, shown in Fig. 5.6 has a corresponding area under the curve (AUC) of 0.701. In Fig. 5.7 the confusion matrix is shown. As can be seen from these results, there is still room for improvement.

As the splicing dataset was derived from *SynthOut*, we have available, as ground

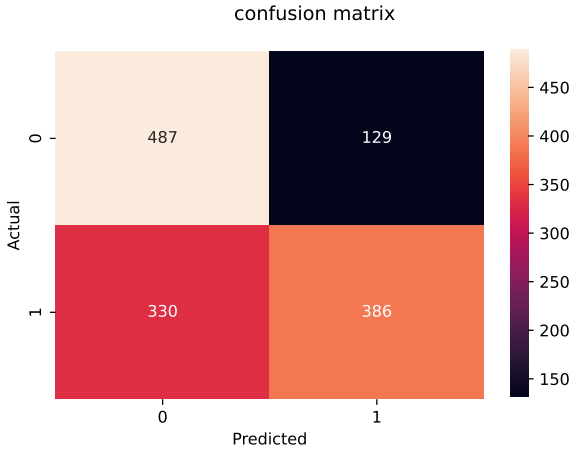


Figure 5.7: Confusion matrix on sythetic splicing dataset.

truth information, not only the binary label on each image but also the 3D light directions of the patches extracted and pasted to create the spliced images. Thanks to this, we were able to carry out an in-depth analysis to understand what caused the classification errors. We recall that, in this experiment, the objects are always correctly segmented, as we are leveraging the object masks directly generated with Unity during the creation of *SynthOut*. Therefore, the classification errors can be due only to two factors: (i) the light predictions are not accurate, or (ii) the chosen light prediction embedding, *i.e.*, the feature vectors x, don't carry enough information to be useful for the binary classifier. In order to exclude the second hypothesis, we performed the following test: we constructed an "ideal" dataset of feature vectors x and corresponding binary labels by using the ground truth of local and global light directions, instead of the model's predictions. We then split this dataset into train and test sets and trained and tested the classifier on these two subsets, respectively. In this case, we obtained an accuracy of 0.960 and an F1-score of 0.964, while the AUC is of 0.965, (the ROC curve is shown in Fig. 5.8). These results allow us to rule out hypothesis (i) as the main cause of classification errors. Indeed, this result demonstrates that, if the light predictions were ideal, the chosen feature vectors x can effectively be used to train the binary classifier.

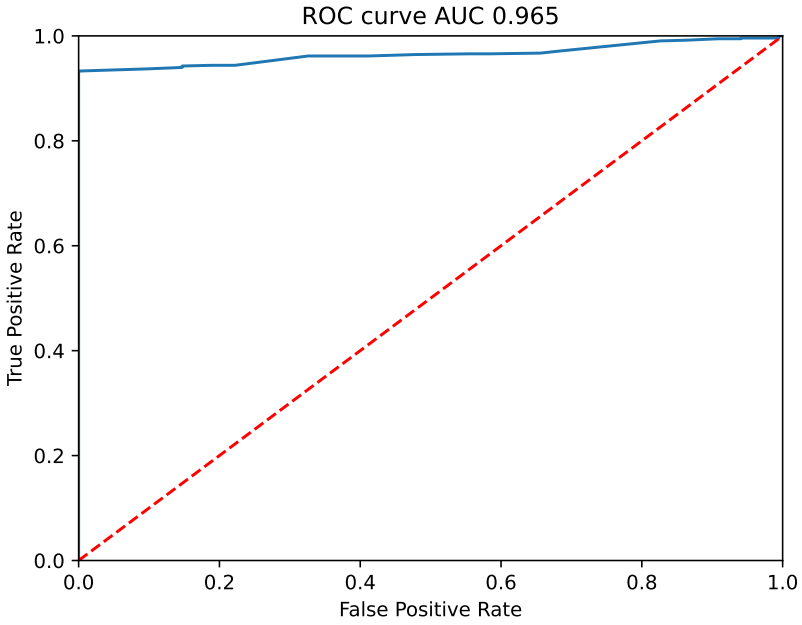


Figure 5.8: ROC curve and AUC obtained on synthetic splicing dataset by using the ground truth 3D light directions instead of the predicted one.

Therefore, we focused our attention on the performance of light direction prediction *i.e.*, hypothesis (i), this being the upstream cause of classification errors. In Fig. 5.9 the cumulative frequency curve, $F(E)$ of the angular errors between the local ground truth light directions and the predicted ones are shown (we used a bin size of 1 degree). In the figure, the percentiles e_α at different α levels (80%, 90%, 95%, 99%) are highlighted. They have been determined as $e_\alpha : \mathbb{P}(E \leq e_\alpha) = F(E = e_\alpha) = \alpha\%$. For example, the 80-th percentile is $e_{80} = 12.0$ degrees, while the 99-th percentile is $e_{99} = 51$ degrees. This means that, by setting 51 degrees as the maximum angular error acceptable, we have a probability of 0.99 that the prediction error will be lower than this threshold, while this probability drops to 0.80 if we admit a smaller error of 12 degrees. From the inverse perspective, if we fix a confidence level, *e.g.*, 80%, we have to expect angular errors up to 12 degrees in $\sim 80\%$ of the cases while, if we want a confidence level of 99%, we need to expect errors up to 51 degrees in 99% of the cases. For simplicity, we refer to these percentiles as "angular confidence thresholds".

We now analyze how the angular confidence thresholds relate to the distribution

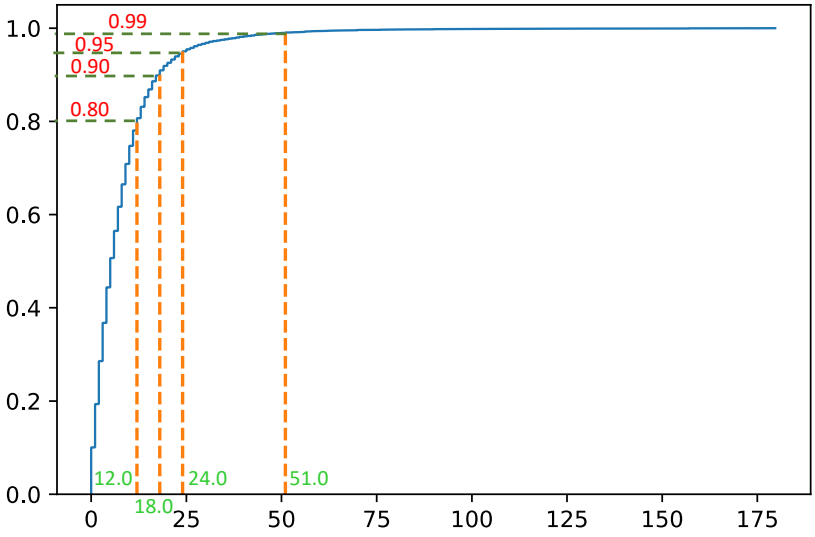


Figure 5.9: Cumulative angular error distribution of light predictions on patches. On the plot, we highlighted four different percentiles, corresponding to the levels 80%, 90%, 95%, and 99%.

of light direction differences in the spliced images in our dataset. In particular, we ask ourselves the following question: given an angular error threshold, *e.g.*, 12 degrees, what is the percentage of spliced images in the dataset having an angular difference between the global light direction vector and the ones in the spliced regions that is below 12? Intuitively, if this percentage is high, we cannot expect good detection results, because for all these images the light prediction obtained from our network is not reliable.

In Fig. 5.10 the cumulative frequency curve of angles between light directions in the spliced images of the dataset is shown (we used a quantization step of 1 degree). The plot also shows the percentage of spliced images for which the light direction angle difference between the spliced objects and the target image is lower than the angular confidence thresholds. For example, if we consider the angular threshold of 12 degrees (corresponding to a confidence level of 80%), we see that 13.81% of spliced images will have light differences lower than this value. This means that for this percentage

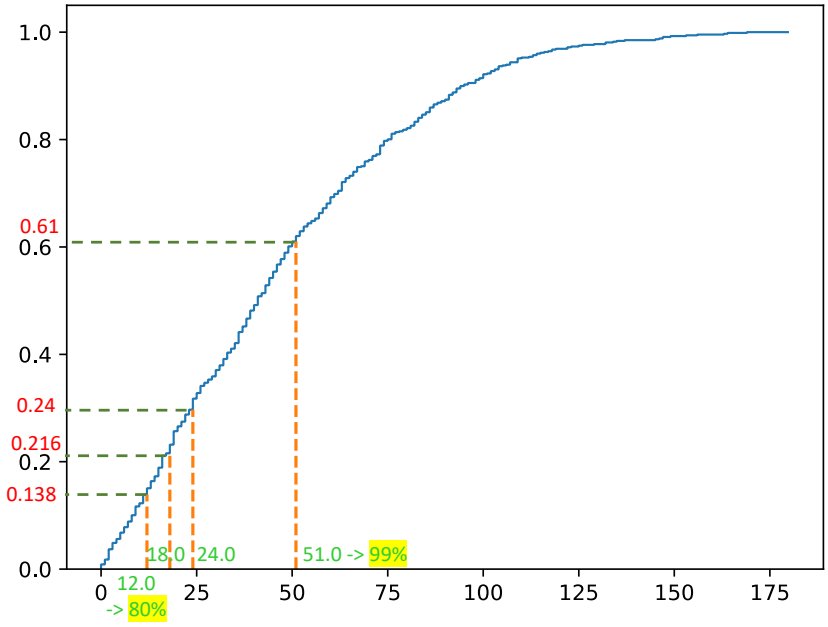


Figure 5.10: Cumulative frequency curve of angular differences between light directions in spliced images. Different percentiles e_α of light direction prediction errors (the α level is highlighted in yellow) are also marked on the plot.

of images the prediction given by the network is not reliable to create meaningful feature vectors. If we wanted a confidence level of 99%, we should consider an angular threshold of 51 degrees; in this case, 61% of spliced images in the dataset will have light differences that are lower than this value.

We, therefore, performed a new experiment in which we recreated the splicing dataset by adding the constraint that each spliced image contained objects from images with a light direction difference greater than 50 degrees. We refer to this dataset as "98%-confidence splicing dataset", because it contains only spliced images with angular differences between angular vectors that are greater than the threshold needed to have a 98% confidence of light prediction. In this case, the performance was better, with an accuracy of 0.774, an F1-score of 0.785, and an AUC of 0.828. The ROC curve is shown in Fig. 5.11, while two examples of predictions are shown in Fig. 5.12. Here, the bounding boxes of the objects on which the local predictions have been computed are drawn in red, while the predicted light direction and the ground truth ones are marked

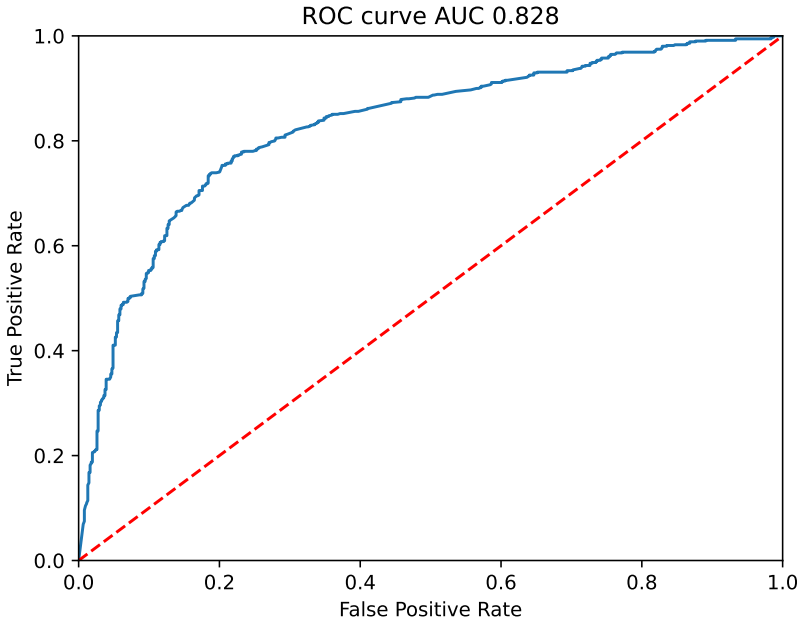


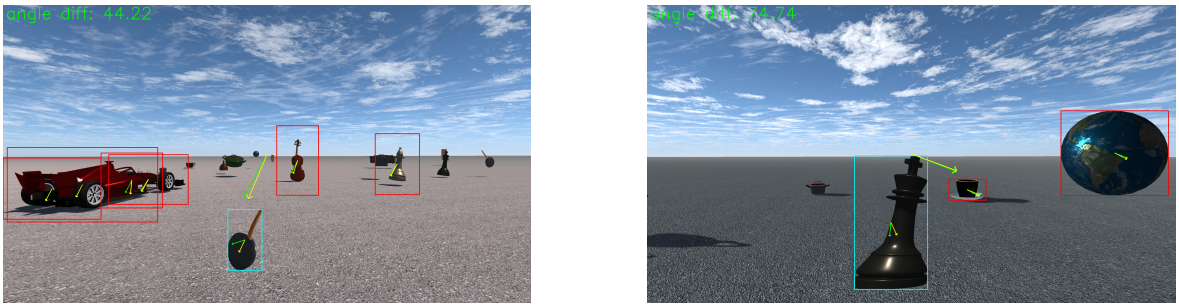
Figure 5.11: ROC curve and AUC obtained on the 98%-confidence splicing dataset.

as green and yellow arrows, respectively. The object for which the predicted light was furthest from the global light prediction is highlighted in cyan. In this way, we are able to obtain an approximate localization of the spliced area.

5.2.2 Evaluation on benchmark datasets

In order to assess the applicability of the proposed method in a realistic scenario, involving real images with complex forgeries, we evaluated it on the benchmark CASIA v2.0 dataset [38]. This dataset consists of 12624 images with resolutions ranging from 320×240 to 800×600, in various formats (jpg, BMP, tif). Of these, 7491 images are original and 5123 are forged. Because the tampering attacks involved are both copy-move and splicing, we filtered out the copy-moved images, as our method is tailored to splicing detection. Note that the ground truth is given as a binary label at the image level (pristine vs. forged). Hence, in contrast to the synthetic dataset case, we don't have any object segmentation mask.

Therefore, the Mask R-CNN model (see Fig. 5.3) was used to segment the objects in the images and extract the corresponding patches on which the local light



(a) (b)

Figure 5.12: Two examples of splicing detection on the 98%-confidence splicing dataset. The bounding boxes of the objects on which the local light predictions were computed are shown in red. The predicted and ground truth light directions are marked as green and yellow arrows, respectively. The object with the greatest difference in light directions from the global one is marked as the spliced one (cyan). The green text indicates the angular difference between the light directions of the two images involved in the splicing.

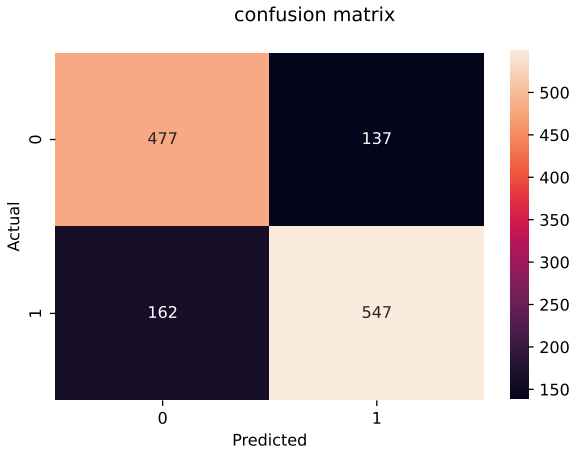


Figure 5.13: Confusion matrix obtained on 98%-confidence splicing dataset.

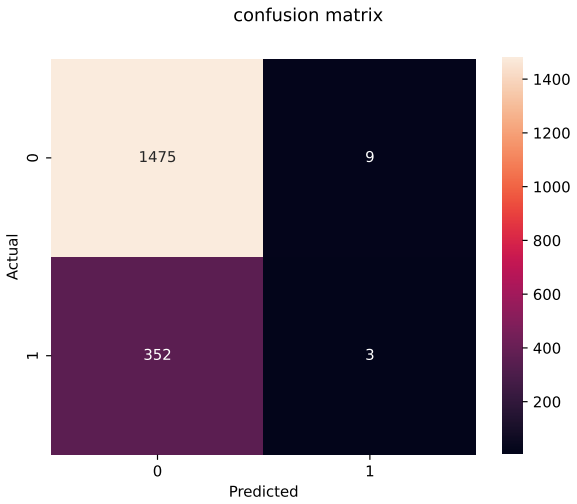


Figure 5.14: Confusion matrix obtained with proposed method on CASIA2 dataset.

predictions will be done.

For this experiment, we used the light prediction network's weights pre-trained on *RealOut* (see Section 4.3.2), as this dataset contains real images and as such, it is more similar to CASIA2 than *SynthOut*. After extracting the patches with Mask R-CNN and computing the local light predictions, the feature vectors were computed by means of Eq. (5.5), in order to build the dataset to train the binary classifier.

In this test, the performance are not satisfactory. In fact, we obtain a good accuracy of 0.804 but the precision is very low: 0.25. The discrepancy between accuracy and precision is due to the imbalance in the number of spliced and original images within the CASIA2 dataset. From the confusion matrix, shown in Fig. 5.14, it is easy to see that the model is heavily biased towards predicting almost always a negative label (pristine). The ROC curve, shown in Fig. 5.15, confirms the bad performance, as it leans towards the random guess.

A possible reason for the poor performance is the fact that, because no information about the involved light condition in the images is available in CASIA2, we couldn't train nor fine-tune our light prediction network on this dataset. As such, we could only use the pre-trained light prediction network as if it were a fixed computing unit. We also don't know what is the distribution of the angular difference between

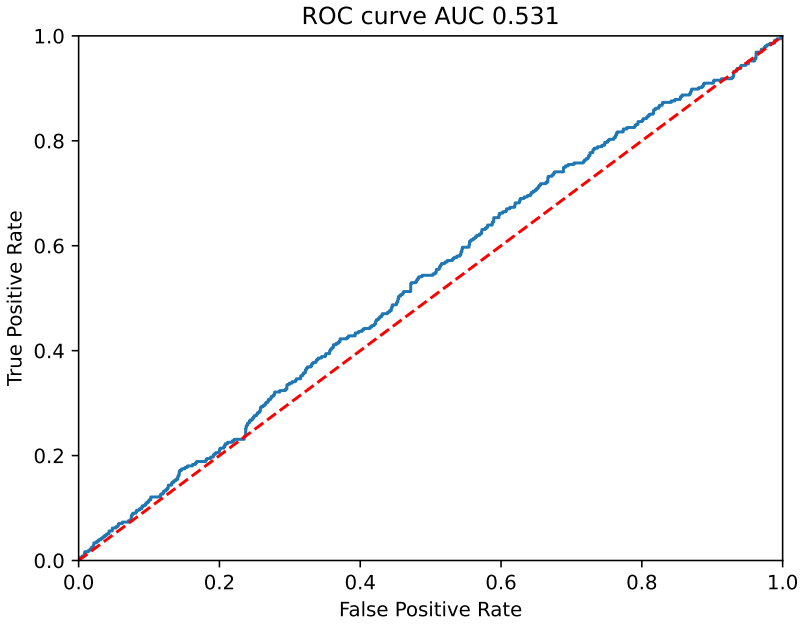
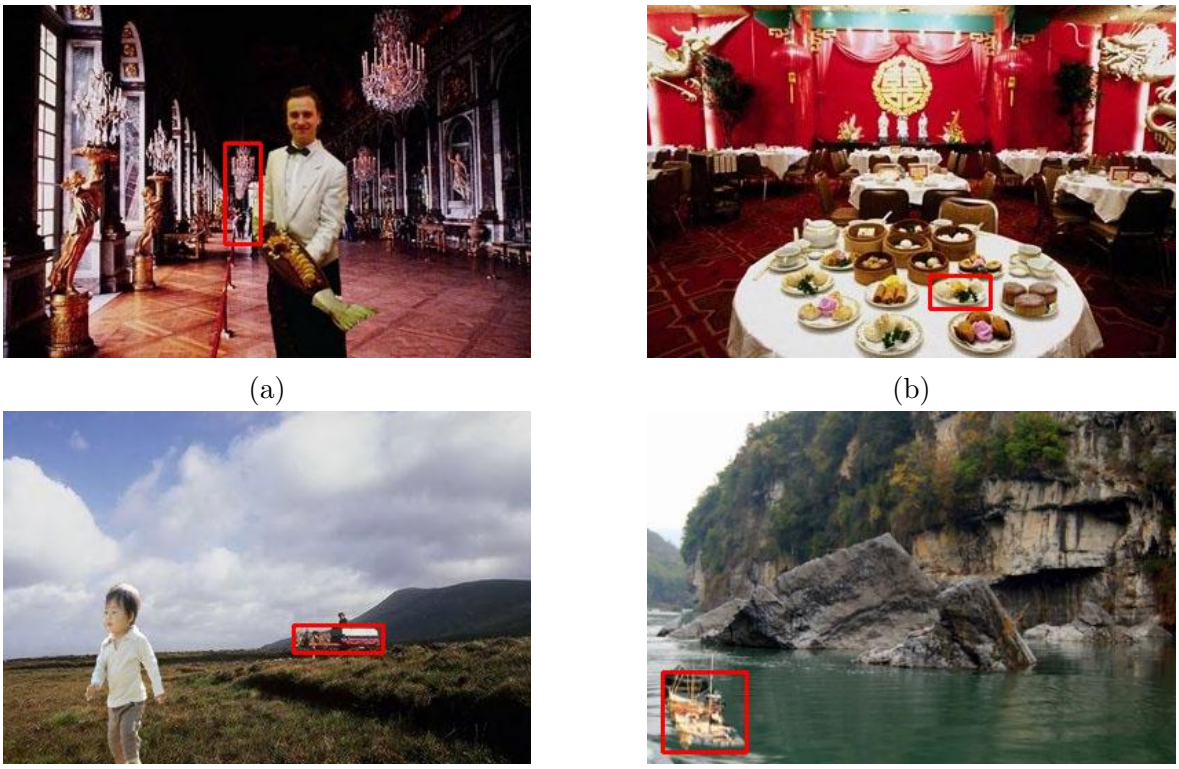


Figure 5.15: ROC curve obtained on CASIA2 dataset.

light directions of the spliced patches, making it impossible to perform an analysis on the level of uncertainty of light predictions. Note also that, the hypothesis of images in an outdoor environment, with a single light source, is not guaranteed. As mentioned at the beginning of this chapter this is crucial for the application of our method, as it expects different objects in the same pristine image to be lit from the same light direction.

We must also consider the fact that there is no certainty that the patches segmented through the Mask R-CNN network actually contain the spliced regions. Mask R-CNN has been trained to segment objects belonging to a priori known classes. However, many of the spliced images in CASIA2 involve irregularly shaped patches that do not necessarily correspond to objects. This can be seen in Fig. 5.16, where the bounding boxes of the segmented objects are shown in red. While we do not have the ground truth map of the spliced regions available, it is possible even by eye to see how often the segmented zones are not related to the spliced regions.



(a) (b)

(c) (d)

Figure 5.16: Examples of segmentation on CASIA2 spliced images. Even if a ground truth map of the spliced region is not available, it can be seen that often the segmented objects don't relate to the spliced regions.

5.3 InverseRenderNet - based approach

In this section, we describe another splicing detection approach we designed, which is based on light consistency estimation. For clarity, we refer to it as *light2*. We leveraged the architecture proposed in [151], referred to as InverseRenderNet by the authors, as a means to predict the light conditions, as well as other 3D information, in a given image. InverseRenderNet is an Encoder-Decoder type deep learning architecture that, given an input image, is able to obtain, as output, the 3D normals map, the albedo diffusion map, and the 3D scene illumination described as a set of coefficients in a spherical harmonic series expansion. This network effectively performs an "inverse" rendering, as it decomposes a 2D image in a set of 3D orthogonal components. The simplifying assumptions made by the authors are the following:

1. the illuminating light sources are distant. This means that the incoming radiance (incident illumination) can be described as a function of the incident angle to each object's surface.

2. The scene is described by a local Lambertian diffuse model. This model neglects effects such as cast shadows and inter-reflections.

3. Only a limited number of spherical harmonic basis functions is considered (9). As a consequence, high frequencies components of lighting conditions are filtered out. In many situations though, such as the outdoor setting, this is still sufficient to approximate the illumination function very well.

One of the key aspects of InverseRenderNet is that it is self-supervised. This means that there is no need to train the network on new data. In fact, the decoder in the network consists of a differentiable forward rendering that, from the extracted 3D components (normals, albedo, and light coefficients), reconstructs an approximation of the input image. The optimal light coefficients are obtained by minimizing the error between the reconstructed and input images.

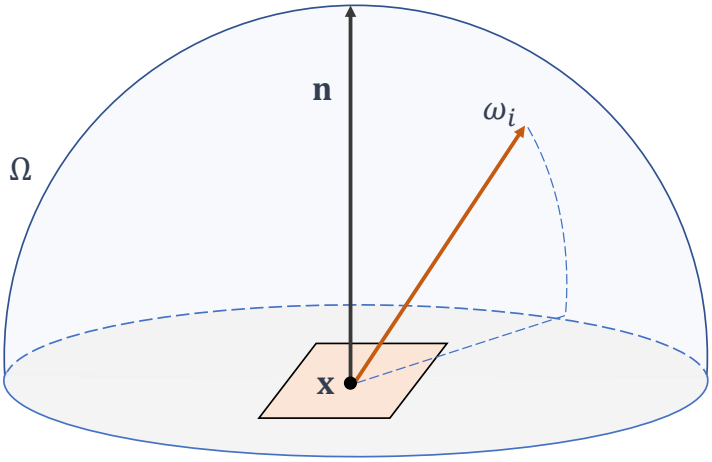


Figure 5.17: Geometry setup for radiance model.

5.3.1 Differentiable forward rendering

In order to understand our approach, we first need to introduce a few physical quantities and mathematical tools (such as spherical harmonic expansion). Then, we can then describe the forward rendering block used by the authors as a decoder in their architecture. This is important as we will also use this block in our approach as a separate unit.

Let us denote as $Y = Y(\mathbf{x})$ the reflected radiance at a given point $\mathbf{x}$ in the scene. Under the assumption of diffuse Lambertian materials, this quantity is a measure of the intensity of reflected light (equally in all directions) at a certain wavelength. The reflected radiance depends also on the incident illumination, usually referred to as incoming radiance, and the materials' reflection properties, which are typically described by a bi-directional reflectance distribution function (BRDF).

Let's consider the geometric setup shown in Fig. 5.17. We have denoted as $\mathbf{x}$ a generic point of an object's surface. $\mathbf{n}$ is a unit vector representing the surface normal at the point $\mathbf{x}$, while ω_i is a unit vector describing a generic direction of incoming light. Ω is the upper hemisphere, centered at $\mathbf{n}$, describing the set of all possible incoming directions.

We can now express the local relationship between reflected radiance and incoming radiance with the following equation:

$$Y(\mathbf{x}) = \int_{\Omega} \rho(\mathbf{x}, \omega_i)\, L(\mathbf{x}, \omega_i)\, (\omega_i \cdot \mathbf{n})\, \mathrm{d}\omega_i \tag{5.6}$$

where:

- $L(\mathbf{x}, \omega_i)$ is the incoming light (irradiance) at point $\mathbf{x}$ from (negative) direction ω_i.
- $\rho(\mathbf{x}, \omega_i)$ is the BRDF function at point $\mathbf{x}$. By using the assumption of Lambertian surfaces, we can drop the dependency on the incoming direction: $\rho(\mathbf{x}, \omega_i) \rightarrow \rho(\mathbf{x})$.

Note that all the quantities are expressed in the *local* coordinate system with the orientation expressed by $\mathbf{n}$ (as in Fig. 5.17).

With the assumption of distant lighting, we can also drop the dependency of L from the position. Furthermore, L will only depend on the global incident direction ω_i^g. Hence, we can use the substitution: $L(\mathbf{x}, \omega_i) \rightarrow L(\omega_i^g)$. With these simplifications, Eq. (5.6) can be rewritten as follows:

$$Y(\mathbf{x}) = \rho(\mathbf{x}) \int_{\Omega} L(\omega_i^g)\, (\omega_i \cdot \mathbf{n})\, \mathrm{d}\omega_i \tag{5.7}$$

In order to compute the integral of Eq. (5.7), we need to express all the quantities in either global or local coordinates. In order to express the incident direction vectors from one reference system to the other we need to perform a 3D rotation, that can be easily expressed as $\omega_i^g = R_{\alpha,\beta}\, \omega_i$ and $\omega_i = R_{\alpha,\beta}^{-1}\, \omega_i^g$, where α and β represent the 3D rotation of the local surface normal n in the global reference system. Hence, Eq. (5.7) becomes:

$$Y(\mathbf{x}) = \rho(\mathbf{x}) \int_{\Omega} L(R_{\alpha,\beta}\, \omega_i)\, (\omega_i \cdot \mathbf{n})\, \mathrm{d}\omega_i \tag{5.8}$$

As both $\mathbf{n}$ and ω_i are unit vectors that lie on the unit sphere, all the functions involved in Eq. (5.8) can be expanded in the spherical harmonic series. Let us consider a generic function $f(\theta, \phi)$ defined on the unit sphere; we can expand it as:

$$f(\theta,\phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} f_{l,m}\, \xi_{l,m}(\theta,\phi) \tag{5.9}$$

where:

- $\xi_{l,m}(.)$ is the spherical harmonic of order l and index m. For a given order l there are $2l+1$ harmonics. These functions form an orthonormal basis for functions on the unit sphere:

$$\langle \xi_{l,m}, \xi_{l',m'} \rangle = \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\pi} \xi_{l,m}(\theta,\phi) \xi_{l',m'}(\theta,\phi) \sin\theta \, d\theta d\phi = \delta_{l,l'} \delta_{m,m'} \tag{5.10}$$

- $f_{l,m}$ are the coefficients of the harmonic series expansion, that can be computed as the inner product:

$$f_{l,m} = \langle f, \xi_{l,m} \rangle = \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\pi} f(\theta,\phi) \xi_{l,m}(\theta,\phi) \sin\theta \, d\theta d\phi \tag{5.11}$$

For simplicity of notation, we will now linearize the indices of the harmonic functions. As discussed in [119], by using the first 9 spherical harmonics (up to order $l = 2$), it is possible to obtain a really good approximation of the majority of lighting fields. Note that, in Eq. (5.8), we are only considering the upper hemisphere as the domain of integration, while the spherical harmonic functions are defined on the complete sphere. This problem can be solved by substituting the $(w_i \cdot \mathbf{n})$ term with a transfer function defined as follows:

$$A(\omega_i, \mathbf{n}) = \begin{cases} (\omega_i \cdot \mathbf{n}) & \text{if} \quad \omega_i \in \Omega \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

And using the complete sphere as a domain for integration.

After expanding both $L(.)$ and $A(.)$ as harmonic series and truncating them at the first 9 terms of the expansion, it can be shown (the reader is referred to [119] for an in-depth derivation) that the following approximation holds:

$$Y(\mathbf{x}) \approx \rho(\mathbf{x}) \sum_{i=1}^{9} L_i \mathrm{b}_i(\mathbf{n}) \tag{5.13}$$

where L_i are the harmonic series expansion coefficients of the incoming light L and $\mathrm{b}_i(\mathbf{n})$ are the basis functions expressed in Cartesian coordinates. Note that the equations seen so far can be applied independently on each wavelength. If we consider the wavelengths corresponding to the R, G, and B channels, we can rewrite Eq. (5.13) as:

$$\hat{\mathbf{Y}}(\mathbf{x}) = \mathrm{diag}(\boldsymbol{\rho}(\mathbf{x}))\mathbf{L}\mathrm{b}(\mathbf{n}) \tag{5.14}$$

where:

- $\hat{\mathbf{Y}}(\mathbf{x}) \in \mathbb{R}^3$ is the approximation of the reflectance $Y(\mathbf{x})$ for the 3 R,G,B channels.
- $\boldsymbol{\rho}(\mathbf{x}) \in \mathbb{R}^3$ is a vector containing the BRDF, or albedo, evaluated at point $\mathbf{x}$ for the R, G and B channels.
- $\mathbf{L} \in \mathbb{R}^{3\times 9}$ is a matrix containing the first 9 harmonic series expansion coefficients for the R, G, and B channels.

Let us now consider a 2D photography of a 3D scene, with $S = H \times W$ pixels. As each point $\mathbf{x}$ in the 3D space is projected to a pixel coordinate, we can directly apply Eq. (5.14) to describe the intensity values of each R,G,B intensity at each pixel coordinate. Let us denote by $\hat{\mathbf{I}} \in \mathbb{R}^{3\times S}$ the matrix of linearized stacked RGB pixels. Analogously, we define $\mathbf{A} \in \mathbb{R}^{3\times S}$ as the matrix containing the albedo R,G,B values at each pixel, and $\mathbf{N} \in \mathbb{R}^{3\times S}$ as a matrix containing the surface normals, evaluated at each pixel coordinate, expressed as vectors with 3 Cartesian components. Finally, we define $\mathbf{B}(\mathbf{N}) \in \mathbb{R}^{9\times S}$ as the matrix obtained by evaluating $\mathrm{b}(\mathbf{n})$ on the surface normal at each pixel.

We can now write Eq. (5.14) for the whole image as:

$$\hat{\mathbf{I}}^{lin} = \mathbf{A} \odot (\mathbf{L}\mathbf{B}(\mathbf{N})) \tag{5.15}$$

where $\odot$ is the Hadamard product (element-wise).

In order to simulate the post-processing of a real camera we also apply a gamma correction on the obtained intensities:

$$\hat{\mathbf{I}} = [(\hat{\mathbf{I}}^{lin}_{i,j,c})^{1/\gamma}], \quad i = 1, .., H, j = 1, .., W, c = 1, .., 3 \tag{5.16}$$

We now compose Eq. (5.15) and Eq. (5.16) in a compact way as:

$$\hat{\mathbf{I}} = f(\mathbf{A}, \mathbf{N}, \mathbf{L}) \tag{5.17}$$

The function $f(.)$ implements the differentiable *forward* renderer used by the authors as a decoder of their InverseRenderNet architecture. Note that, during inference, while both the surface normals and the albedo maps are directly predicted by the encoder, the L coefficients are obtained by minimizing the error between the input image I and the reconstructed version through $f(.)$.

5.3.2 Proposed approach

In Fig. 5.18 the architecture of the proposed method is shown. In the upper branch of the diagram, the InverseRenderNet is used to predict the albedo, normals, and light coefficients of the input image, denoted as $\mathbf{A}_{glob}, \mathbf{N}_{glob}, \mathbf{L}_{glob}$, respectively. In the lower branch, Mask R-CNN (see Section 5.1) is used to perform an instance segmentation on the input image and extract the ROIs $\mathbf{P}_i$, $i = 1, .., k$, containing the recognized objects. The k ROIs are given as input to the InverseRenderNet and the corresponding albedos $\mathbf{A}_i$, normals $\mathbf{N}_i$, and lighting coefficients $\mathbf{L}_i$, are obtained, for $i = 1, .., k$.

An embedding block is then used to combine both local and global predictions and build the features $\mathbf{X}$ and $\mathbf{Z}_{rel}$. Finally, these features are given as input to a CNN-based classifier that is trained to predict a global, binary label on the image: pristine VS forged.

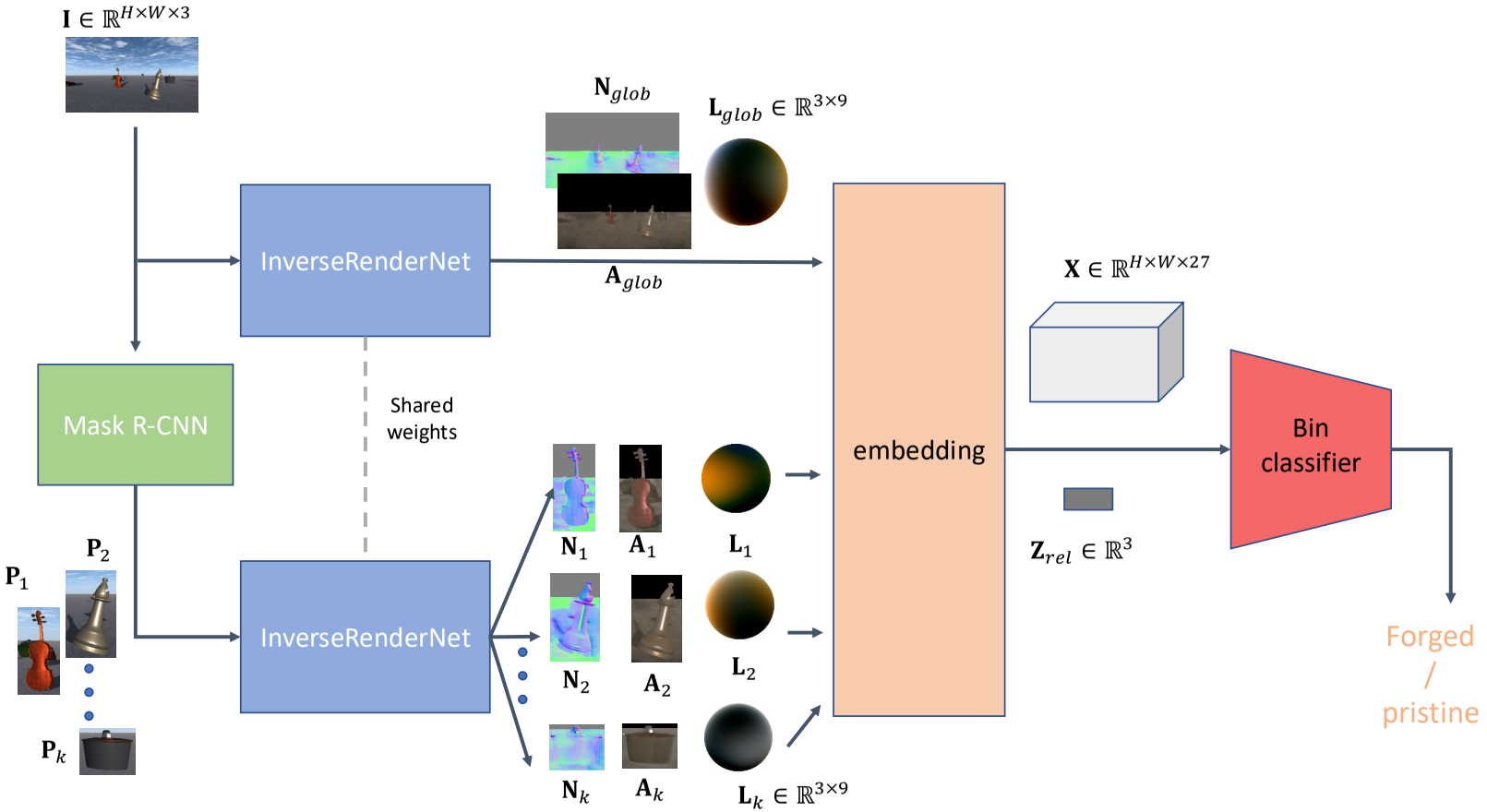


Figure 5.18: Architecture of proposed splicing detection approach based on InverseRenderNet [151].

5.3.2.1 Feature extraction

This section describes the feature extraction procedure (implemented in the embedding block in Fig. 5.18).

Let

$$\mathbf{X}^c = [\mathbf{X}^c_{i,j,h}] = \begin{cases} [\mathbf{L}_m(c,h)],\ h = 1,..,9,\ c = 1,2,3, & \text{if } \exists\, m \in \{1,..,k\} \,|\, (i,j) \in \mathbf{P}_m, \\ [\mathbf{L}_{glob}(c,h)],\ h = 1,..9,\ c = 1,2,3 & \text{otherwise} \end{cases} \tag{5.18}$$

be the map of local and global light coefficients for the channel c. This map assigns each pixel (i,j) to the set of light coefficients predicted for a particular segmented patch if (i,j) belongs to that patch, otherwise, the pixel is assigned to the set of global light coefficients. The feature map $\mathbf{X} \in \mathbb{R}^{H \times W \times 27}$ is obtained by stacking on the third dimension the maps for the three channels $\mathbf{X}^c$.

We also designed a "cross-relighting" feature, that evaluates the effect of relighting a patch u with the light coefficients of another patch v, through the forward rendering function $f(.)$ (Eq. (5.17)). The idea is that, under the assumption of distant lighting, if we relight a patch with a sufficiently different light field (such as the one predicted from a spliced patch), the obtained relighted patch should be quite different from the original one. Formally, we define the cross-relighting error matrix as follows:

$$\mathbf{E}_{rel} = [e_{u,v}] = [||\mathbf{P}_u - \hat{\mathbf{P}}^v_u||_2] = [||\mathbf{P}_u - f(\mathbf{A}_u, \mathbf{N}_u, \mathbf{L}_v)||_2], \quad u,v = 1,..,k \tag{5.19}$$

Finally, we define the feature vector $\mathbf{Z}_{rel} \in \mathbb{R}^3$ as:

$$\mathbf{Z}_{rel} = [\min(\mathbf{E}_{rel}, \max(\mathbf{E}_{rel}), \text{avg}(\mathbf{E}_{rel})] \tag{5.20}$$

In this way, we obtain a fixed-size feature vector, regardless of the number of extracted patches. Note that, in the case in which no patches are extracted by Mask R-CNN, we simply set $\mathbf{Z}_{rel} = [0,0,0]$.

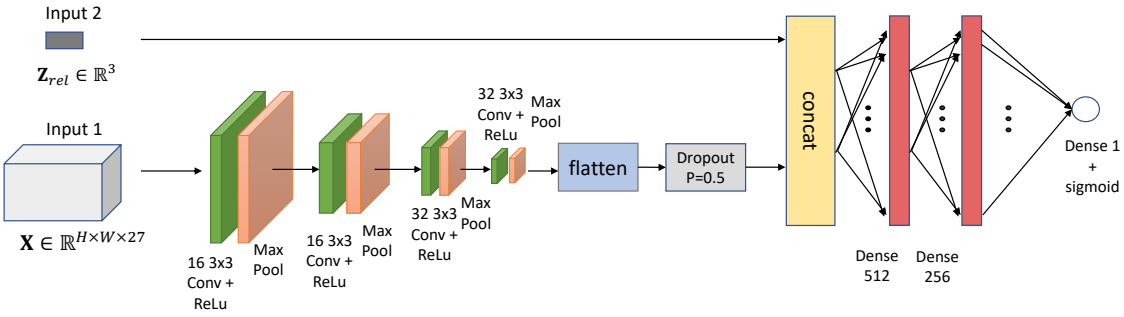


Figure 5.19: Architecture of binary classifier employed in the proposed method.

5.3.2.2 Binary classifier

In this section, we briefly describe the used binary classifier (red block in Fig. 5.18).

First, we input the light coefficients map $\mathbf{X}$ to a set of four convolutional layers + ReLu activation functions, each followed by max pooling. The number of filters is $16, 16, 32, 32$, for the four layers, respectively. Then, we flatten the obtained feature and pass it through a dropout layer, with a probability of 0.5 of removing each neuron. Then, we concatenate this vector with the other input feature $\mathbf{Z}_{rel}$. The obtained vector is then given as input to a fully connected layer with 512 and neurons, followed by another fully connected layer with 256 neurons. Finally, a neuron with a sigmoid activation function is used to obtain the single output probability.

5.4 Experiments

In this section, we discuss the experimental setup and the results obtained with our InverseRenderNet-based approach.

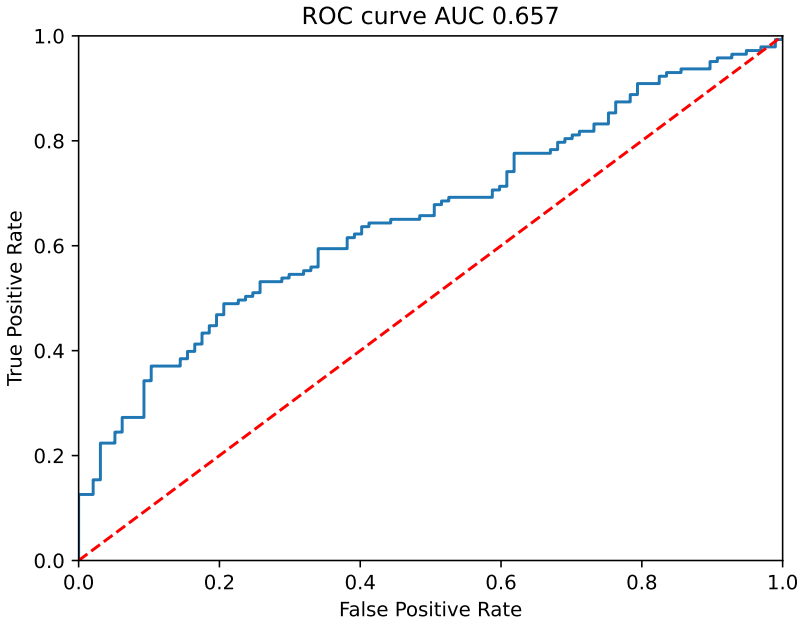


Figure 5.20: ROC curve obtained with proposed InverseRenderNet-based approach on synthetic splicing dataset.

Similarly to the approach discussed in Section 5.2, we first tested the method on the synthetic splicing dataset derived from *SynthOut*. Subsequently, we repeated the experiment on the CASIA2 dataset. Note that, in both experiments, the InverseRenderNet was used without fine-tuning, as this architecture is self-supervised. Also, in the experiment on the synthetic splicing dataset, the objects were segmented by using the available masks (as was done in Section 5.2). In this way, we can fairly compare the two proposed methods. Finally, the binary classifier (discussed in Section 5.3.2.2) is trained from scratch on the two datasets for the two experiments, respectively.

5.4.1 Results evaluation on Synthetic dataset

This section presents the results obtained with our splicing detection framework evaluated on the synthetic splicing dataset (see Section 5.2.1).

We obtained an accuracy of 0.608, a precision of 0.681, and an F1-score of 0.662. In Fig. 5.20 the ROC curve, with an AUC of 0.657 is shown, while the confusion matrix is reported in Fig. 5.21.

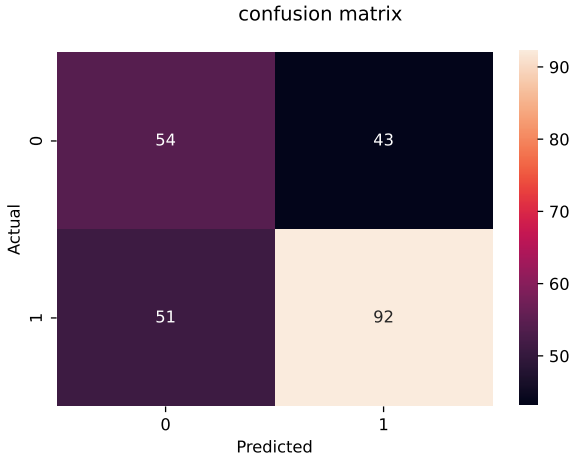


Figure 5.21: Confusion matrix obtained with proposed InverseRenderNet-based approach on synthetic splicing dataset.

Analogously to what we did for our first approach, we performed a test on the 98%-confidence splicing dataset, in which the splicing attacks are done only using images with an angular difference of light greater than 50 degrees. Predictably, the performance improved, with an accuracy of 0.776, a precision of 0.819, and an F1-score of 0.848. The ROC curve (AUC = 0.821) and the confusion matrix are shown in Fig. 5.22 and Fig. 5.23, respectively.

5.4.2 Results evaluation on Benchmark dataset

The accuracy, precision, and F1-score obtained on CASIA2 are 0.688, 0.349 and 0.326, respectively. The ROC curve, shown in Fig. 5.25 has a corresponding AUC of 0.646. The confusion matrix reported in Fig. 5.24, shows that our approach still commits a considerable amount of false negative errors.

5.5 Discussion

In this section, we briefly compare and comment on the results obtained with our two light-based splicing detection methods. For brevity of notation, we refer to the first method, described in Section 5.1 as *light1*, and to the second one, presented in Section 5.3.2 as *light2*.

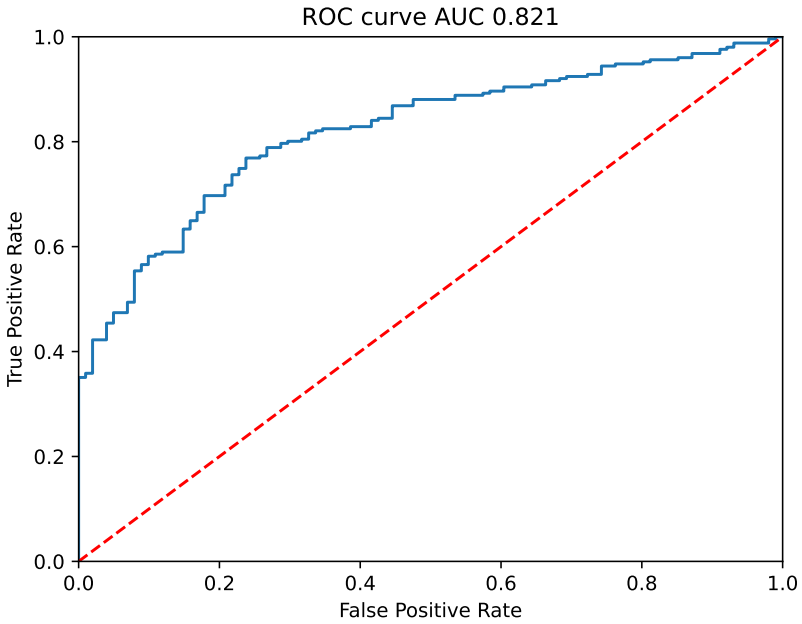


Figure 5.22: ROC curve obtained with proposed InverseRenderNet-based approach on 98%-confidence splicing dataset.

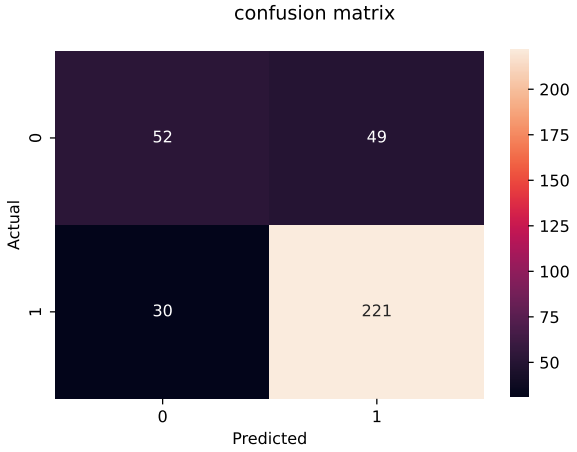


Figure 5.23: Confusion matrix obtained with proposed InverseRenderNet-based approach on 98%-confidence splicing dataset.

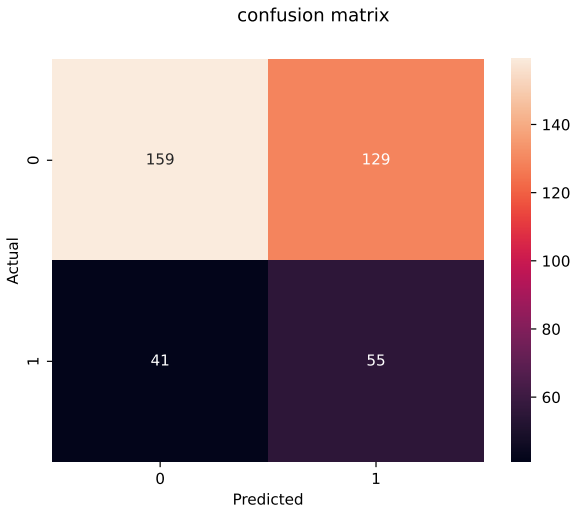


Figure 5.24: Confusion matrix obtained with proposed InverseRenderNet-based approach on CASIA2 dataset.

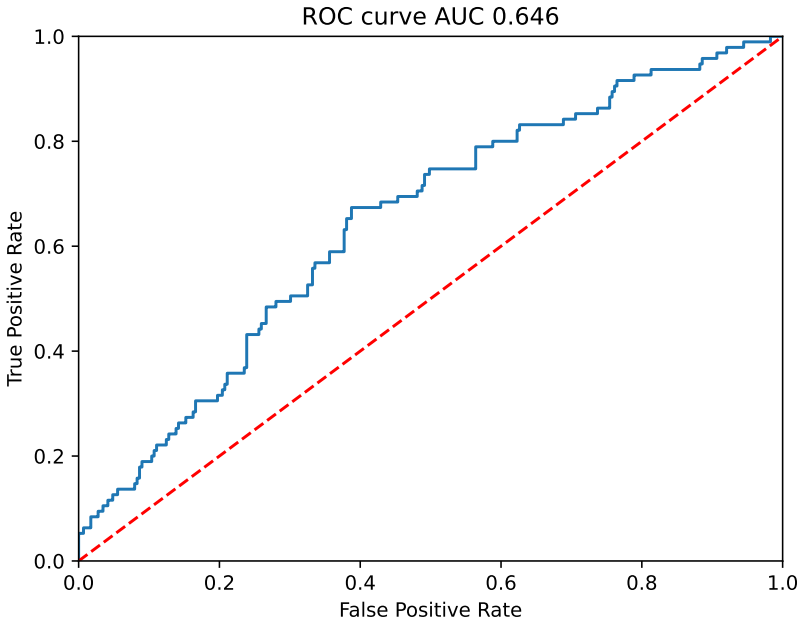


Figure 5.25: ROC curve obtained with proposed InverseRenderNet-based approach on CASIA2 dataset.

	Synth-spliced	**Synth-spliced filt.**	**CASIA2**
light1	0.655	0.785	0.381
light2	**0.662**	**0.848**	**0.463**

Table 5.1: Comparison of F1-score between the two proposed light-based splicing detection methods on the three mentioned datasets.

As can be noted from the results presented in Section 5.2 and Section 5.4, the performances of both methods are not satisfactory. In particular, in the case of realistic images, for which no ground truth information about light is available (such as in the case of CASIA2), our approaches cannot be directly applied in a reliable way, as there is no possibility to fine-tune the light estimation network, which is the core of our whole detection pipeline. However, we showed that when this is possible and the hypothesis of sufficiently different light fields in the spliced images is met, we can successfully train the light network on the new data and obtain good detection results (as for the filtered synthetic splicing dataset).

Table 5.1 shows that *light2* outperforms *light1*, especially for the filtered synthetic splicing dataset, for which the difference in terms of F1-score is almost 6%. The same happens in the CASIA2 case, for which the improvement of the F1-score is more than 8%. This is probably due to the intrinsic property of self-supervision of the InverseRenderNet architecture, which enhances its generalization ability.

Even if the performance are not really good, we think that these methods can still be applied in conjunction with other approaches, possibly in a forensic scenario where the explainability of the adopted techniques is an important requirement. Both the local and global light predictions obtained with our methods can, in fact, be used as clues for the localization of forged areas or as features to combine with other local descriptors. Moreover, Farid has recently shown [47] that even state-of-the-art text-guided generative diffusion models, such as DALL-E2, struggle with complex lighting and reflections, due to their reliance on generating images that appear visually realistic

to humans rather than on explicit physical or geometric modeling. This underscores the importance of maintaining the use of physical models and constraints in digital forensic research as a complementary tool for detecting image forgeries.

Chapter 6

ENSEMBLE FORGERY DETECTION APPROACH

In this chapter, we describe an ensemble architecture that exploits the methods presented in the previous chapters, with the aim of (i) achieving better forgery detection performance, (ii) performing multi-class classification, distinguishing between copy-move and splicing forgery types, (iii) providing localization of the forged areas and (iv) retrieving the source regions used to perform the attack in the case of copy-move forgeries. The idea we leveraged is to train a neural network - "FusionForgery" - to combine the outputs of a set of available forgery detection approaches (we refer collectively to these as "base" methods). It is worth noting that these approaches operate independently of each other, with no constraints on their architecture or structure. The only requirement is that the output of each method can be encoded as a fixed-size, one-dimensional vector, though these vectors may vary in size across methods.

Due to the scarcity and lack of quality of annotated data, both for classification and localization (see Section 2.9), in order to demonstrate the effectiveness of the proposed method, we extended our synthetic splicing dataset (see Section 5.2.1), by adding a set of copy-moved images and corresponding source/forged segmentation masks. This dataset has been used to train and evaluate both the base methods and the ensemble net. The obtained performance was excellent in the case of binary classification and localization. Also, the results showed that the ensemble approach outperforms the base methods in terms of F1-score, accuracy, and precision. Still, there is room for improvement in the task of multi-class decision and source region retrieval.

This chapter is organized as follows: Section 6.1 presents the architecture of the proposed ensemble scheme, the base methods, and the computational blocks used for multi-class decision and source region retrieval. Section 6.2 gives some details about

the synthetic dataset used for training and evaluating the proposed approach. Finally, Section 6.3 discusses the experimental results.

6.1 Method overview

The architecture and logic flow of the proposed ensemble is shown in Fig. 6.1. We use two "base" methods to perform an initial binary classification on the image (pristine vs. forged). For simplicity, we refer to these methods as Base1 and Base2, respectively. Base1 is the SIFT+DBSCAN-based approach presented in Chapter 3, and Base2 is a U-net-based network that outputs the segmentation map of the areas identified as forged.

A feature vector $x \in \mathbb{R}^{260}$ is then built by combining the outputs of the base methods and fed to a "FusionForgery" module that outputs the final probability of forgery p_f. The image is labeled pristine if $p_f \leq 0.5$ and forged otherwise. In the case of forgery, our method tries to make a further decision regarding the type of forgery, namely splicing or copy-move. This is done as follows: a "region coherence" condition between the keypoints extracted with Base1 method and the forgery segmentation mask $\mathbf{B}$ is tested. If the condition is satisfied, it means that there is an agreement between the localization obtained with keypoints matching (for copy-move detection) and the segmented forged areas. In this case, the image is labeled as copy-moved, and an estimation of the source regions used in the copy-move attack is performed. If the condition is not satisfied, further analysis is carried out to check if the image has been spliced. This is done by analyzing the consistency of the light field between the whole image and the predicted forged regions in $\mathbf{B}$. We accomplish this by means of our light direction estimation model, described in Chapter 4. If the light consistency condition is not met, the image is labeled as spliced, otherwise as "unknown forged".

In the next sections we describe, in more detail, the base methods and FusionForgery model, as long as the region coherence, source region retrieval, and splicing analysis steps.

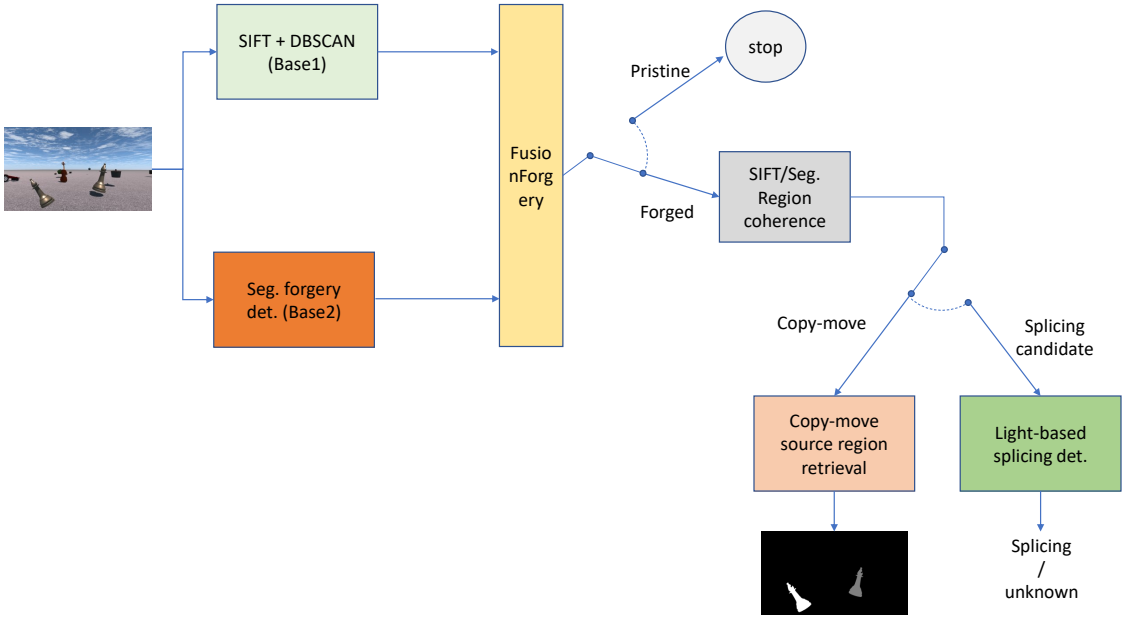


Figure 6.1: Logic flow-chart of proposed ensemble forgery detection approach.

6.1.1 Base2 method

The second forgery detection method employed in our architecture, Base2, consists of an Encoder-Decoder network, based upon a U-net, which outputs the probability map **B** of the forged zones in the image. In this implementation, we used four layers for the down-sampling branch (encoder) and four layers for the up-sampling branch (decoder). In the encoder, we used convolutional layers and max pooling, while in the decoder we used transposed convolutional layers and up-sampling layers. Skip-connections are implemented as concatenation between feature maps at the same corresponding layers in the encoder and decoder branches. The architecture is shown in Fig. 6.2. Note that, as this model is fully convolutional, multiple input resolutions can be handled.

6.1.2 FusionForgery classifier

In our ensemble architecture, we embedded a "FusionForgery" module that learns how to combine the outputs of the base methods, aiming to improve the forgery

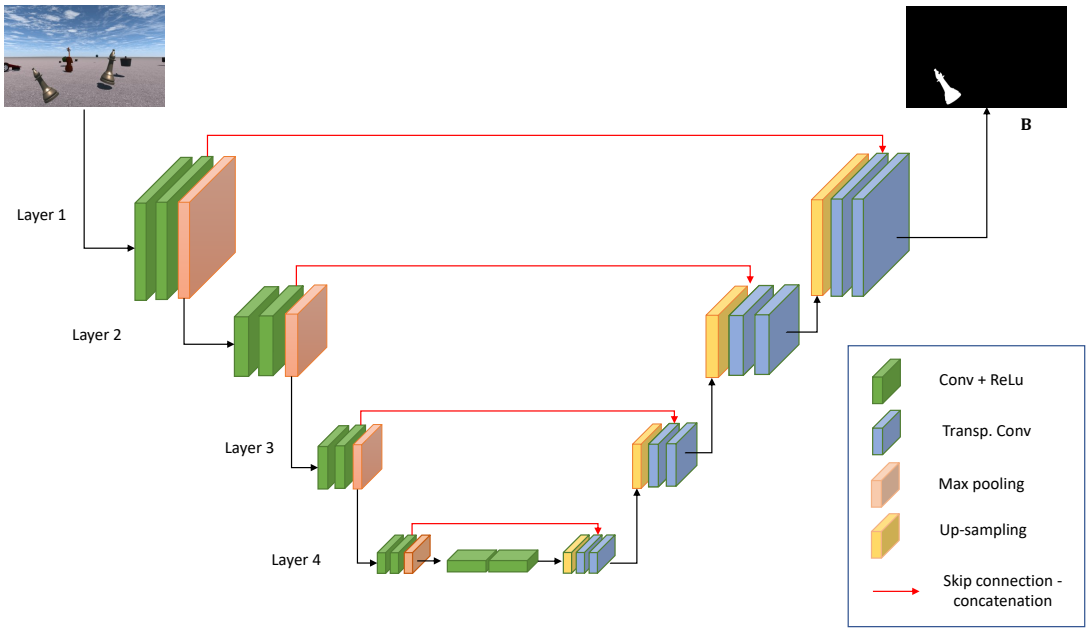


Figure 6.2: U-net architecture of "Base2" forgery localization model.

detection performance. This module consists of a simple fully connected neural network that takes as input a feature vector $x \in \mathbb{R}^{260}$ built as follows:

$$\begin{aligned} x_{1:256} &= \text{hist}(\mathbf{B}), \\ x_{257} &= p_1 \\ x_{258} &= N_{matches} \\ x_{259} &= N_{clusters} \\ x_{260} &= \alpha \end{aligned} \tag{6.1}$$

where:

- $p_1 \in \{0, 1\}$ is the binary classification obtained with the Keypoint-based algorithm (Base1)
- $N_{matches}$ and $N_{clusters}$ are, respectively, the number of keyoints' matches and clusters identified.
- α is the average number of keypoints contained in a cluster.

- hist($\mathbf{B}$) is the histogram of the forgery probability map $\mathbf{B} \in \mathbb{R}^{H \times W}$ obtained as output of the Base2 model. Note that the histogram is computed by subdividing the $[0, 1]$ interval in 256 equally sized bins.

Of course, different choices in the form of x could be used. One could, for example, encode the forgery segmentation map in a different way than just using its histogram, or use different statistics obtained as output from the keypoints-based method. However, we wanted to maintain the "philosophy" of the ensemble approach, in which typically only the final outputs of the involved methods are combined. In this way, methods for which intermediate information/feature vectors are not available could still be plugged into our architecture. Note that the flexibility of the proposed framework lies in the fact that it can be extended by including more, independent base methods. The base methods don't require a particular structure or architecture, as long as their output is binary or it can be encoded and flattened in a fixed-size feature vector. However, it will be necessary to change the input layer of the Fusion classifier according to the dimension of the new feature vector and, consequently, re-train the Fusion classifier.

The architecture of the FusionForgery model (shown in Fig. 6.3) is composed of five fully connected layers and a dropout layer in order to avoid overfitting. The output layer is composed of a neuron with a Sigmoid activation function, that outputs the final forgery probability p_f.

6.1.3 Region coherence analysis

When the image is labeled as forged by the FusionForgery classifier (*i.e.*, $p_f > 0.5$), a test is performed to assess a possible copy-move attack. To do this, we designed an algorithm that takes as input the segmentation map $\mathbf{B}$ and the list of clustered keypoints and verifies whether the areas labeled as forged in $\mathbf{B}$ are consistent with the position of the keypoints (see Fig. 6.4). If at least one cluster is found for which more than 50% of its keypoints are inside a forged region the image is labeled as copy-moved and the subsequent source region retrieval procedure is performed.

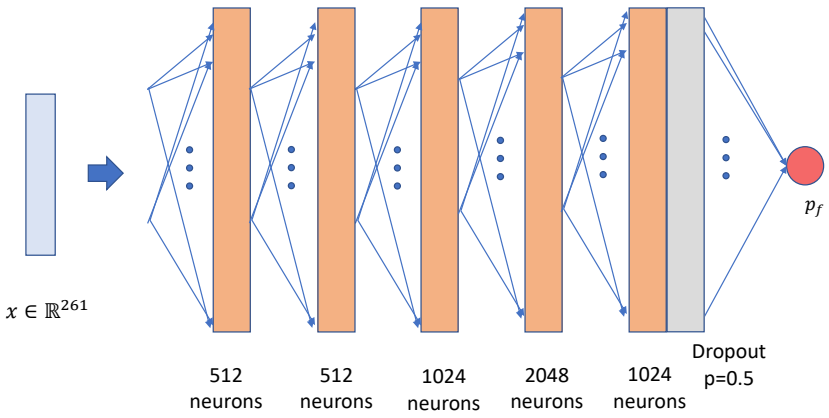


Figure 6.3: Architecture of FusionForgery model.

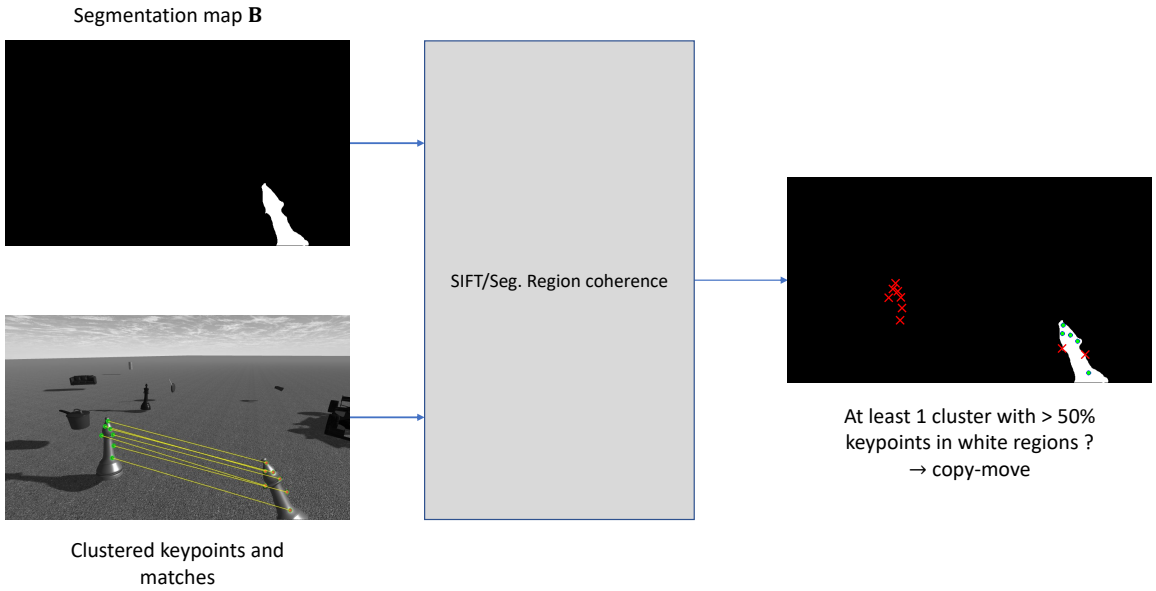


Figure 6.4: Region coherence analysis between forgery segmentation map **B** and keypoints positions.

6.1.4 Source region retrieval

This section describes the source region retrieval algorithm we developed that is applied when an image is labeled as copy-moved by the region coherence analysis. In this case, at least one cluster has more than half of its keypoints inside a forged region in $\mathbf{B}$.

Let $\mathbf{X} \in \mathbb{R}^{N\times 2}$ be the keypoints' coordinates in a cluster for which the region coherence condition is valid. Let then $\mathbf{X'} \in \mathbb{R}^{N\times 2}$ be the coordinates of the matching keypoints, *i.e.*, if $(x, y) \in \mathbf{X}$, then $(x', y') \in \mathbf{X'} \iff [(x, y), (x', y')]$ is a match.

Let us then assume that the copy-move attack has been performed by applying a linear transformation to the source region pixel coordinates. Denoting by $H \in \mathbb{R}^{2\times 3}$ the transformation matrix, we can write $\mathbf{X}'^{T} = H\mathbf{W}^{T}$, where $\mathbf{W} = [\mathbf{X}|\mathbf{1}] \in \mathbb{R}^{N\times 3}$ is the extension of $\mathbf{X}$ to homogeneous coordinates. Provided that $N \geq 3$, an estimation $\tilde{H}$ of the transformation matrix H can be obtained in a closed form by means of a least squared errors optimal solution or with a noise-robust algorithm such as RANSAC [48]. A visualization of the transformation matrix estimation is shown in Fig. 6.5.

Once we have computed the transformation matrices estimates $\tilde{H}_i$, for each cluster i that satisfies the region coherence condition, we can retrieve each source region map $\mathbf{B}_i^s$ by applying the matrices $\tilde{H}_i$ to the forgery segmentation map $\mathbf{B}$.

Note that the quality of the retrieved source region maps is heavily dependent on the clusters obtained with the Base1 method. Additionally, if the number of keypoints in a cluster is not sufficient, the matrix estimation could fail; in this case, the source region is not retrieved.

6.1.5 Splicing detection

This section describes the last computational block of our unified ensemble forgery detection approach: the splicing analysis. For simplicity, we refer to this method as Base3. This procedure is performed when the region coherence analysis gives a negative outcome, due to the non-consistency between the keypoints' positions and the forged regions identified by the Base2 method.

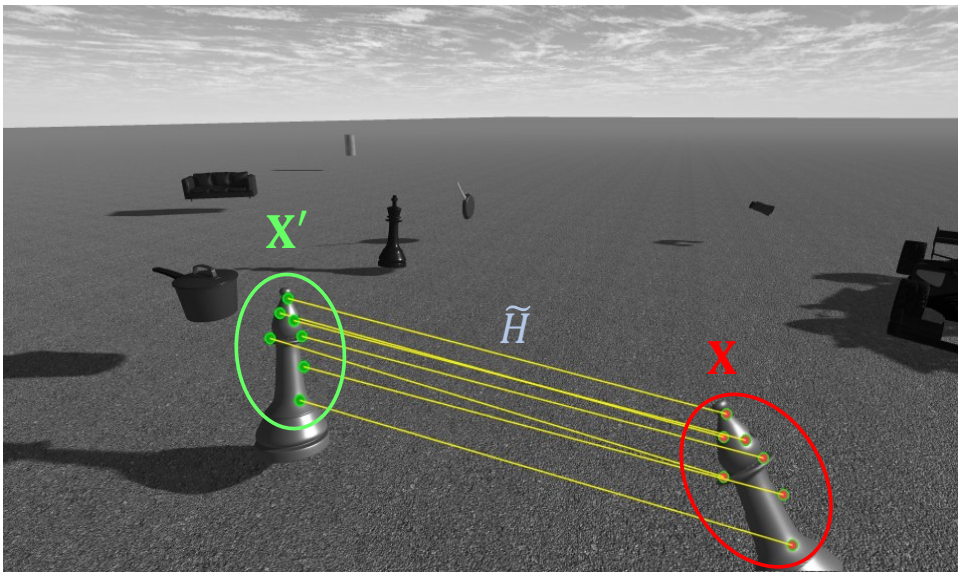


Figure 6.5: A copy-moved image with matching clusters $\mathbf{X}$ and $\mathbf{X}'$. H is the transformation matrix between the points coordinates of the two clusters.

The splicing analysis is done by checking the consistency between light direction vectors across the input image. In order to do that, we use our light direction estimation network (described in Chapter 4) as follows:

1. a connected component analysis is run on the binary forgery segmentation map $\mathbf{B}$ and the corresponding ROIs are extracted. The smallest ones (with an area under a certain threshold) are discarded.

2. If no ROIs remain, the splicing analysis cannot be done and the image is labeled as "unknown forged". Go to point 7.

3. The global 3D light direction vector $\mathbf{L}_g$ is predicted by the light estimation network on the whole image.

4. For each extracted ROI i, its local 3D light direction vector $\mathbf{L}_i$ is predicted by the light estimation network.

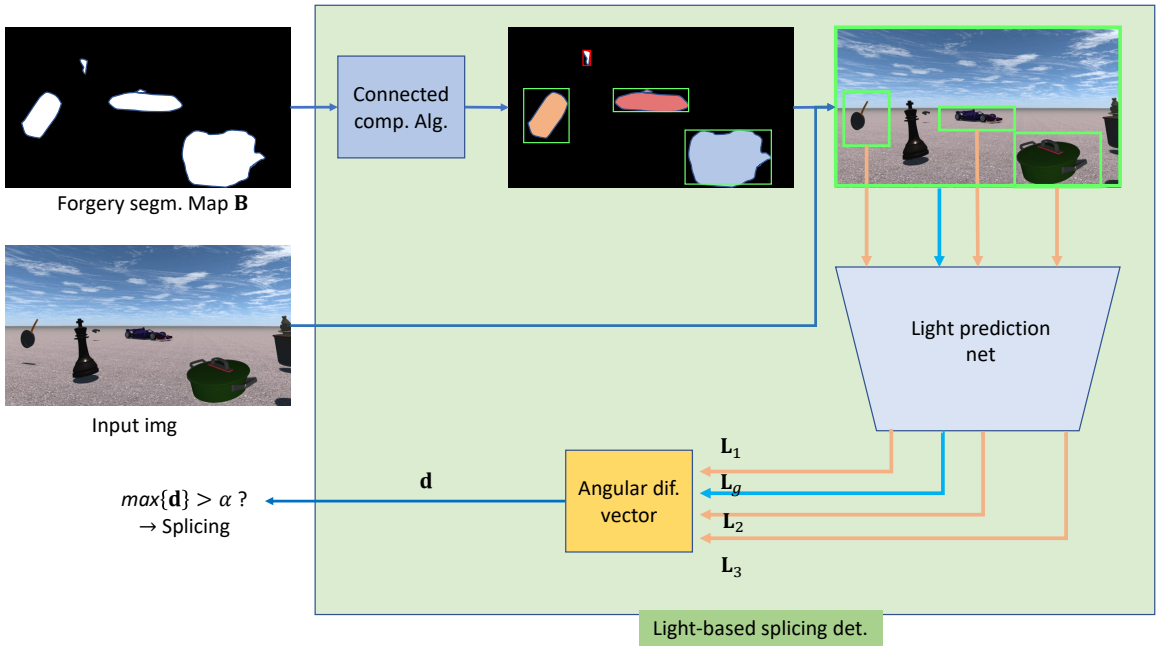


Figure 6.6: Overview of Base3 method: light-based splicing detection procedure

5. The angular difference vector $\mathbf{d} = [d_i] = [\arccos\left(\frac{\mathbf{L}_g \cdot \mathbf{L}_i}{||\mathbf{L}_g|| * ||\mathbf{L}_i||}\right)], \quad i = 1, .., N$, where N is the number of valid ROIs, is computed.

6. If $max\{\mathbf{d}\} > \alpha$ the image is labeled as spliced, as a sufficient inconsistency between the global and the local light directions is observed. Otherwise, the image is labeled as copy-moved. We set $\alpha = 40 * \frac{\pi}{180.0}$.

7. End.

A visualization of the light-based splicing detection procedure is shown in Fig. 6.6.

6.2 Dataset generation

This section gives some details about the forgery dataset we built to train and evaluate the models involved in the proposed ensemble architecture. As stated before (Section 2.9), the lack of adequately sized benchmark datasets annotated with ground truth forgery maps makes it difficult to train highly parameterized deep models, in particular for the task of forgery localization. To cope with this problem, and to assess

the validity of the proposed ensemble approach, we built a synthetic forgery dataset, including both splicing and copy-move attacks and the corresponding ground truth maps. Note that, in the case of copy-moved images, the pixels belonging to source and target regions are assigned to different labels in the ground truth maps; in this way, we will also be able to test the quality of the retrieved source regions.

We started from our Unity-generated dataset *SynthOut* (Section 4.3.1) and, by using the objects' masks (which were also generated with Unity) we created copy-moved images by selecting one object and pasting it onto a different position in the image. In order to make the detection more challenging, we also applied linear transformations such as scaling (0.5 to 1.5 factor) and rotation (up to 90 degrees) before pasting the cropped object. We also performed copy-move attacks of the "hide" type, in which a patch of background is selected and pasted onto an object in order to hide it from the scene. The complete dataset, which we refer to as *SynthOutForgery*, is created by including all the copy-moved images, the original images in the *SynthOut* dataset, and the images of the synthetic splicing dataset introduced in Section 5.2.1.

In total, the dataset consists of 45507 images, of which 3783 are spliced, 22810 are copy-moved, and 18914 are pristine. While it is true that the dataset is quite unbalanced in the number of copy-moved vs. spliced images, it must be noted that the algorithms used for differentiating between these two types of forgeries are not learned on this dataset. As such, this is not a source of bias towards copy-move decision.

The dataset was split into train, validation, and test subsets, by a proportion of 70%, 10%, and 20%, respectively. The Base2 model was trained on this dataset. The FusionForgery classifier, instead, was trained on a dataset obtained by computing the feature vectors x of all train subset images (as in Eq. (6.1)) and associating the corresponding binary ground truth label.

6.3 Experimental results

This section describes the detection results and the performance of our ensemble approach on the unified synthetic forgery dataset *SynthOutForgery* (see Section 6.2).

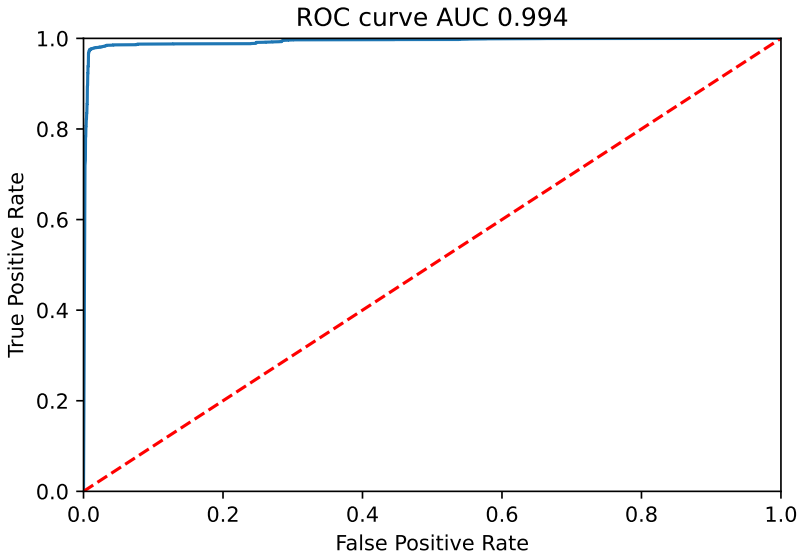


Figure 6.7: ROC curve obtained with proposed ensemble approach on *SynthOutForgery* dataset.

The classification performance is reported both in terms of binary prediction (pristine vs. forged) and in terms of multi-class prediction (pristine vs. copy-move vs. splicing). The quality of forgery localization of our method is also analyzed, both for forged regions and source regions (in the case of copy-moved images). The contribution of each of the two base methods is also investigated by (i) comparing their individual performance and the performance of the complete ensemble approach and (ii) through an ablation analysis in which one of the two base methods is excluded from the architecture. Both experimental procedures effectively demonstrate the efficacy of the ensemble strategy.

6.3.1 Binary forgery classification and localization performance

The binary classification performance of the ensemble approach are as follows: accuracy: 0.965, precision: 0.992, and F1-score: 0.984. The ROC curve and the associated AUC of 0.994 (Fig. 6.7) further confirm the good detection properties of the method.

In order to evaluate the performance of localization of forged areas, we used the

mean Intersection over Union metric (IoU), defined as follows:

$$\begin{aligned} \text{IoU} &= \frac{|\{(i,j) \mid \mathbf{B}(i,j) = 1 \wedge \mathbf{B}_{gt}(i,j) = 1\}|}{|\{(i,j) \mid \mathbf{B}(i,j) = 1 \vee \mathbf{B}_{gt}(i,j) = 1\}|} \\ \text{mean IoU} &= \text{avg}\{\text{IoU}_i\}, \quad \forall i \in \text{Test set} \end{aligned} \tag{6.2}$$

where $\mathbf{B}$ and $\mathbf{B}_{gt}$ are the predicted and ground truth forgery maps, respectively.

The obtained mean IoU is 0.945.

6.3.1.1 Ablation analysis

In order to investigate the contribution of each of the base methods employed in the architecture and the effectiveness of the FusionForgery net, we carried out an ablation study, in which we compared the performance of the two base methods alone and the ensemble architecture excluding either Base1 or Base2. For reference, we also compare the results of another forgery detection model we designed - "Base CNN", used as baseline. This is a standard Convolutional Neural Network that we trained from scratch on the *SynthOutForgery* dataset (its architecture is reported in Fig. 6.8).

The performance in terms of F1-score obtained by the different approaches on *SynthOutForgery* is reported in Table 6.1. As it can be seen, the complete ensemble approach outperforms all of the other methods. The difference in performance between the complete ensemble approach and the one without Base1 is small though, meaning that Base1 is not significantly contributing to the overall performance, as can be inferred also by the fact that the obtained F1-score value (0.751) as a standalone method is not at the same level of the other approaches. Nevertheless, this method is still crucial for the subsequent step of forgery localization in the case of copy-move decision. In particular, the matches between keypoints are needed for the source regions retrieval (see Section 6.1.4). This analysis also reveals how significant is the contribution given by the Base2 method. In fact, when removed from the ensemble, the F1-score drops. This means that the information carried out by the segmentation map is useful, even for the purpose of global binary classification. Finally, note how combining the two approaches through the FusionForgery network allows for better performance compared

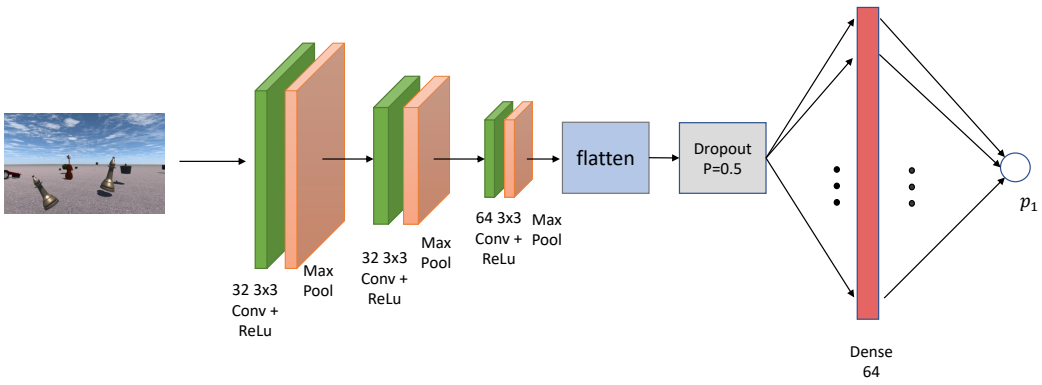


Figure 6.8: Architecture of "Base CNN" forgery detection model.

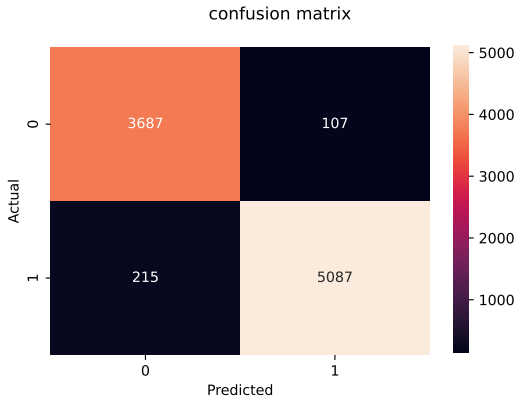
to the baseline architecture ("Base CNN"). The same considerations can be drawn by analyzing the confusion matrices shown in Fig. 6.9.

6.3.2 Splicing detection performance

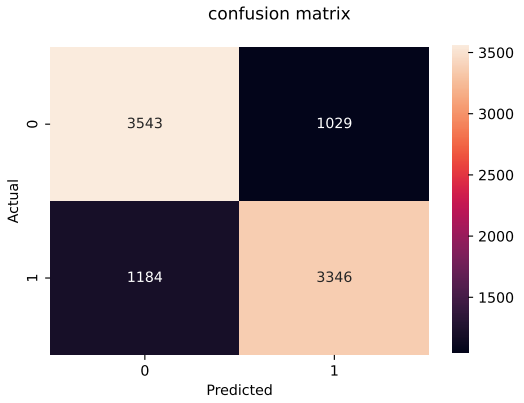
The proposed method also attempts to distinguish between forgery of type copy-move and splicing. We used the term "attempts" because, as stated in Section 6.1.5, the splicing detection algorithm cannot always be applied (this is the case when no valid

Method	**F1-score**
Base CNN	0.969
Base1	0.751
Ensemble No Base2	0.746
Ensemble No Base1	0.981
Ensemble	**0.984**

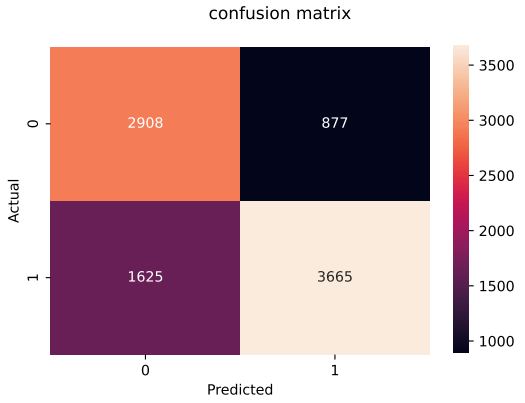
Table 6.1: Results of ablation study on *SynthOutForgery*.



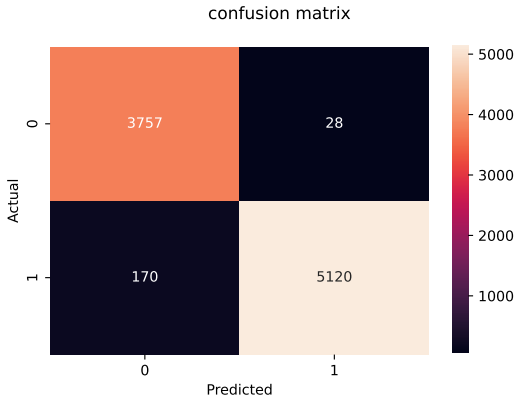
(a) "Base CNN method" (Section 6.3.1.1).



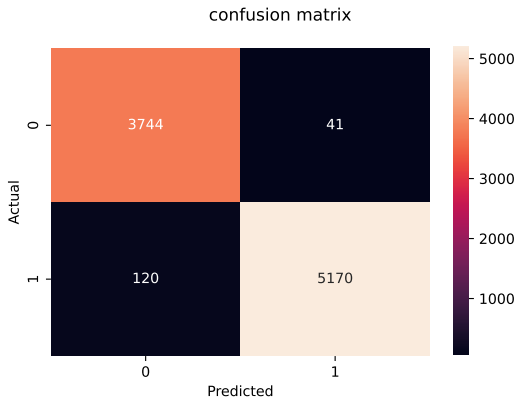
(b) Base1 method (keypoints-based + clustering. Chapter 3).



(c) Ensemble without Base2.



(d) Ensemble without Base1.



(e) Ensemble (Section 6.1.2).

Figure 6.9: Comparison of confusion matrices (0: pristine, 1: forged) obtained on *SynthOutForgery* with different approaches in the ablation analysis.

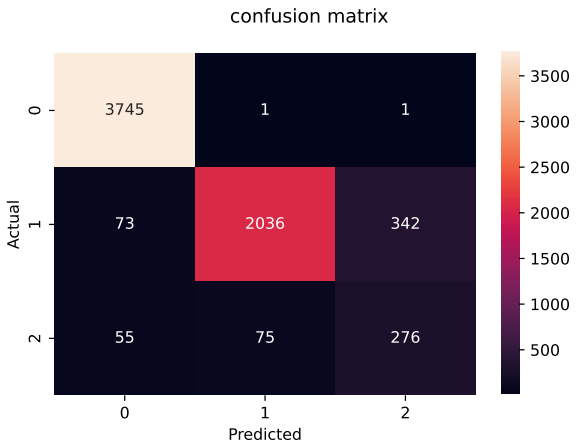


Figure 6.10: Multi-class Confusion matrix obtained with the proposed ensemble method on *SynthOutForgery*. The classes are 0: pristine, 1: copy-moved (forgery type 1), 2: spliced (forgery type 2). In this evaluation, we didn't consider the images predicted as "unknown-type forged" (27.2% on the total of test images).

regions can be extracted for light estimation from the forgery map **B**). In this case, the image is labeled as "unknown forgery" ("3"). The percentage of testing images predicted as unknown forgery is 27.2%.

In Fig. 6.10 a confusion matrix with three classes (0: pristine, 1: copy-move, 2: splicing) is shown. Note that in the confusion matrix computation, we are not considering the images predicted as unknown. As can be seen, there is still room for improvement, as a lot of copy-moved images are predicted as spliced. Also, as stated before, the percentage of images lacking a further decision between splicing and copy-move decision is high.

6.3.3 Source region localization performance

In this section, we analyze the source region localization properties of the proposed method in the case of images labeled as copy-moved.

Similarly to what we did to assess the quality of localization of forged areas, we use the mean IoU metric. In this case, however, we use, as ground truth, binary maps where only the source areas are set as "1", and all the other pixels are assigned a "0" label. Note that, as stated in Section 6.1.4, the source region cannot always be

retrieved; this is the case when the number of valid matching keypoints is not enough to obtain a reliable transformation matrix. The percentage of correctly predicted copy-moved images for which the source region retrieval could not be done is still high - 50.5%. Nevertheless, in the cases where the region can be reconstructed, we get a very good mean IoU performance of 0.832. Examples of correctly predicted source regions is shown in Fig. 6.11, while some of the (few) fail cases are depicted in Fig. 6.12.

6.4 Discussion

In this chapter, an ensemble architecture to combine different forgery detection methods was presented, with the aim of achieving better performance.

A peculiarity of the proposed method is that, in the case of forged-classified images, it attempts to provide a more detailed classification regarding the type of forgery, differentiating between splicing and copy-move attacks. Also, localization of the forged areas and the source regions (in the copy-move case) is given as output.

The proposed ensemble architecture achieved excellent performance for the binary classification and localization task (forged vs. pristine). Furthermore, it outperformed all the individual base methods involved, in terms of F1-score, confirming the validity of the fusion strategy.

Nevertheless, there is still room for improvement in the task of distinguishing between splicing and copy-move attacks, as the performance is still lacking. This is due to the fact that the splicing detection approach based on the light inconsistency analysis performs well only when the light conditions of the target image and the spliced portion, are different "enough".

Another point of improvement of the proposed approach relates to the task of source region localization (for copy-moved images). As mentioned in Section 6.3, in fact, in half of the cases of correctly labeled copy-moved images, the source region retrieval algorithm could not be applied due to the lack of valid matched keypoints obtained with the Base2 method. In the other cases, though, the source region is reconstructed with good accuracy. Lastly, it is important to note the limitations of

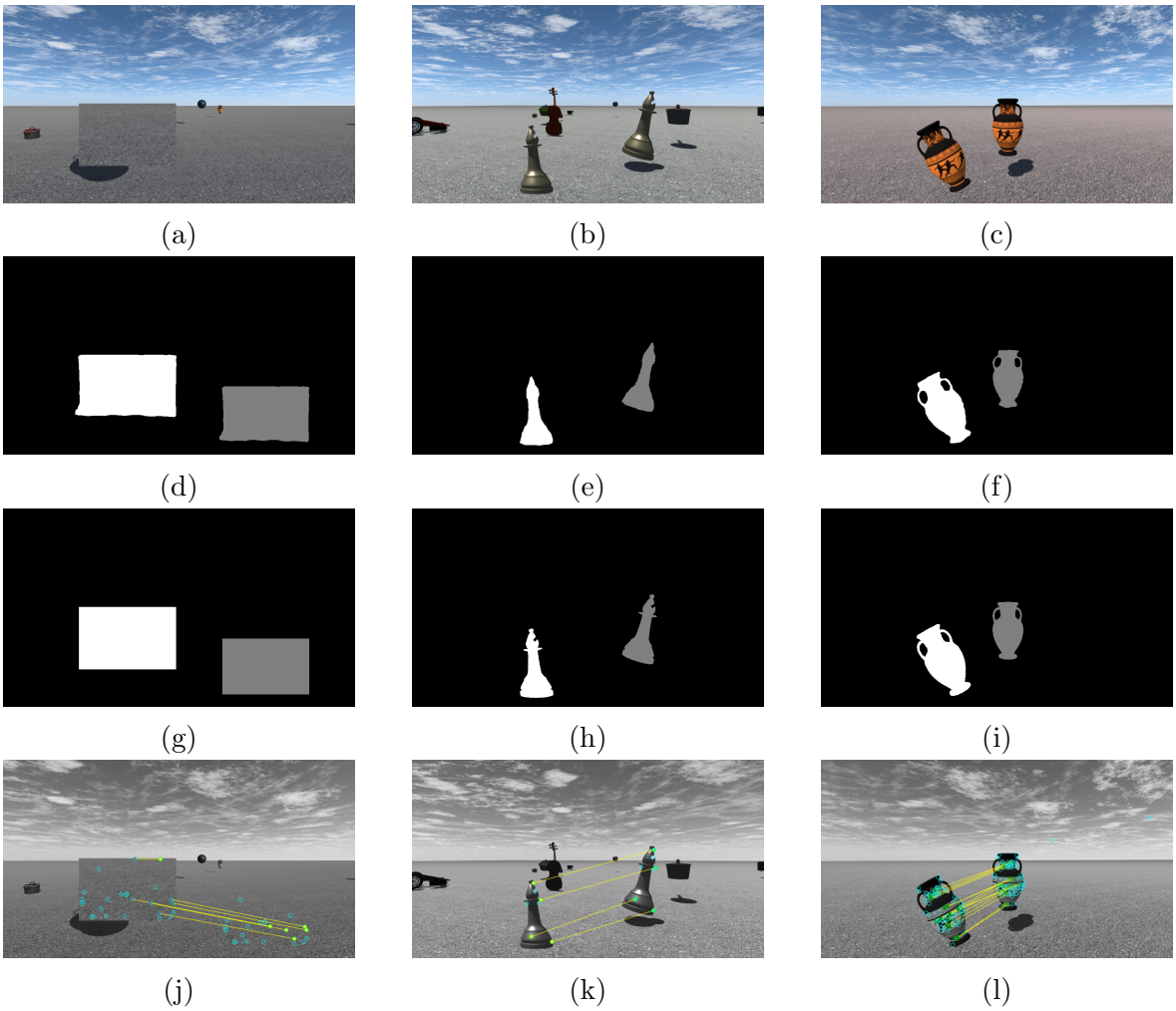


Figure 6.11: Examples of source (gray) /forged (white) regions maps correctly predicted by the proposed source region retrieval method in copy-moved images. Figs. a - c show the input images, while the predicted maps and the corresponding ground truth are shown in Figs. d - f and g - e, respectively. Finally, Figs. g - h show the results of the keypoints-based method.

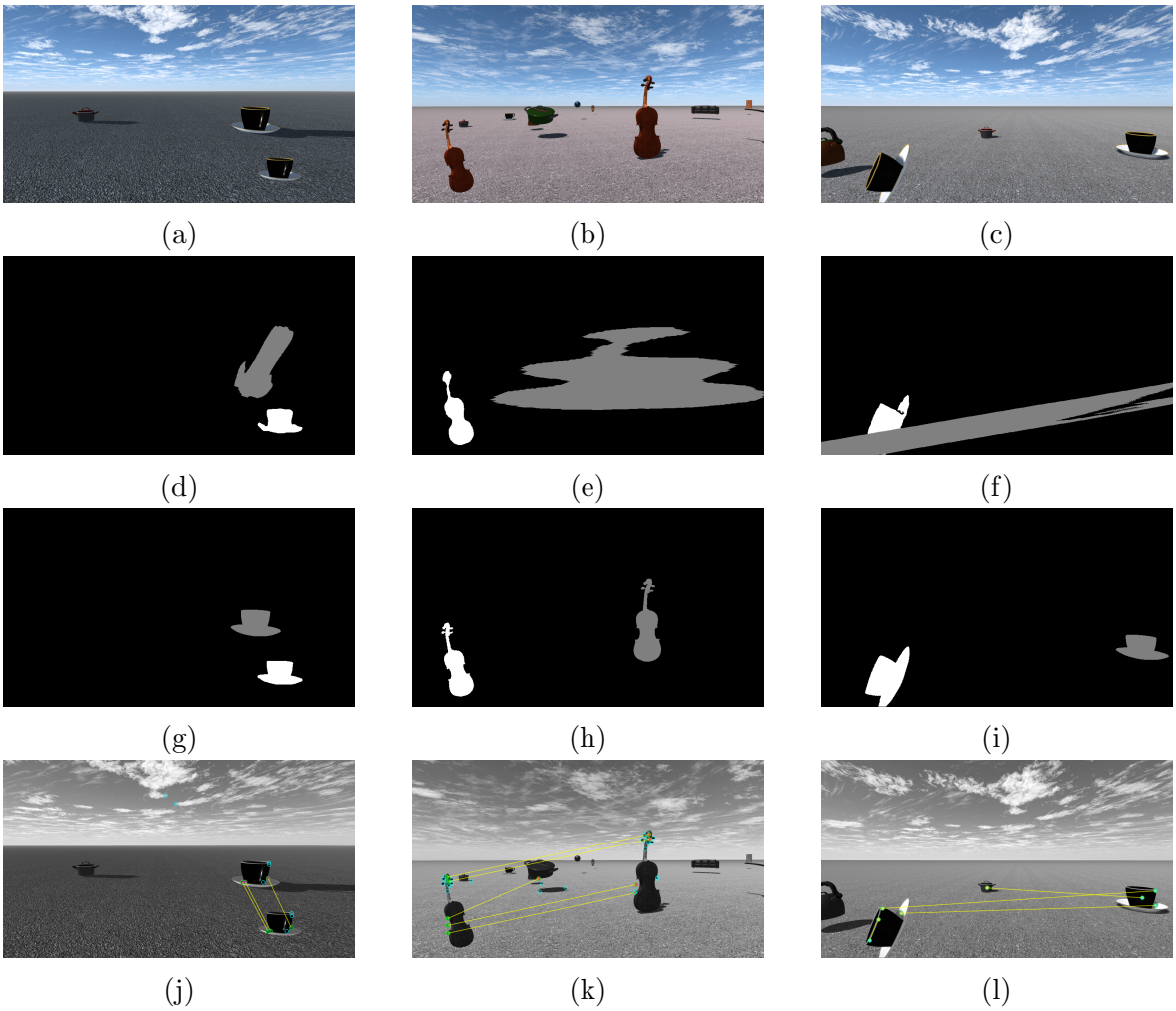


Figure 6.12: Examples of source (gray) /forged (white) regions maps wrongly predicted with proposed source region retrieval method in copy-moved images. Figs. a - c show the input images, while the predicted maps and the corresponding ground truth are shown in Figs. d - f and g - e, respectively. Finally, Figs. g - h show the results of the keypoints - based method.

our proposed approach when applied to compressed images. Since the components of the architecture were exclusively trained on our *SynthOutForgery* dataset comprising uncompressed PNG images, the method may not perform optimally on JPG images with varying compression levels. To enhance its generalization capabilities on this type of data, future work will focus on creating a more diverse dataset that includes images with different levels of compression and training the proposed architecture on this dataset.

CONCLUSIONS

The main contribution of this thesis is the development of a unified ensemble approach for forgery detection and localization in digital images. In particular, we focused our attention on the detection of two of the most common, yet effective types of forgery attacks: copy-move and splicing. Throughout this thesis, we reviewed, compared, and developed a set of "base" forgery detection and localization approaches, that were used as building blocks in our ensemble architecture to improve the performance of the single methods.

In chapter 2, we performed extensive research on the state of the art in the field of forgery detection, with a particular interest in the most recent deep-learning-based approaches. A comparison of their performance was carried out on the most cited benchmark datasets in this field. This analysis highlighted the fact that, even if most of these approaches seem promising, they can hardly be compared in terms of performance, as they are typically evaluated on some custom-built datasets or only on a few of the benchmark ones. This is usually done because, as these are typically supervised deep learning-based methods, they require a great amount of precisely annotated data, which is not the case for the task of forgery detection. The lack of adequately sized and precisely annotated data is, in fact, one of the points that pushed us towards the implementation of two data generation algorithms, both for training and evaluation of deep learning approaches.

Chapter 3 presents a keypoint-based copy-move algorithm that uses density-based clustering to filter out weak matches between keypoints. This method was shown to perform really well on two benchmark datasets, outperforming one of the most cited state of the art method, in terms of precision and accuracy.

Chapter 4 presents a novel architecture to estimate the 3D light direction in outdoor images. The main peculiarity of this approach is that it is a fusion of a data-driven, learning-based approach, and a physically inspired illumination model. In order to train this architecture, we designed two data generation algorithms, that allowed us to build two datasets with a great number of precisely annotated light information. The first, *SynthOut*, is a computer graphic rendered dataset, in which outdoor scenes involving simple, yet realistic objects are depicted in a great variety of lighting conditions. The second, *RealOut*, is a set of outdoor realistic images automatically extracted from 360-degrees high-resolution panoramas in latitude-longitude format. The light direction is automatically inferred from each panorama from our generation algorithm. As a matter of fact, the two data generation algorithms also constitute an important contribution, as they address the problem of data scarcity even in the research field of intrinsic scene understanding and, more in general, inverse rendering of digital images. We trained and evaluated the proposed method on the two generated datasets, achieving satisfactory regression performance. Also, through an extensive ablation study, we demonstrated how the embedding of the physical model in our architecture effectively improves performance with respect to a data-driven-only approach.

The light direction estimation model has been used as the main building block in a splicing detection approach, presented in chapter 5. This method aims to detect spliced regions by checking the (in)consistency of 3D light direction vectors across multiple patches of the input image with respect to the global light direction (relative to the whole image). A feature vector is then built by embedding the extracted vectors and a classifier is trained to output binary prediction: pristine vs. spliced. The prediction performance was satisfactory on a synthetic splicing dataset that we derived from the *SynthOut*. Though, the model's performance was not as good on CASIA2, which is one of the most challenging benchmark datasets. This is due to the fact that our approach relies on the light estimation task, which in turn, requires the involved light estimation network to be trained on the target data. This was not possible in the case of most benchmark datasets, as the ground truth forgery maps are usually not available.

Finally, chapter 6 details the architecture of the proposed unified ensemble scheme for forgery detection and localization. In this architecture, a module is trained to optimally combine the outputs of two "base" methods and assign a binary label (forged vs. pristine) to the input image. The binary forgery map is also given as output. Then, a "region coherence" analysis is performed on the forgery map in order to assess the presence of a copy-move attack. In the positive case, the proposed method attempts to reconstruct the source regions. In the negative case, a light-based splicing detection, based on the architecture discussed in chapter 4 is performed, analyzing the (in)consistency between the predicted light direction vectors across multiple patches in the image. The models and computational blocks involved in the ensemble scheme need different types of ground truth information (including light direction, source/forgery localization maps, and spliced vs. copy-moved labels). Hence, in order to assess the validity of the proposed approach, we generated a big, unified, high-resolution synthetic forgery detection dataset that includes pristine, copy-moved, and spliced images, along with the associated forgery localization masks. We then trained and evaluated the proposed method on this dataset, achieving really good detection performance and localization for the binary prediction task (forged vs. pristine). Through an ablation study, we showed how the achieved performance of the complete ensemble framework is higher than the "base" forgery detection methods developed in this work.

A peculiarity of the ensemble approach is its modularity. In fact, more "base" forgery detection methods can easily be included in our scheme by embedding their outputs in the feature vector used by the network that combines all the outputs of "base" detectors.

Limitations and future directions

In this last paragraph, we highlight some possible future research directions and challenges. We start by analyzing some of the current limitations of the proposed ensemble architecture and by proposing possible improvements and extensions. Then, we

focus our attention on possible research directions to take into account more sophisticated and powerful technologies that leverage generative A.I. for creating realistic fake content.

1. The ensemble architecture should be expanded by including more base forgery detection methods.

2. The classification of the type of attack, *i.e.*, copy-move vs. splicing should be improved, possibly by using/combining more sophisticated splicing detection methods.

3. The mechanism to retrieve the source region in the case of copy-move forgeries should be refined in order to be more robust when not many keypoints' matches are found.

In this work, we didn't address more sophisticated and powerful attacks such as GAN/AI-generated content, adversarial attacks, and augmented reality. In particular, the rapid development of generative approaches like diffusion-based models (*e.g.,* DALL-E2) has made it increasingly difficult to distinguish between real and generated content, posing new challenges for forgery detection methods. Also, because these models are evolving at a pace unthinkable just until a couple of years ago, mainly thanks to the hype about A.I. among big-tech companies, the media and the general public, new detection approaches should specifically address this kind of technology and evolve at a similar pace.

One approach to detect AI-generated content could be to leverage *adversarial training*, where a forgery detection model is trained concurrently with a generative model (in a GAN fashion). This could help the detector learn to identify the subtle artifacts and patterns associated with AI-generated forgeries. Conversely, an attacker could make use of *adversarial attacks* (see Section 2.9) that are specifically designed to fool forgery detection methods.

Let's consider now the DALL-E2 model. DALL-E2 images are created by a diffusion model, which involves applying a series of random transformations to a small initial image. As a result, these images may have certain patterns or textures that are unique to the specific diffusion model employed. By analyzing the statistical properties of these images, it may be possible to detect these unique patterns and use them as a clue for a generated image.

Finally, to better evaluate the performance of forgery detection methods on AI-generated content, it is crucial to create new benchmark datasets containing a diverse range of images generated by various generative models such as diffusion models. This will allow researchers to systematically compare the performance of different methods and identify areas for improvement. Similarly as for the detection methods, these datasets should be constantly updated by including content generated with newer versions of the aforementioned models or with newer approaches that generate even better content.

BIBLIOGRAPHY

[1] 2022 ieee video and image processing cup - synthetic image detection challenge. https://grip-unina.github.io/vipcup2022/. [Accessed: 05-may-2023].

[2] Adobe Photoshop. https://www.adobe.com/it/products/photoshop.html. [Accessed: 16-march-2022].

[3] Blog post on Elcomsoft, April 2011. https://blog.elcomsoft.com/2011/04/nikon-image-authentication-system-compromised/. [Accessed: 16-march-2022].

[4] Faceswap. https://github.com/deepfakes/faceswap. [Accessed: 16-march-2022].

[5] Gimp. https://www.gimp.org/. [Accessed: 16-march-2022].

[6] Interactive Web demo: Whichfaceisreal. https://www.whichfaceisreal.com/index.php. [Accessed: 16-march-2022].

[7] Keek. https://keeex.me/products/. [Accessed: 16-march-2022].

[8] Numbersprotocol.io. https://numbersprotocol.io/. [Accessed: 16-march-2022].

[9] Online article on Arstechnica, May 2007. https://arstechnica.com/uncategorized/2007/05/latest-aacs-revision-defeated-a-week-before-release/. [Accessed: 16-march-2022].

[10] Single image surface normal estimation. https://github.com/dfan/single-image-surface-normal-estimation. [Accessed: 30-june-2022].

[11] Younis Abdalla, Tariq Iqbal, and Mohamed Shehata. Copy-move forgery detection and localization using a generative adversarial network and convolutional neural-network. *Information*, 10:286, 09 2019.

[12] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurélien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. *Technical report, EPFL*, 2010.

[13] Ritu Agarwal and Om Verma. An efficient copy move forgery detection using deep learning feature extraction and matching algorithm. *Multimedia Tools and Applications*, 79, 03 2020.

[14] Irene Amerini, Lamberto Ballan, Roberto Caldelli, Alberto Del Bimbo, and Giuseppe Serra. A SIFT-based forensic method for copy-move attack detection and transformation recovery. *IEEE Transactions on Information Forensics and Security*, pages 1099–1110, 2011.

[15] Michael Konrad Arnold, Martin Schmucker, and Stephen D Wolthusen. *Techniques and applications of digital watermarking and content protection*. Artech House, 2003.

[16] M. Barni, Q. T. Phan, and B. Tondi. Copy move source-target disambiguation through multi-branch cnns. *IEEE Transactions on Information Forensics and Security*, 16:1825–1840, 2021.

[17] Patrick Bas, Tomáš Filler, and Tomáš Pevnỳ. "Break Our Steganographic System": the ins and outs of organizing BOSS. In *International Workshop on Information Hiding*, pages 59–70, 2011.

[18] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

[19] Shida Beigpour, Andreas Kolb, and Sven Kunz. A comprehensive multi-illuminant dataset for benchmarking of the intrinsic image algorithms. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 172–180, 2015.

[20] Gajanan K Birajdar and Vijay H Mankar. Digital image forgery detection using passive techniques: A survey. *Digital investigation*, 10(3):226–245, 2013.

[21] Giulia Boato, Cecilia Pasquini, Antonio L. Stefani, Sebastiano Verde, and Daniele Miorandi. Trueface: a dataset for the detection of synthetic face images from social networks. In *2022 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–7, 2022.

[22] Zhe Cao, Hang Gao, Karttikeya Mangalam, Qi-Zhi Cai, Minh Vo, and Jitendra Malik. Long-term human motion prediction with scene context. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV*, pages 387–404, 2020.

[23] Jiaxin Chen, Xin Liao, and Zheng Qin. Identifying tampering operations in image operator chains based on decision fusion. *Signal Processing: Image Communication*, 95:116287, 2021.

[24] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. 04 2016.

[25] Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. Single-image depth perception in the wild. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. [Accessed: 08-december-2022].

[26] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. pages 1800–1807, 07 2017.

[27] V. Christlein, C. Riess, and E. Angelopoulou. On rotation invariance in copy-move forgery detection. In *2010 IEEE International Workshop on Information Forensics and Security*, pages 1–6, 2010.

[28] V. Christlein, C. Riess, J. Jordan, C. Riess, and E. Angelopoulou. An evaluation of popular copy-move forgery detection approaches. *IEEE Transactions on Information Forensics and Security*, 7(6):1841–1854, 2012.

[29] D. Cozzolino and L. Verdoliva. Noiseprint: A cnn-based camera model fingerprint. *IEEE Transactions on Information Forensics and Security*, 15:144–159, 2020.

[30] Duc-Tien Dang-Nguyen, Cecilia Pasquini, Valentina Conotter, and Giulia Boato. Raise: A raw images dataset for digital image forensics. In *Proceedings of the 6th ACM Multimedia Systems Conference*, MMSys '15, page 219–224, New York, NY, USA, 2015. Association for Computing Machinery.

[31] Tiago José de Carvalho, Christian Riess, Elli Angelopoulou, Hélio Pedrini, and Anderson de Rezende Rocha. Exposing digital image forgeries by illumination color classification. *IEEE Transactions on Information Forensics and Security*, 8(7):1182–1194, 2013.

[32] Loïc Dehan, Wiebe Van Ranst, Patrick Vandewalle, and Toon Goedemé. Complete and temporally consistent video outpainting. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 686–694, 2022.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[34] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *CoRR*, abs/2105.05233, 2021.

[35] Jana Dittmann. Content-fragile watermarking for image authentication. In *Security and Watermarking of Multimedia Contents III*, volume 4314, pages 175–184. International Society for Optics and Photonics, 2001.

[36] Amit Doegar, M. Dutta, and K. Gaurav. Cnn based image forgery detection using pre-trained alexnet model. *Electronic*, 2019.

[37] Brian Dolhansky, Russ Howes, Ben Pflaum, Nicole Baram, and Cristian Ferrer. The deepfake detection challenge (dfdc) preview dataset, 10 2019.

[38] J. Dong, W. Wang, and T. Tan. Casia image tampering detection evaluation database. In *2013 IEEE China Summit and International Conference on Signal and Information Processing*, pages 422–426, July 2013.

[39] Chau Xuan Truong Du, Le Hoang Duong, Huynh Thanh Trung, Pham Minh Tam, Nguyen Quoc Viet Hung, and Jun Jo. Efficient-frequency: a hybrid visual forensic framework for facial forgery detection. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 707–712, 2020.

[40] Majed El Helou, Ruofan Zhou, Johan Barthas, and Sabine Süsstrunk. VIDIT: Virtual image dataset for illumination transfer. *arXiv preprint arXiv:2005.05460*, 2020.

[41] Mohamed Elaskily, Heba Elnemr, Ahmed Sedik, Mohamed Dessouky, Ghada El Banby, Osama Elaskily, Ashraf A. M. Khalaf, Heba Aslan, Osama Faragallah, and Fathi Abd El-Samie. A novel deep learning framework for copy-move forgery detection in images. *Multimedia Tools and Applications*, 79, 03 2020.

[42] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. *CoRR*, abs/2012.09841, 2020.

[43] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[44] Kevin Eykholt, I. Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Xiaodong Song. Robust physical-world attacks on deep learning visual classification. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018.

[45] H. Farid. Detecting digital forgeries using bispectral analysis. *AI Lab, Massachusetts Institute of Technology, Tech. Rep. AIM-1657*, 1999.

[46] Hany Farid. Image forgery detection: A survey. *Signal Processing Magazine, IEEE*, 26:16 – 25, 04 2009.

[47] Hany Farid. Lighting (in)consistency of paint by text. *ArXiv*, abs/2207.13744, 2022.

[48] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[49] J. Fridrich, M. Chen, and M. Goljan. Imaging sensor noise as digital x-ray for revealing forgeries. *Proc. 9th Int. Workshop on Information Hiding, Sant Malo, France*, pages 342–358, 2007.

[50] J. Fridrich, D. Soukal, and J. Lukás. Detection of copy move forgery in digital images. *Proc. Digital Forensic Research Workshop*, 2003.

[51] Penglei Gao, Xi Yang, Rui Zhang, John Y. Goulermas, Yujie Geng, Yuyao Yan, and Kaizhu Huang. Generalized image outpainting with u-transformer. *Neural Networks*, 162:1–10, 2023.

[52] Eric Goldman. The complicated story of FOSTA and Section 230. *First Amend. L. Rev.*, 17:279, 2018.

[53] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 2014.

[54] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv 1412.6572*, 12 2014.

[55] Zhiqing Guo, Gaobo Yang, Dengyong Zhang, and Ming Xia. Rethinking gradient operator for exposing ai-enabled face forgeries. *Expert Systems with Applications*, 215:119361, 2023.

[56] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[57] Geoffrey Hinton, Alex Krizhevsky, and Sida Wang. Transforming auto-encoders. volume 6791, pages 44–51, 06 2011.

[58] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020.

[59] Yannick Hold-Geoffroy, Akshaya Athawale, and Jean-François Lalonde. Deep sky modeling for single image outdoor lighting estimation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6920–6928, 2019.

[60] Yannick Hold-Geoffroy, Kalyan Sunkavalli, Sunil Hadap, Emiliano Gambaretto, and Jean-François Lalonde. Deep outdoor illumination estimation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2373–2382, 2017.

[61] Berthold Horn and Brian Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 08 1981.

[62] Lukas Hosek and Alexander Wilkie. An analytic model for full spectral sky-dome radiance. *ACM Transactions on Graphics - TOG*, 31, 07 2012.

[63] Tu K Huynh, Khoa V Huynh, Thuong Le-Tien, and Sy C Nguyen. A survey on image forgery detection techniques. In *The 2015 IEEE RIVF International Conference on Computing & Communication Technologies-Research, Innovation, and Vision for Future (RIVF)*, pages 71–76. IEEE, 2015.

[64] M. K. Johnson and H. Farid. Exposing digital forgeries by detecting inconsistencies in lighting. *Proc. ACM Multimedia and Security Workshop, New York, NY*, pages 1–10, 2005.

[65] M. K. Johnson and H. Farid. Exposing digital forgeries through chromatic aberration. *Proc. ACM Multimedia and Security Workshop, Geneva, Switzerland*, pages 48–55, 2006.

[66] M. K. Johnson and H. Farid. Metric measurements on a plane from a single image. *Tech. Rep. TR2006- 579*, 2006.

[67] M. K. Johnson and H. Farid. Detecting photographic composites of people. *Proc. 6th Int. Workshop on Digital Watermarking, Guangzhou, China*, 2007.

[68] M. K. Johnson and H. Farid. Exposing digital forgeries through specular highlights on the eye. *Proc. 9th Int. Workshop on Information Hiding, Saint Malo, France,*, pages 311–325, 2007.

[69] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. In *Proc. NeurIPS*, 2021.

[70] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. pages 4396–4405, 06 2019.

[71] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *CoRR*, abs/1912.04958, 2019.

[72] Lei Ke, Yu-Wing Tai, and Chi-Keung Tang. Occlusion-aware video object inpainting. *CoRR*, abs/2108.06765, 2021.

[73] Katarzyna Koptyra and Marek R Ogiela. Imagechain—application of blockchain technology for images. *Sensors*, 21(1):82, 2021.

[74] P. Korus and J. Huang. Multi-scale analysis strategies in prnu-based tampering localization. *IEEE Trans. on Information Forensics & Security*, 2017.

[75] Paweł Korus. Digital image integrity–a survey of protection and verification techniques. *Digital Signal Processing*, 71:1–26, 2017.

[76] Paweł Korus and Jiwu Huang. Evaluation of random field models in multi-modal unsupervised tampering localization. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2016.

[77] Marek Kowalski. [accessed 16-march-2022].

[78] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research), 2009.

[79] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 2012.

[80] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. 07 2016.

[81] Jean-François Lalonde, Louis-Philippe Asselin, Julien Becirovski, Yannick Hold-Geoffroy, Mathieu Garon, Marc-André Gardner, and Jinsong Zhang. The Laval HDR sky database. http://sky.hdrdb.com, 2016.

[82] Jean-François Lalonde and Iain Matthews. Lighting estimation in outdoor image collections. *Proceedings - 2014 International Conference on 3D Vision, 3DV 2014*, pages 131–138, 02 2015.

[83] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[84] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010. [accessed 16-march-2022].

[85] Yuezun Li, Ming-Ching Chang, and Siwei Lyu. In ictu oculi: Exposing ai created fake videos by detecting eye blinking. pages 1–7, 12 2018.

[86] Yuezun Li and Siwei Lyu. Exposing deepfake videos by detecting face warping artifacts, 11 2018.

[87] Yuezun Li, Xin Yang, Honggang Qi, and Siwei Lyu. Celeb-df: A large-scale challenging dataset for deepfake forensics. pages 3204–3213, 06 2020.

[88] Xin Liao, Zihang Huang, Lin Peng, and Tong Qiao. First step towards parameters estimation of image operator chain. *Information Sciences*, 575, 06 2021.

[89] Xin Liao, Kaide Li, Xinshan Zhu, and K. J. Ray Liu. Robust detection of image operator chain with two-stream convolutional neural network. *IEEE Journal of Selected Topics in Signal Processing*, 14(5):955–968, 2020.

[90] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[91] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[92] Guilin Liu, Fitsum Reda, Kevin Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. 04 2018.

[93] Xosé López-García, Alba Silva-Rodríguez, Ángel-Antonio Vizoso-García, Oscar Westlund, and João Canavilhas. Mobile journalism: Systematic literature review. *Comunicar. Media Education Research Journal*, 27(1), 2019.

[94] David Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–, 11 2004.

[95] Chun-Shien Lu and H-YM Liao. Multipurpose watermarking for image authentication and protection. *IEEE Transactions on Image Processing*, 10(10):1579–1592, 2001.

[96] J. Lukás and J. Fridrich. Estimation of primary quantization matrix in double compressed jpeg images. *Proc. Digital Forensic Research Workshop*, 2003.

[97] M. T. H. Majumder and A. B. M. Alim Al Islam. A tale of a deep learning approach to image forgery detection. In *2018 5th International Conference on Networking, Systems and Security (NSysS)*, pages 1–9, 2018.

[98] Francesco Marra, Diego Gragnaniello, Luisa Verdoliva, and Giovanni Poggi. A full-image full-resolution end-to-end-trainable cnn framework for image forgery detection. *IEEE Access*, PP:1–1, 07 2020.

[99] Daniel Moreira, Aparna Bharati, Joel Brogan, Allan Pinto, Michael Parowski, Kevin W Bowyer, Patrick J Flynn, Anderson Rocha, and Walter J Scheirer. Image provenance analysis at scale. *IEEE Transactions on Image Processing*, 27(12):6109–6123, 2018.

[100] G. Muzaffer and G. Ulutas. A new deep learning-based method to detection of copy-move forgery in digital images. In *2019 Scientific Meeting on Electrical-Electronics Biomedical Engineering and Computer Science (EBBT)*, pages 1–4, 2019.

[101] Huy Nguyen, Junichi Yamagishi, and I. Echizen. Use of a capsule network to detect fake images and videos, 10 2019.

[102] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models. *CoRR*, abs/2112.10741, 2021.

[103] Sophie J Nightingale, Kimberley A Wade, and Derrick G Watson. Can people identify original and manipulated photos of real-world scenes? *Cognitive research: principles and implications*, 2(1):1–21, 2017.

[104] Nikos Nikolaidis and Ioannis Pitas. Copyright protection of images using robust digital signatures. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 4, pages 2168–2171. IEEE, 1996.

[105] M. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, pages 722–729, 2008.

[106] Hao Ouyang, Tengfei Wang, and Qifeng Chen. Internal video inpainting by implicit long-range propagation. *CoRR*, abs/2108.01912, 2021.

[107] J. Ouyang, Y. Liu, and M. Liao. Copy-move forgery detection based on deep learning. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–5, 2017.

[108] Andrea Passarella. A survey on content-centric technologies for the current internet: CDN and P2P solutions. *Computer Communications*, 35(1):1–32, 2012.

[109] R. Perez, R. Seals, and J. Michalsky. All-weather model for sky luminance distribution—preliminary configuration and validation. *Solar Energy*, 50(3):235–245, 1993.

[110] James Philbin, Relja Arandjelović, and Andrew Zisserman. The Oxford Buildings Dataset. https://www.robots.ox.ac.uk/~vgg/data/oxbuildings/, 2007. [accessed 16-march-2022].

[111] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, jun 1975.

[112] Alessandro Piva. An overview on image forensics. *International Scholarly Research Notices*, 2013, 2013.

[113] A. C. Popescu and H. Farid. Exposing digital forgeries by detecting traces of re-sampling. *IEEE Transactions on Signal Processing*, 53(2):758–767, 2005.

[114] A.C. Popescu and H. Farid. Exposing digital forgeries by detecting duplicated image regions. *Tech. Rep. TR2004-515*, 2004.

[115] A. Preetham, Peter Shirley, and Brian Smits. A practical analytic model for daylight. *Proceedings of ACM SIGGRAPH*, 99:91–100, 01 1999.

[116] Q.-T. Phan, G. Boato, R. Caldelli, I. Amerini. Tracking multiple image sharing on social networks. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2019.

[117] Muhammad Ali Qureshi and Mohamed Deriche. A bibliography of pixel-based blind image forgery detection techniques. *Signal Processing: Image Communication*, 39:46–74, 2015.

[118] N. Hema Rajini. Image forgery identification using convolution neural network. *International Journal of Recent Technology and Engineering*, 8, 2019.

[119] Ravi Ramamoorthi. Modeling illumination variation with spherical harmonics. In *Face Processing: Advanced Modeling Methods*, pages 385–424. 2006.

[120] Y. Rao and J. Ni. A deep learning approach to detection of splicing and copy-move forgeries in images. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2016.

[121] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[122] Sujoy Roy and Qibin Sun. Robust hash for detecting and localizing image tampering. In *2007 IEEE International Conference on Image Processing*, volume 6, pages VI–117. IEEE, 2007.

[123] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[124] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115, 09 2014.

[125] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images, 01 2019.

[126] Gerald Schaefer and Michal Stich. UCID: an uncompressed color image database. In Minerva M. Yeung, Rainer W. Lienhart, and Chung-Sheng Li, editors, *Storage and Retrieval Methods and Applications for Multimedia 2004*, volume 5307, pages 472 – 480. International Society for Optics and Photonics, SPIE, 2003.

[127] Victor Schetinger, Manuel M Oliveira, Roberto da Silva, and Tiago J Carvalho. Humans are easily fooled by digital images. *Computers & Graphics*, 68:142–151, 2017.

[128] Marc Schneider and Shih-Fu Chang. A robust content based digital signature for image authentication. In *Proceedings of 3rd IEEE International Conference on Image Processing*, volume 3, pages 227–230. IEEE, 1996.

[129] Cuihua Shen, Mona Kasra, Wenjing Pan, Grace A Bassett, Yining Malloch, and James F O'Brien. Fake images: The effects of source, intermediary, and digital media literacy on contextual assessment of image credibility online. *New media & society*, 21(2):438–463, 2019.

[130] Dasara Shullani, Marco Fontani, Massimo Iuliani, Omar Al Shaya, and Alessandro Piva. Vision: a video and image dataset for source identification. *EURASIP Journal on Information Security*, 2017(1):15, 2017. TY-JOUR.

[131] Hassan A. Sial, Ramon Baldrich, María Vanrell, and Dimitris Samaras. Light direction and color estimation from single image with deep regression. *CoRR*, abs/2009.08941, 2020.

[132] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 2014.

[133] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015.

[134] Dominic Spohr. Fake news and ideological polarization: Filter bubbles and selective exposure on social media. *Business Information Review*, 34(3):150–160, 2017.

[135] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 06 2015.

[136] R. Thakur and R. Rohilla. Copy-move forgery detection using residuals and convolutional neural network framework: A novel approach. In *2019 2nd International Conference on Power Energy, Environment and Intelligent Control (PEEIC)*, pages 561–564, 2019.

[137] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Transactions on Graphics*, 38:1–12, 07 2019.

[138] Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. *Communications of the ACM*, 62:96–104, 12 2018.

[139] D. Tralic, I. Zupancic, S. Grgic, and M. Grgic. Comofod — new database for copy-move forgery detection. In *Proceedings ELMAR-2013*, pages 49–54, 2013.

[140] Various. Columbia image splicing detection evaluation dataset - list of photographers, 2004. [accessed 16-march-2022].

[141] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[142] Luisa Verdoliva. Media forensics and deepfakes: An overview. *IEEE Journal of Selected Topics in Signal Processing*, PP:1–1, 06 2020.

[143] Nor Bakiah Abd Warif, Ainuddin Wahid Abdul Wahab, Mohd Yamani Idna Idris, Roziana Ramli, Rosli Salleh, Shahaboddin Shamshirband, and Kim-Kwang Raymond Choo. Copy-move forgery detection: Survey, challenges and future directions. *Journal of Network and Computer Applications*, 75:259 – 278, 2016.

[144] Zbigniew Wojna, Vittorio Ferrari, Sergio Guadarrama, Nathan Silberman, Liang-Chieh Chen, Alireza Fathi, and Jasper Uijlings. The devil is in the decoder. In *British Machine Vision Conference (BMVC)*, pages 1–13, 2017.

[145] Y. Wu, W. AbdAlmageed, and P. Natarajan. Mantra-net: Manipulation tracing network for detection and localization of image forgeries with anomalous features. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9535–9544, 2019.

[146] Yue Wu, Wael Abd-Almageed, and Prem Natarajan. Busternet: Detecting copy-move image forgery with source/target localization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 168–184, 2018.

[147] Jianxiong Xiao, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Recognizing scene viewpoint using panoramic place representation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2695–2702, 2012.

[148] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[149] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. Free-form image inpainting with gated convolution. *CoRR*, abs/1806.03589, 2018.

[150] Peipeng Yu, Jianwei Fei, Zhihua Xia, Zhili Zhou, and Jian Weng. Improving generalization by commonality learning in face forgery detection. *IEEE Transactions on Information Forensics and Security*, 17:547–558, 2022.

[151] Ye Yu and William AP Smith. Inverserendernet: Learning single image inverse rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[152] Jinsong Zhang and Jean-François Lalonde. Learning high dynamic range from outdoor panoramas. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4529–4538, 2017.

[153] Jinsong Zhang, Kalyan Sunkavalli, Yannick Hold-Geoffroy, Sunil Hadap, Jonathan Eisenman, and Jean-François Lalonde. All-weather deep outdoor lighting estimation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10150–10158, 2019.

[154] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[155] Qingchen Zhang, Laurence T Yang, Zhikui Chen, and Peng Li. A survey on deep learning for big data. *Information Fusion*, 42:146–157, 2018.

[156] Y. Zhang, Jonathan Goh, Lei Lei Win, and Vrizlynn L. L. Thing. Image region forgery detection: A deep learning approach. In *SG-CRC*, pages 1–11, 2016.

[157] Peng Zhou, Ning Yu, Zuxuan Wu, Larry S. Davis, Abhinav Shrivastava, and Ser-Nam Lim. Deep video inpainting detection. *CoRR*, abs/2101.11080, 2021.

Appendix

SCALE INVARIANT FEATURE TRANSFORM (SIFT)

The SIFT method, introduced in 2004, significantly influenced the field of image processing, especially in applications such as image matching and retrieval. This method extracts a set of key-points from a given image, and for each one of these, it generates a descriptor vector (see Fig. A.1). This algorithm has four major stages: scale-space extrema detection, keypoint localization, orientation assignment, and keypoint descriptors computation.

A.1 Scale-space extrema detection

Key-points are regions in an image with an abrupt change in intensity. It is possible to find keypoints using the second-order derivatives of images and finding zero-crossings on them (Laplacian Filtering). However, this approach is highly noise-sensitive; hence, the image is first smoothed via Gaussian filtering before applying the Laplacian. Laplacian of Gaussian's (LoG) response is highly dependent on the scale (σ) of the Gaussian filter used, as shown on A.2.

Hence, a scale-space volume, obtained by convolving the image with different values for scale (σ) of the Gaussian filter is built to find all the key-points in this 3D volume, with x, y, and σ as independent variables, where x and y are the horizontal and vertical pixel coordinates, respectively.

In addition, LoG can be approximated by a difference of two Gaussian filters with different scales, as shown in Eq. A.1.

$$G(x, y, \sigma) - G(x, y, k\sigma) \approx (k-1)\sigma^2\nabla^2 G \tag{A.1}$$

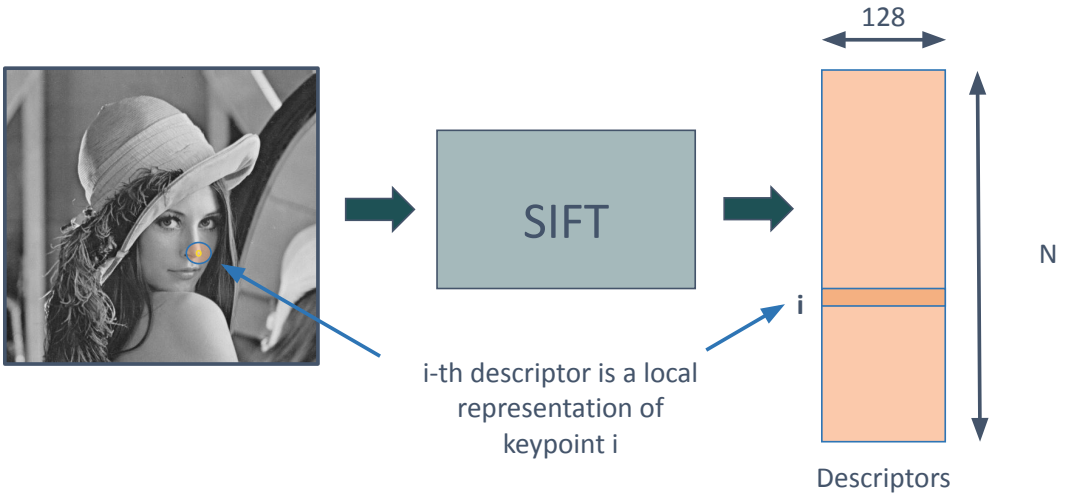


Figure A.1: Descriptors of key points extracted from an image.

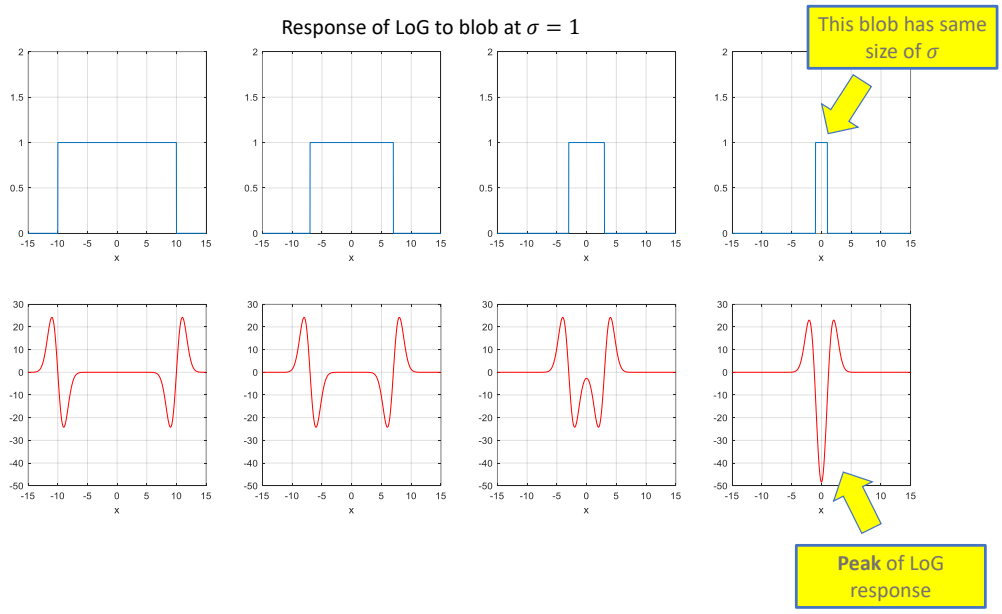


Figure A.2: Laplacian of Gaussian's response to blobs with different scales at $\sigma = 1$.

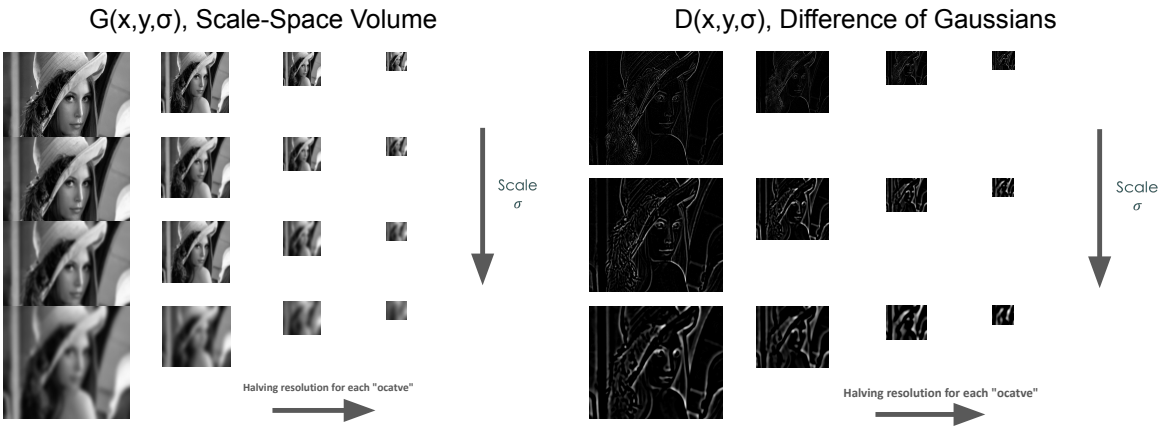


Figure A.3: Left: Creating scale-spaces for each octave. Right: The process of calculating DoGs.

Where $G(x, y, \sigma)$ and $G(x, y, k\sigma)$ are the Gaussian filters with scales of σ and $k\sigma$ respectively, k is a constant factor, and $\sigma^2\nabla^2 G$ is the scale-normalized Laplacian of Gaussian.

For each scale of the Gaussian filter, several octaves are created by halving the resolution in each step. Fig. A.3 shows the process of creating scale-spaces for each octave and calculating differences of Gaussians(DoG).

The maxima and minima are detected in this DoG volume by comparing each pixel to its 26 neighbors in 3×3 regions at the current and adjacent scales. If the value of the current pixel is larger or smaller than all of its neighbors, it is selected as a local extrema location (see A.4.) The factor $(k - 1)$ in eq. A.1 is a constant for all scales and therefore does not influence extrema locations.

A.2 Key-point localization

After identifying candidate key-points as extrema in the DoG space, a further localization step is performed, in which the position of the key-points is refined to

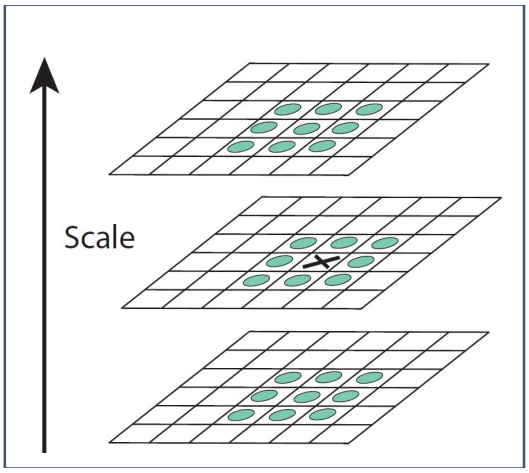


Figure A.4: Finding maxima and minima in DoGs for each pixel compared to its 26 neighbors.

achieve a sub-pixel accuracy.

Let $\mathbf{X}_i = [x_i, y_i, \sigma_i]^T$ be one of the extrema in the DoG volume. A local *continuous* approximation of the DoG volume (which is discrete) is now computed by fitting a polynomial function of order 2 to the neighborhood of $\mathbf{X}_i$. This can be done by using the Taylor series expansion:

$$\tilde{D}(\mathbf{X}) = D(\mathbf{X}_i) + \nabla D\big|_{\mathbf{X}_i}(\mathbf{X} - \mathbf{X}_i) + \frac{1}{2}(\mathbf{X} - \mathbf{X}_i)^T \mathbf{H}(D)\big|_{\mathbf{X}_i}(\mathbf{X} - \mathbf{X}_i) \tag{A.2}$$

where D is used to denote the discrete DoG function. The gradient and the Hessian matrix $\mathbf{H}(D)$ are approximated by using the differences of adjacent points.

The precise key-point location can now be determined at sub-pixel accuracy by finding the minima/maxima of the continuous function $\tilde{D}$. As this function is parabolic, the maxima/minima can be found simply by setting $\nabla \hat{D}(\mathbf{X}) = \mathbf{0}$, yielding the closed form solution:

$$\hat{\mathbf{X}}_i = -\mathbf{H}(D)^{-1} \nabla D + \mathbf{X}_i \tag{A.3}$$

A.3 Orientation assignment

An orientation is assigned to each detected key-point, and the associated descriptor is computed relative to this orientation to make the descriptor invariant to rotations. For a given key-point $\hat{\mathbf{X}}_i = [x_i, y_i, \sigma_i]^T$, we select the plane $G(x, y, \sigma)$ with closest σ to σ_i, then the magnitude $m(x, y)$ and angle $\theta(x, y)$ for every (x, y) in a neighborhood region of $\hat{\mathbf{X}}_i$ are calculated as follows:

$$m(x,y) = \sqrt{(G(x+1,y) - G(x-1,y))^2 + (G(x,y+1) - G(x,y-1))^2} \tag{A.4}$$

$$\theta(x,y) = \tan^{-1} \frac{G(x,y+1) - G(x,y-1)}{G(x+1,y) - G(x-1,y)} \tag{A.5}$$

Using m and θ, an orientation histogram with 36 bins is built to describe the main orientation around the key-point $\hat{x}$. Note that, each entry of the histogram is weighted by the corresponding magnitude m. Finally, a local interpolation around the peak of the histogram is performed and its maximum location is assigned as the orientation of $\hat{\mathbf{x}}_i$. The procedure is shown in fig. A.5.

A.4 Key-point descriptors computation

The SIFT algorithm's last stage consists of assigning a fixed-size descriptor vector to each key-point. This vector is a compact feature that describes the neighborhood of the considered key-point.

For each key-point $\hat{\mathbf{X}}_i$ the corresponding descriptor is built as follows:

1. A 16×16 pixel region around $\hat{\mathbf{X}}_i$ is considered.
2. For each 4×4 sub-block the orientation histogram with 8 possible directions is built.
3. Each histogram is circular-shifted to match the key-point orientation (sec. A.3). In this way, the orientation invariance is achieved.
4. The descriptor is built by the concatenation of the $16 \times 8 = 128$ bins values.

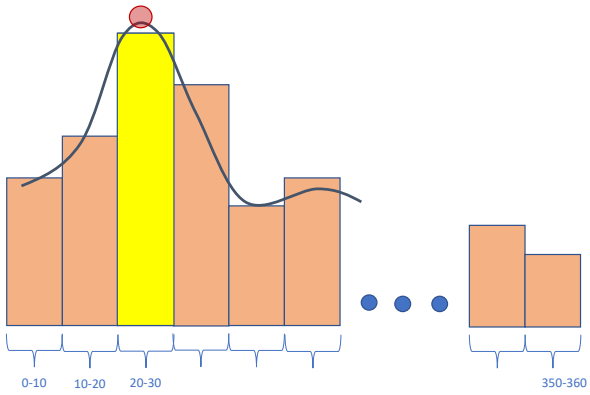


Figure A.5: Orientation histogram computation.