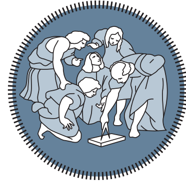

Dipartimento di Elettronica, Informazione e Bioingegneria

Master Degree in Computer Science and Engineering

Convolutional and recurrent neural networks for video tampering detection and localization

Image & Sound Processing Lab (ISPL)

Supervisor:
Prof. Paolo Bestagini

Co-supervisors:
Dott. Luca Bondi
Ing. Nicolò Bonettini

Candidate:
Edoardo Daniele Cannas

Student ID:
879063

Academic Year
2018-2019

Abstract

The development of Internet based communication systems, such as social networks, chat services, etc., has exponentially increased the amount of multimedia objects we can share with other people. The parallel diffusion of easy-to-use editing tools poses issues on the authenticity of the content we are seeing. With the pervasiveness of social media in everyday life, including politics, the malicious use of these edited objects is a threat to the personal and public safety of people. There is therefore an urgent need for tools that automatically determine the authenticity of the content at hand, which are the object of study of the multimedia forensics research field. Goal of this thesis is the development of an efficient methodology for the detection of malicious editing on video signals. Specifically, we resort to deep learning tools, convolutional and recurrent neural networks in particular, to identify and localize **image-based attacks**, consisting in the addition of portions of other source videos in order to alter or hide the content of the original signal.

We address this task using an anomaly detection approach. We propose different types of autoencoders, networks designed to learn a reduced dimensionality representation of their input. Training them to perfectly reconstruct non-altered portions of videos, image-based attacks are localized as regions of the input that are badly reconstructed. By measuring the reconstruction error, we are able to create an heatmap of the possible attacks. We train three different families of autoencoders, both in a recurrent version, where we resort to the convolutional LSTM model, and in a non-recurrent one, where instead we use convolutional neural networks only.

Part of the novelty of this work consists in the way we look at the video signal. Considering it as a volume, we train each family of networks to reconstruct it from different perspectives, by "rotating" the volume of 90° along the frame axes. The use of the convolutional LSTM model, along with this volume rotation procedure, to the best of our knowledge have never been tried in literature, and we show that in some scenarios help the localization of the attack.

To train the autoencoders, we resorted to different forms of loss functions. Among them, we propose a regularization term that empirically enabled us to achieve the best results in our tests.

The results obtained pave the way to future lines of research, such as the use of neural networks that are more suited for handling video

data according to the "volume" paradigm and semantic (3D convolutional neural networks indeed), or the integration of the different perspectives of the rotated volume in a single model. Using semi-supervised approach, or mixing unsupervised and supervised approaches, are also viable options.

Sommario

Lo sviluppo di sistemi di comunicazione basati sull'utilizzo della rete, come i social network, le app di messaggistica istantanea, etc., ha esponenzialmente aumentato la quantità di contenuti multimediali scambiati fra le persone ogni giorno. La diffusione, allo stesso tempo, di strumenti di editing dal facile utilizzo, pone il problema di valutare l'autenticità del contenuto che stiamo osservando. Considerando la pervasività dei social media nella vita di tutti i giorni, e in ambiti sensibili come quello politico, l'uso illecito o malevolo di questi strumenti di editing è una seria minaccia per la sicurezza privata, e pubblica, di un gran numero di persone. Vi è pertanto un bisogno urgente di strumenti in grado di determinare in modo automatico l'autenticità di contenuti multimediali. Tali strumenti sono oggetto di studio del campo di ricerca della forensica multimediale.

Obiettivo di questa tesi è lo sviluppo di una metodologia per l'identificazione di operazioni di manomissione su segnali video. In particolare, utilizziamo strumenti di deep learning, reti neurali convoluzionali e ricorrenti, per identificare e localizzare i cosiddetti *image-based attack*, ovvero le addizioni di porzioni di video esterni usate per alterare o nascondere il contenuto originale del segnale.

Affrontiamo il problema utilizzando un approccio ispirato al campo della *anomaly detection*, proponendo diversi tipi di *autoencoder*, reti progettate per imparare una rappresentazione di dimensionalità ridotta del proprio input. Addestrando queste reti a ricostruire perfettamente porzioni di video non alterate, gli *image-based attack* sono localizzati come regioni del video in input che sono ricostruite sommariamente. Misurando l'errore di ricostruzione, siamo in grado di creare una *heatmap* delle zone attaccate. Addestriamo tre differenti famiglie di *autoencoder*, declinate sia in una versione ricorrente, basata sul *convolutional LSTM model*, sia in una non ricorrente, basata invece solamente sulle reti neurali convoluzionali.

Parte del contributo innovativo di questo lavoro consiste nella modalità con cui osserviamo il segnale video. Considerando il video come un volume, addestriamo ogni famiglia di reti a ricostruirlo da differenti prospettive, "ruotando" il volume di 90° lungo gli assi dell'immagine. L'uso del *convolutional LSTM model*, insieme alla procedura di rotazione del volume, a nostra conoscenza non sono mai stati proposti in letteratura, e mostriamo come in alcuni scenari possano aiutare la localizzazione dell'attacco.

Per addestrare gli *autoencoder*, ricorriamo a diversi tipi di *loss function*. Tra di essi, proponiamo un termine di regolarizzazione che empiricamente ci ha permesso di raggiungere i migliori risultati nei nostri test.

I risultati ottenuti aprono diverse future prospettive di ricerca, come l'uso di reti neurali che siano più adatte all'elaborazione dei segnali video secondo il paradigma e la semantica dei "volumi" (come ad esempio le reti neurali convoluzionali 3D), o l'integrazione delle differenti prospettive del volume ruotato all'interno di un unico modello. Altre opzioni percorribili sono l'uso di approcci semi-supervisionati, o l'utilizzo di approcci misti supervisionati e non supervisionati.

Ringraziamenti

Questa tesi è il risultato di (praticamente) un anno di lavoro svolto presso l'Image and Sound Processing Lab (ISPL).

Non posso quindi che iniziare questo breve paragrafo ringraziando il mio relatore, Paolo, per avermi dato l'opportunità di vedere da vicino il mondo della ricerca scientifica sotto la sua supervisione. Avere come riferimento una figura che stimo moltissimo, per l'essere estremamente preparata e competente, ma allo stesso tempo disponibile, amichevole e premurosa in ogni momento nei confronti dei suoi studenti, è stata probabilmente una delle fortune più grandi che potessi avere in questi anni di percorso universitario.

Come Paolo, e probabilmente anche più, mi hanno seguito durante questi mesi Nicolò e Luca, da cui sono felice di aver potuto "rubare" tanti piccoli segreti e trucchi del mestiere. Vorrei soprattutto scrivere due righe per ringraziare Nicolò, che mi è stato accanto, con una pazienza grandissima, in ogni aspetto di questi ultimi mesi di lavoro senza pause, e nonostante vari "incidenti" di percorso.

Un ringraziamento speciale va poi a tutti i colleghi con cui ho condiviso anche solo una piccola parte di questo incredibile viaggio. In particolar modo, vorrei ringraziare la "squad" di Cremona, con Giorgino, Raf, Mic, Jackie. Un ringraziamento a parte va sicuramente a Vago, super compagno di progetto, e a Molgo, sempre presente compagno di tesi.

Un ringraziamento poi va a tutta la "famiglia allargata" del laboratorio, ISPL e ANTLab insieme, che ogni giorno accoglie con calore o saluta qualche membro, portando avanti con passione e dedizione il difficilissimo mestiere del ricercatore universitario.

Citando il mio caro relatore della tesi triennale, ringrazio poi gli amici, «altrimenti si offendono», e tutta la mia famiglia, senza il cui supporto costante nell'inseguire tutti i miei progetti e desideri, non credo sia necessario dirlo, non avrei potuto essere qua a scrivere queste poche righe.

L'ultimo ringraziamento, il più speciale, va ovviamente a Vi.

Contents

Abstract i

Sommario iii

Ringraziamenti v

List of Figures xiii

Introduction xv

1 Theoretical background 1
- 1.1 Multimedia forensics: a brief introduction 1
- 1.2 Convolutional and Recurrent Neural Networks and autoencoders 3
  - 1.2.1 Machine learning and deep learning basics 5
  - 1.2.2 Convolutional Neural Networks (CNNs) 9
  - 1.2.3 Recurrent Neural Networks (RNNs) 14
  - 1.2.4 Autoencoders 23

2 State of the Art (SOA) 27
- 2.1 Images tampering detection overview 27
- 2.2 Video splicing localization overview 32

3 Problem formulation and proposed methodology 39
- 3.1 Formulation for splicing localization 39
- 3.2 Proposed methodology 42
- 3.3 Video pre-processing 44
- 3.4 Autoencoders architectures 49
- 3.5 Heatmap and mask computation 55

4 Experimental results 59
- 4.1 Dataset description 59
- 4.2 Models training and validation 60
- 4.3 Evaluation metrics 62
- 4.4 Preliminary experiments 62
- 4.5 Main experiments 67
  - 4.5.1 Basic autoencoders 67

4.5.2 Encoding loss autoencoders 70
4.5.3 Regularized autoencoders 71
4.6 Label-unaware results 76

5 Conclusions and future works 77

List of Figures

1.1 *Image lifecycle stages. Videos' lifecycle is almost identical, with the only difference of the repetition in time of the acquisition stage and the different coding format.* 4

1.2 *Typical lossy coding scheme architecture for images (JPEG/JPEG2000 like). The transformation and coding usually are invertible operations, while the quantization introduces distortion and other artifacts that are recognizable as digital footprints. Other footprints might be introduced depending on the codec (for instance, the blocking artifacts of the JPEG codec).* 4

1.3 *Illustration of a deep learning model, taken from [1] page 21. We can see how the different layers construct more and more abstract representations that, from the raw pixel data in input, allow to identify the different objects present in the image.* 6

1.4 *Example of a simple computational graph of a 3-layer neural network. Each element of the graph is a variable, input/output of a layer, and each edge a function implemented by the layer.* 8

1.5 *Graphical representation of what is considered a convolutional layer by the two terminologies. Throughout this thesis, we will use the "complex" one.* 12

1.6 *Example of a feature map produced by the 64th filter of the first layer of the VGG16 network [2]. In this case, the activation function is a ReLU. We can see how some details of the image (like the edges) are more accentuated. This feature map will be then used by the successive layers of the network to produce a more abstract representation of the input.* 12

1.7 *Example of how pooling over the outputs of filters learning the same features with different parameters allow the layer to detect features invariantly with respect to the transformation applied to the input. Figure taken from [1], page 344.* 15

1.8 *Schematization of the architecture of the VGG16 model [2]. The model won the first and the second places in the localisation and classification tracks respectively at the ImageNet Challenge 2014. Figure taken from* https://neurohive.io/en/popular-networks/vgg16/. 15
1.9 *Computational graphs from the folded and unfolded point of view of two generic RNNs used for a classification task where the input and output have the same length. The two design patterns differ for the use of the previous state or of the previous output for the computation of the next state. Therefore, they differ in the function W used for the computation of the next state.* 17
1.10 *1D linear projection of a 100-dimensional hidden state of a RNN, resulting from the (multiple) composition of a non-linear activation function (an hyperbolic tangent in this case) due to the unrolling procedure of the computational graph. On the y axis we have the projection, on the x axis the coordinate of the initial state along a random direction in the 100-dimensional space. The figure shows the behaviour of the hidden state up to 5 composition of the* $\tanh$ *function, so up to 5 steps in the unrolling graph. Taken from [1], page 403.* 18
1.11 *Example of a unfolded computational graph of a RNN implementing the LSTM model. We can see the internal structure composed by the three gates. Picture taken from [3].* 20
1.12 *Use of the convolution operation inside the gates of the convLSTM for the computation of the output $\mathcal{Y}^{(\tau)}$ and state $\mathcal{H}^{(\tau)}$. From this point of view is clearer the nature of the variables as 3D tensors.* 23

2.1 *Pipeline used in [4] for splicing localization.* 31
2.2 *Block diagram of the algorithm proposed in [5].* 32
2.3 *Representation of temporal splicing with frame addition.* 33
2.4 *Representation of image based attack taken from [6]. The forger can perform this operation by repeating a still image in time or by inserting a whole 3D volume, in this case taken from another region of the video.* 34
2.5 *Compression chain assumed by the authors of [7].* 34
2.6 *Pipeline for the video splicing localization followed by the authors in [8].* 36

3.1 *Example of a frame taken from a spliced video and relative mask used to represent the spliced region. In this case the man walking is added through an editing software. The mask belongs to the dataset presented in [8].* 41
3.2 *Pipeline followed by our methodology.* 45

3.3 *Spatio-temporal and spatial cropping on the normalized video signal. We can see how the sequence "patch" extracted is smaller in the height, width and temporal dimension, as an "extension" in time of a normal frame patch. The frames are taken from a video of the dataset presented in [8].* . . . 46
3.4 *Example of volume rotation along the y axis.* . . . 47
3.5 *Example of patches extraction with spatio-temporal cropping from the volume rotated by 90° along the y axis.* . . . 48
3.6 *The spatio-temporal autoencoder proposed in [9] (on the left) and our basic recurrent architecture (on the right). On the right of each layer we can see the dimensionality of the tensor in output.* . . . 50
3.7 *The convolutional autoencoder proposed in [10]. Please notice that also in this case no pooling is used, and that the activation of each layer is linear.* . . . 51
3.8 *In order to compute the loss between the hidden representations, we first make the autoencoder reconstruct the input patch, and then reconstruct the "reconstructed" patch another time. We then take the hidden representation of the two patches and compute the MSE between them. In this figure we use as an example the reconstruction of a sequence patch by our recurrent autoencoder.* . . . 53
3.9 *Heatmap computation procedure executed on sequence patches. The process is almost identical for frame patches, changing only the size of the mean filter.* . . . 56
3.10 *Thresholding of the heatmap in order to create different mask estimates. Each of them are then confronted with the reference mask to compute ROC curves as explained in Chapter 4.* . . . 57

4.1 *Illustration of the splicing process executed by the authors in [8]. In the top row we can see frames of the original video, in the middle the green-screen clip used for forging, and in the bottom the final spliced video. Image taken from [8].* . . . 60
4.2 *Illustration of the division of samples in the feature space into true positives and false positives according to a binary classification algorithm. Samples inside the circle are classified as positives, those outside as negatives.* . . . 63
4.3 *AUC values for the ROC curves of the first four prototypes.* 65
4.4 *AUC values for the ROC curves of the prototypes with batch normalization.* . . . 65
4.5 *AUC values for the ROC curves of the prototypes trained with volume rotation.* . . . 66

4.6 *AUC values for the ROC curves of the basic autoencoders, both recurrent and not.* 68

4.7 *ROC curves scored by the recurrent autoencoder working on the x axis rotated volume for the video number 7 (left) and the video number 6 (right).* 68

4.8 *Forged version of video number 8 and relative mask representing the spliced region. As we can see, the texture of the spliced area (a tree among other trees) is almost identical.* 69

4.9 *AUC values for the ROC curves scored by the encoding loss autoencoders trained with $\alpha = 0.25$ and $\beta = 0.75$.* . . . 70

4.10 *AUC values for the ROC curves scored by the encoding loss autoencoders trained with $\alpha = 0.5$ and $\beta = 0.5$.* 70

4.11 *AUC values for the ROC curves scored by the encoding loss autoencoders trained with $\alpha = 0.75$ and $\beta = 0.25$.* . . . 71

4.12 *Some of the ROC curves scored by the recurrent autoencoder working with the y axis rotated volume and trained with $\alpha = 0.25$ and $\beta = 0.75$. On the top row we have: on the left, the ROC curve for the video number 8; on the right the one scored for video number 6. On the bottom row: on the left the ROC scored for the video number 1; on the right for video number 7.* 72

4.13 *AUC values for the ROC curves scored by the regularized autoencoders trained with $\alpha = 0.25$ and $\beta = 0.75$.* 73

4.14 *AUC values for the ROC curves scored by the regularized autoencoders trained with $\alpha = 0.5$ and $\beta = 0.5$.* 74

4.15 *AUC values for the ROC curves scored by the regularized autoencoders trained with $\alpha = 0.75$ and $\beta = 0.25$.* 74

4.16 *ROC curves for the video number 8. On the top row we can see: on the left, the ROC scored by the recurrent autoencoder working with no volume rotation; on the right, the ROC obtained by working with the x axis rotated volume. On the bottom, we can see: on the left, the ROC scored by the non-recurrent autoencoder working on the y axis rotated volume; on the right, the recurrent autoencoder working with the x axis rotated volume. The values of α and β are equal to 0.25 and 0.75 for all networks.* . . . 75

4.17 *AUC mean values for a selected set of networks. We took networks working only with volumes rotated along the x axis.* 76

5.1 *Example of a possible procedure of volume merging. The three colored small volumes represent sequence patches extracted from the rotated volume separately. If instead they are cropped from the same region of interest, as it happens in the bottom left corner, we can construct an overall representation that is a weighted perspective of the three different angles.* 80

Introduction

In the last few years, the development and diffusion of low-cost computational devices such as smartphones, tablets, etc., together with the spreading and improvement of Internet connections all around the world, has given a further boost to the already impressive development of the World Wide Web as the most preferred tool for connection and communication among people.

The combination of these factors, along with the rise of social media and chat services like Facebook, Instagram, Twitter, WhatsApp, Telegram, etc., has guided the type of virtual interactions between people towards forms that are usually integrated by, if not completely centered on, some kind of multimedia content sharing. Considering that the type of content accessible by web sites nowadays is almost always integrated with images, videos or audio files, and that this trend couples together with the presence of peer-to-peer file sharing systems of different nature from the early '90s like Napster, eDonkey, etc., the dimension of multimedia content shared on the Internet has now reached really impressive numbers[1]. While on one hand, we can think of this phenomenon as a direct manifestation of the democratic nature of the Web, where information can be shared and obtained freely by anybody, on the other this free exchange of data poses severe issues on whether we can trust the authenticity of the content we are seeing.

In 2016, the defense minister of Pakistan Khawaja Muhammad Asif, reacting to a false article published on a website reporting that Pakistan had been threatened with nuclear weapons, wrote a menacing Twitter post directed to Israel [12]. This is a clear example of not verified content available on the Web (a so called "fake news"), that could lead to a potential terrible outcome since both countries have a nuclear arsenal. This example clarifies the urgent need for tools that are able to automatically verify the authenticity of the uploaded material, being a fake news or a forged image, video or audio file. This is one of the main object of study of the **multimedia forensics community**.

Multimedia forensics is a discipline mainly concerned in assessing the

[1] The statistics offered by this website [11] give an overall idea on the dimension of data exchanged through social networks and chat services alone. The figures are quite impressive, and give just a little picture of what Big Data is really about. However, this theme is of no concern for this thesis.

trustworthiness of a multimedia content, that is a digital (multidimensional) signal like an image, audio, or video file. In fact, the vast offer of dedicated software, from entry-level to professional tools, for editing almost any type of multimedia file, has made in the recent years digital objects authentication and validation significantly harder. Digital photographs or videos, for instance, which are largely used to provide objective evidence in several applications ranging from surveillance to legal and medical services, can no longer be trusted since their integrity might be compromised [13]. Besides, due to the wide diffusion and pervasiveness of Internet's social media in today's society, cases such those of the minister Khawaja suggest that the possible dangers resulting from a forged content can arrive from areas completely different from the legal or medical ones. Forged photos or videos can be used for creating political tensions, with unthinkable consequences, and for this reason they are starting to be considered as destabilising factors even by governmental agencies [14].

Anyway, forged multimedia content is somehow recognizable, not only by some skilled professional looking at its actual content, but also in an automatic way from a signal processing perspective. Indeed, the main idea behind multimedia forensics is that most of the operations that alter the authenticity of a signal are not reversible. Depending on the specific nature of the signal itself and of the operation executed, each step of the forging process alters in a more or less detectable way the underlying features of the multimedia object, leaving some kind of **traces** or **footprints** behind. Therefore, by simply looking for the characteristic footprints left by an editing operation, it is possible to reconstruct at least part of the processing chain that the object has undergone.

Great parts of the research efforts of the multimedia forensics community has been directed towards the analysis of still images. For giving an idea of the results obtained in this field, nowadays forensics techniques allow to determine many steps of the processing history of an image [15] [16] [17], whether it has been altered through cut and paste operations from other images, or if part of it has been artificially modified and how. However, in spite of the rich literature available on images, forensics analysis of **video signals** presents many unexplored area of research [18].

This is partly due to the specific nature of video signals which are difficult to manage for their dimensionality and size, and partly due to the wider range of possible forgeries that can be executed on the original content in order to alter it. Indeed, considering videos simply as sequences of frames, all alterations that can be performed on still images can be executed singularly on each video frame too. Moreover, the forger might think of working directly in the temporal dimension, erasing or hiding details from the scene. The spread of powerful yet easy-to-use video editing tools, such as Adobe Premier, FaceSwap or Adobe After Effect CC, enlarges the set of options available for the forger.

All these different, yet concurring, aspects make the work of forensics

researchers in detecting some characteristic footprints in the content under analysis more difficult when dealing with video signals. As we have already said, "footprints" depend on the type of signal and on the type of forgery executed, therefore some kind of knowledge of the attack from the researcher which has to devise a method for detecting it is needed.

In the recent years, with computational devices increasingly getting cheaper, techniques coming from the **deep learning** field have regained popularity in different research applications. The main assumption behind these techniques is that if we have a reasonable end-to-end model that formulates our problem, together with enough data to train it, so to tune its parameters, we are close to solve it. The multimedia forensics community has started recently to approach these methods, also called **data-driven algorithms**, since they show the appealing property of learning by themselves, directly from the data, the features characterizing the footprints left by a forging process [19] [20] [21] [22].

Goal of this thesis is to try to apply deep learning algorithms in order to detect whether and where a video signal has been tampered. In particular, our work has been focused in using **convolutional** and **recurrent neural networks (CNNs and RNNs)**, in order to localize specific regions of video signals forged by the addition of material coming from another source object. This is the so called **splicing** or **image-based** attack. CNNs are tools developed in their most known and diffused form starting from the '80s. They have been applied with success for different tasks, included imaging and vision [23] [24], and recently they have started to be applied with good results in the image and video forensics fields too [19] [25] [26]. RNNs instead are tools explicitly designed to capture patterns and dependencies in sequences of data samples. Their most famous model, the **Long Short Term Memory (LSTM)**, has been proposed originally in 1997 [27], and in the last years it has been applied widely in almost every task involving time-series data like speech recognition, handwriting recognition, polyphonic music modeling, etc [28].

In our work, while still utilizing these tools belonging to the area of deep learning, we tried to cope with one of the main difficulties encountered by multimedia forensics researchers: the scarcity of datasets of forged content. In order for deep learning algorithms to work in fact, well organized datasets are needed. Referring ourselves to the specific context of the multimedia forensics, this often translates in having a dataset containing samples of both forged and "clean" objects in a sufficient number in order to make the algorithm "learn" to discriminate between the two classes. However, this is seldom the case, mainly due to the lack of forged data.

We tried therefore to address this problem taking inspiration by the work done in the **video anomaly detection** field. Anomaly detection is a subdiscipline of unsupervised learning whose main goal is, given a set of training samples containing no "anomalies", to learn a representation

of data that captures the behaviour of "normal" samples in some kind of feature space. Anomalies are therefore defined as samples deviating from this normal behaviour, with the deviation measured in different ways, for instance like the approximation error of the anomaly projection in a vector space, or like a posterior probability of seeing the anomaly given a distribution modeling the normal samples. We can see that considering the "normal" behaviour of samples that of clean content, and "anomalous" that of forged content, we can cast the forensic problem of detecting if a video has been tampered with as an anomaly detection one.

Anomaly detection for video signals presents the same problems related to the high dimensionality structure of videos faced by multimedia forensics. Employing deep learning methods has offered therefore the same appeal to the researchers of this area, and in fact in the last years many contributions to the literature using deep learning methods have appeared [29].

In our case, we resorted to the idea of using CNNs and RNNs as **autoencoders**, tools able to reconstruct the signal gave as input to them. "Training" these algorithms to perfectly reconstruct forged-free videos, we show that they are able to detect forging simply by "badly" reconstructing the video in forged areas. Confronting the original tampered video with the reconstructed one, measuring the deviation of the reconstruction through a measure of error, areas with high reconstruction error are highly probable forged for the addition of material coming from another object. Interested in how the use of a model that takes into account sequentiality along a dimension of the input data might improve the detection performances, we have devised and compared two different versions of autoencoders: one, based on CNNs, which is not recurrent, and therefore works on each video frame separately; one based on the LSTM model, which instead works considering sequences of frames extracted from the videos.

Other authors have approached the video forensics problem in a way similar to ours: we cite the work done for image forensics in [5] and in video forensics in [8]. This last paper particularly has been chosen as a starting point for this thesis: in first place, because the authors have used RNNs, and secondly because they have published the dataset on which they have worked, which was used for our tests too. However, our work differs from theirs for two main reasons.

First, in both works they resorted to deep learning models starting from features extracted from the images/frames [2], while we have directly worked with the pixel data of the videos. Therefore, our models differ significantly, and in fact the ones which can be considered the closest to our work are those presented by [29] as **reconstruction** and **predictive models**. Specifically, we cite works using models that take into account explicitly the sequentiality given by the temporal dimension in video

[2] In particular, they used **co-occurrences** extracted from the residuals of a third-order high-pass filter for each image/frame.

data such as [30], using LSTM networks like for instance [31], and more specifically those using **convolutional LSTM** networks i.e. [9]. This last paper employs the same recurrent neural network architecture used by us, convolutional LSTM networks indeed, which was first proposed by Shi et al. in [32] for precipitation forecasting problems. To the best of our knowledge however, this kind of network has never been applied to the video forensics task.

Secondly, another novel aspect of our work is that we analyze each video from different perspectives. A video signal can be considered as a simple sampling over a 3D lattice of a function $s(x, y, t)$ representing the evolution of **luminance** along the time axis in a spatial location defined by the cartesian coordinates x, y. From another point of view, we can consider all the samples corresponding to a time instant t as an image or frame of the video. We have already said that RNNs are models explicitly designed to exploit patterns in sequences of data samples, but in video processing areas they have usually been used in order to capture information along the time axis only. In order to obtain a representation of clean content more and more robust, we have trained different models of our autoencoders, both the recurrent and non-recurrent version, changing the axis along which we wanted to focus the attention of the networks. This operation can be viewed, if we consider the whole video data as a 3D volume, as a "rotation" of the volume that allows the autoencoders to represent the same information from a different point of view, while preserving the semantic of the data. We will show that this approach helps our autoencoders in detecting some of the malicious manipulations that are not detectable when considering videos as only a sequence of frames along the temporal dimension.

The thesis is organized as follows.

In Chapter 1 we provide a general background on the deep learning tools used and on the video forensics problem in general.

In Chapter 2 we present some techniques and state of the art methods for tampering detection and localization in both image and video forensics.

In Chapter 3, we describe the proposed methodology along with the models and architectures developed, including a general formulation of the problem of tampering detection and localization and a description of the volume rotation operation.

In Chapter 4, we illustrate the dataset used and the experiments performed with some considerations on the results obtained.

Finally, in Chapter 5 we draw the final conclusions on our work and we outline some possible future lines of research in this same topic.

1
Theoretical background

This chapter aims at providing to the reader who is not acquainted with the matter of this thesis the necessary theoretical background in order to understand its results.
We will first illustrate the main concepts behind **multimedia forensics** techniques, remarking the main goals of this discipline and the differences between **active** and **passive** methods for achieving them, and then in a separate section introduce the main **deep learning** tools used in our work. In particular, we will focus first on some general concepts of machine learning, and then on **convolutional and recurrent neural networks** and **autoencoders**.

1.1 Multimedia forensics: a brief introduction

In the previous pages we have analyzed how the evolution of the Internet-based media communication systems, along with the diffusion of devices and tools allowing an easy manipulation and editing of multimedia content, has made the need for multimedia forensics techniques more and more pressing for different types of applications.

To assess the content authenticity and trustworthiness, we can rely on two main categories of methods: **active** and **passive**. While the first are based on the idea of a **trustworthy** acquisition device, meaning that at acquisition time a particular signature or **watermark** is computed such that the authenticity of the multimedia object can be assessed by simply looking at it, the second comprehends all the techniques that blindly analyzing the content at disposal try to determine its genuine-

ness. The former methods are seldom adopted, since they require a set of implementation duties all on the acquisition device producer. The latter techniques instead constitute the main research field of **multimedia forensics** experts.

The core idea of passive methods is the identification of forging by searching for **characteristics footprints** left during each of the stages of a multimedia object lifecycle. Every digital signal lifecycle[1] in fact can be divided into three main steps: **acquisition, coding** and **editing**, each of them characterized by some non-linear operations that generate some more or less recognizable artifacts on the resulting object.

For images and videos, acquisition consists in the generation of a digital signal through the focusing of the light coming from a natural scene into the camera **active-pixel sensor (APS)**, an integrated circuit made by the combination of thousands of small units called **pixels**, which in turn consist of a photodetector and an active amplifier. Light generally is focused by means of **lenses** on pixels, but not directly: in fact, a thin film is interposed between the lenses and the pixels so that only certain components of light are actually captured. This film is the so called **Color Filter Array (CFA)**, which makes each pixel gather only one particular colour (green, red or blue). The overall image or video frame is then obtained by **interpolating** the different components captured by the pixels of the APS, obtaining the digital colored image. This is the so called **demosaicing process**, which is usually followed by additionally elaboration and processing like white balancing, color processing, etc. Typical footprints that we can found left at this stage are related to **lens characteristics** (like artifacts left by the phenomenon of the **chromatic aberration**), APS characteristics (like the **Photoresponse Non Uniformity noise, PRNU**), and the pattern used in the **demosaicing** process, and are common to image and video signals.

After acquisition the image or video is saved into the memory of the digital camera, and to accomplish this task usually they are **coded** or **compressed** to save space. The compression schemes used are in general **lossy**, and one of the most common standards applied for images is the **JPEG** (see figure 1.2 for an example of processing chain of a lossy compression codec). Typical artifacts left by JPEG for instance are related to the **blocking** and **quantization** operations performed by the codec, and can be found not only in images but in video frames too, since many of the most common video compression codecs use operations similar to those of JPEG in part of their processing chain. However, even though the acquisition of a video signal can be considered as a repetition in time of the image acquisition stage, video codecs are not a mere application of image standards for compression frame by frame: they usually exploits **temporal redundancy** in order to reduce the amount of information to encode. Therefore, along with JPEG-like traces, we can found artifacts connected to the **spatial and temporal prediction techniques** used

[1] The scheme in figure 1.1 resumes the typical lifecycle of a digital image.

to further reduce the information encoded. Typical video formats of this type are the **MPEG-x, H.26-X or 3GP** for instance.

Finally, we arrive to the final stage of a multimedia object lifecycle, the **editing**. This step comprehends any kind of signal processing manipulation that can enhance or modify its content. For images, we range from filtering, blurring, sharpening, etc., to geometric transformations (rotations, scaling, etc.), cloning (copying and pasting portion of the image on the image itself) to, finally, **splicing**, which consists in the composition of an image using parts taken from other images. Obviously, each operation executable on single images can be executed on single video frames too, therefore comprehending the splicing altogether a new range of attacks that exploit the temporal dimension in order to cover or alter the video's content. Footprints left in editing are extremely various and strongly depends on the type of operation performed: in our work we focused on **splicing**, most specifically on splicing **localization**, and we refer the reader to the following chapter focused on the SOA methods for tampering detection and localization for images and videos.

Each of the traces described so far can be used for a wide range of tasks. In the literature, methods that look for acquisition stage footprints are commonly used for **device identification**, while coding footprints for determining if a **double compression** has been executed on an image or video, and both of them along with some other specific footprints are used for editing detection [2]. In general, by detecting the footprints we can reconstruct the lifecycle of the object under our analysis, and therefore expose any possible forging that it might had undergone. A detailed overview regarding the different footprints and the tasks they are used for by multimedia forensics researchers is out of the scope of this thesis, but can be found in [17] for images and in [18] for videos.

While the presence of these artifacts is related to the non-linear operation characterizing each lifecycle stage of the multimedia content, is a duty of the researcher the development of methods for detecting them, and of course the development of algorithms that use these footprints to accomplish the task at hand, being a device identification or a tampering detection. This is why in the recent years, **deep learning** techniques have gained a great popularity in the multimedia forensics community.

1.2 Convolutional and Recurrent Neural Networks and autoencoders

Before analyzing the SOA for image and video tampering detection, we dedicate some time to the description of the deep learning tools used in our work. As we stated previously, the use of deep learning techniques has gained a great appeal in the multimedia forensics community in the recent

[2]For instance, double compression detection is considered as a sign related to the presence of an out-camera processing.

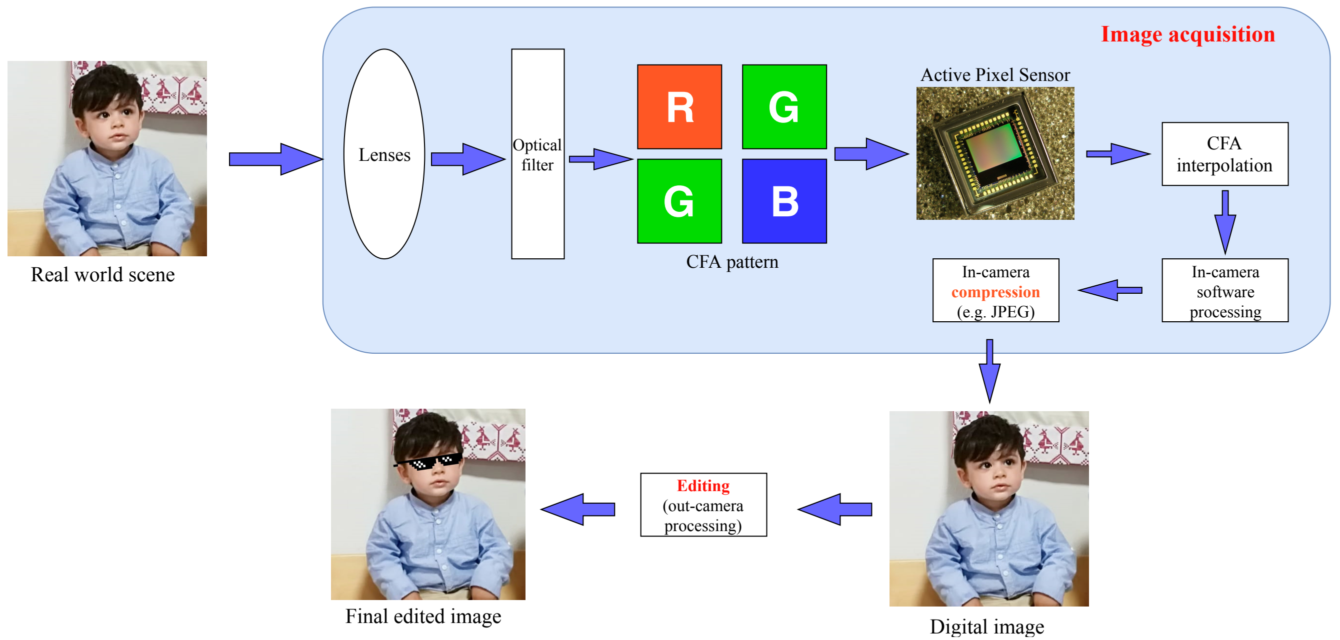

Figure 1.1: *Image lifecycle stages. Videos' lifecycle is almost identical, with the only difference of the repetition in time of the acquisition stage and the different coding format.*

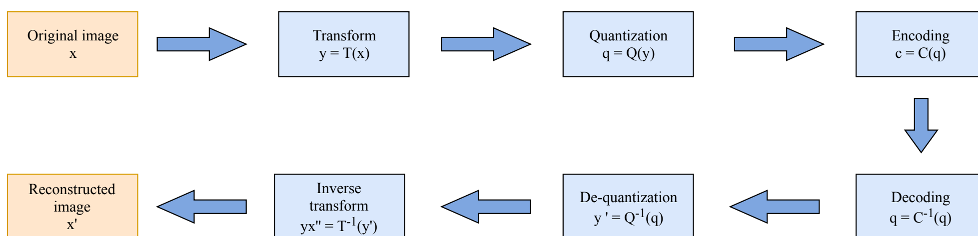

Figure 1.2: *Typical lossy coding scheme architecture for images (JPEG/JPEG2000 like). The transformation and coding usually are invertible operations, while the quantization introduces distortion and other artifacts that are recognizable as digital footprints. Other footprints might be introduced depending on the codec (for instance, the blocking artifacts of the JPEG codec).*

years. Deep learning can be considered as a subfield of the **machine learning** research area, and before the recent rise of deep learning tools machine learning techniques have been consistently used by multimedia forensics researchers too. But what do we mean by machine learning?

1.2.1 Machine learning and deep learning basics

Following the classic definition by [33], a machine learning program is defined as a program which can improve its performance accomplishing a certain task T with respect to some performance measure P, using experience E. The task T is usually described in terms of how the program should process a certain **input**, which is composed as a set of **features** quantitatively measured from the object or event we want the program to elaborate, in order to produce a desired **output**. Features are almost always organized in vectors $x_i \in \mathbb{R}^n$, with each vector x_i being a new set of n features fed to the program.

In the context of multimedia forensics, the task at hand depends on the goal of the forensics analysis, whether it is a device identification, a splicing detection, etc.; the input instead is made out of the **footprints** extracted by the researcher from the multimedia object. We can see therefore how machine learning techniques helped the researchers, since they allowed, given the footprints extracted from the object under analysis, to automatically derive an algorithm capable of accomplishing the task. However, these techniques show a major drawback: their performances are heavily influenced by the way data are *presented* to the program, meaning that depend on the features extracted from the raw objects. In our case, this means that these algorithms need to be provided with meaningful footprints, leaving to the researcher the major duty of the forensics analysis.

Manually designing features is a time-consuming work that can keep busy a research community for years. Therefore, it would be greatly desirable to have some mechanism through which we can extract the footprints in an automatic way too. In other words, it would be desirable to let the program learn directly from the data a useful representation in order to accomplish the task. This is the main object of research of **representation learning**, not only let the program learn how to process the representation, but allow it to learn the best representation, so the best set of features, to produce the desired output.

Deep learning can be considered as a subset of the machine learning research field that incorporates the ideas from representation learning. Core of this set of techniques is the use of **neural networks**, of which the most common and used versions are the **Multi Layer Perceptron (MLP)** and the **Convolutional Neural Networks (CNN)**[3].

[3] We will explain in details this second type of networks along with the other tools used in this work, while for a deeper discussion of the MLP we address the reader to the classical book on deep learning by Goodfellow et al [1].

Without going too much deep into details, we can say that both typologies of networks base their functioning on the construction of a good representation of the input data through the stacking of simple modules, called **layers**. Each layer is a simple mathematical function mapping input to output values, where the output is a new representation of the data that is used by the successive layer. Concatenating this outputs together substantially creates a hierarchical list of concepts, with each concept defined starting from simpler ones (the outputs of the previous layers), allowing to construct representations that are more and more abstract with respect to the starting input data (see figure 1.3 for an example). The **depth** of a model therefore refers to the number of layers concatenated together. But how can these networks learn the representation and the task?

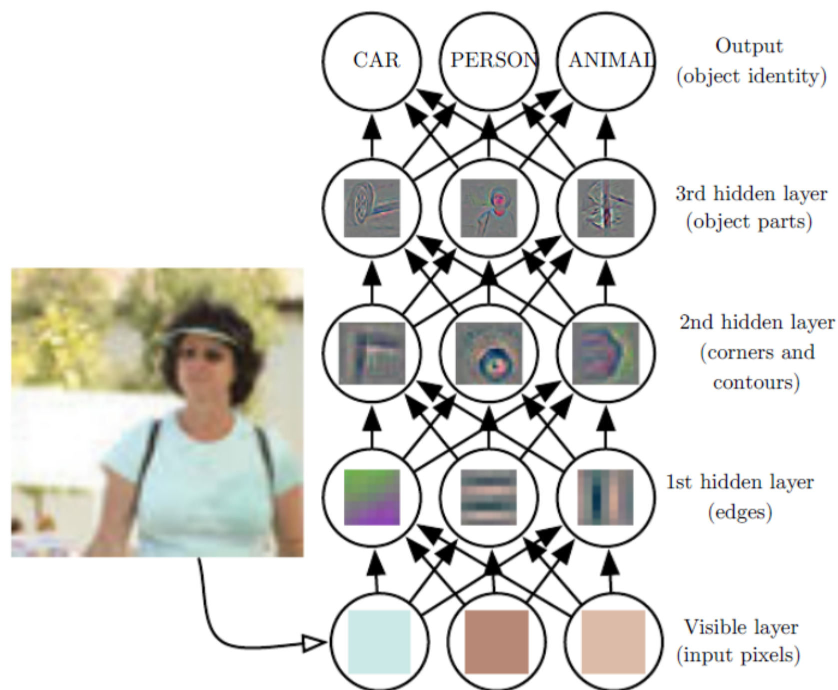

Figure 1.3: *Illustration of a deep learning model, taken from [1] page 21. We can see how the different layers construct more and more abstract representations that, from the raw pixel data in input, allow to identify the different objects present in the image.*

They learn the representation while learning the task. All machine learning, including deep learning and multimedia forensics, tasks can be roughly divided into some general categories. The most common are the simplest ones like **classification** of the input into some category, or predicting a continuous numerical value given a input (**regression**). However, we can develop even some more elaborated computations like **image description**, **machine language translation**, or some very specific jobs like **probability density estimation** or **anomaly detection**, where the program is asked to model a particular category of events/input and flag those inputs which are unusual with respect to the category modeled.

For all of these the role played by the performance measure P is crucial in determining how the model learns both the task and the representation. The performance measure is a quantitative measure of the overall behaviour of the model, and is the main, and only, means through which the model itself can learn the task.

This statement is clearer if we analyze the process through which these networks are able to learn, which is called **training**. Training is generally an iterative procedure, consisting in the processing of some inputs by the algorithm (the **training set**) and the update of the model's **weights** by the maximization/minimization of the performance measure used to evaluate it after it has processed the inputs.

All machine learning algorithms indeed are written in a parametric form, where the parameters are usually called **weights**. At the beginning of training these are set usually randomly or with some initialization procedure, and during training are updated accordingly to the form of the model and the task (and loss) being taught to the algorithm. In the case of classification or regression, the performance measure is defined as a **loss function**, a function that quantifies the value of an hypothetical loss when an input is misprocessed compared to a desired output, and the values of the weights are found by computing the **gradient** of the loss function with respect to the weights of the model and posing it equal to zero. In formulas, we can express the output of the algorithm as

$$y = f(\mathbf{x}; \mathbf{w}), \tag{1.1}$$

with $\mathbf{x}$ being the input and $\mathbf{w}$ the parameters of the model, and our loss function $J(\mathbf{y}; \hat{\mathbf{y}}; \mathbf{w})$ as a function of the output, desired output $\hat{\mathbf{y}}$ and of the weights. Example of loss functions are the **mean squared error (MSE)** for regression tasks, for instance, or the **cross entropy** for classification. The equation for finding the best set of parameters $\mathbf{w}^*$ given $J(\mathbf{y}; \hat{\mathbf{y}}; \mathbf{w})$ is then equal to

$$\nabla_{\mathbf{w}} J(\mathbf{w}^*) = \frac{\partial J(\mathbf{w}^*)}{\partial \mathbf{w}} = 0. \tag{1.2}$$

where we have omitted the other variables of the function.

Some algorithms permit a closed form solution for equation 1.2 that leads to the best performances; others need **iterative** algorithms that update the weights sequentially during training. The latter is the case of neural networks, where the most common used algorithm is the **stochastic gradient descent (SGD)** and its variants (like Adam [34]). The value of the weights is found iteratively using the following formula, where k indicates the step of the algorithm and α is a convergence parameter

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha \nabla_{\mathbf{w}} J(\mathbf{w}), \tag{1.3}$$

until some convergence criteria is met.

While this method might seem really simple, it has some numerical implementation details that are critical. Neural networks can implement some really complicated **computational graphs**, due to the stacking of the different layers. On one hand, this stacking gives them great flexibility in learning very difficult mappings from input to output, but on the other makes the computation of the gradients not an easy task.

The reason lies on the fact that each layer has a parametric form of its own. Sequentially stacking functions makes more difficult to understand which parameter of which function influenced the final result, and therefore when training the model we should update each weight of each layer accordingly to how it determined the final output. Mathematically, this problem translates in computing the gradient of the final output function 1.1, which derives from the composition of the function of the different layers. We solve this problem using the **chain rule of calculus**. A simple example is given in figure 1.4, where the final output function can be expressed as

$$t = f_3(z) = f_3(f_2(y)) = f_3(f_2(f_1(x))), \tag{1.4}$$

and the update of the weights to be inserted as the second term of equation 1.3 can be computed considering pair of nodes of the computational graph as

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i} \cdot \frac{\partial x_i}{\partial w_{ij}}, \tag{1.5}$$

with y being the output, x being the input and w the weights of the layer between the nodes i, j of the computational graph.

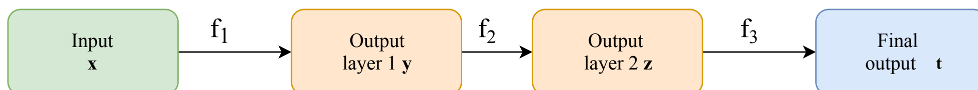

Figure 1.4: *Example of a simple computational graph of a 3-layer neural network. Each element of the graph is a variable, input/output of a layer, and each edge a function implemented by the layer.*

The algorithm of computation of the gradients from the loss for each weight of the layers of the network is referred to as **backpropagation (backprop)**.

Training neural networks therefore can be thought as a two phase "sweep" iterative procedure:

1. a **forward** computation of the output of the network given the input, comprehending the performance measure altogether;

2. a **backpropagation** of the gradients computed from the loss in order to update the model's weights;

3. repeat until the model reaches some desired performance.

To this end, it is recommendable to evaluate the performance not only on the data used for training, but on some unseen examples in order to have a better and clearer idea of the generalization capabilities of the model. For this reason, usually the data available for training a neural network is split into three different sets:

- a **training set**, used for training indeed and for updating the model's weights;

- a **validation set**, used for stopping the training;

- a **test set**, used to evaluate the general performance of the model at the end of training.

The motivation behind this approach is the fact that training loss is generally an optimistically biased measure of the performance of the model. Validation loss instead can be considered as a more unbiased measure, and can be used for determine different characteristics of the models like the number of layers, type of function implemented etc., that are usually called **hyperparameters** (parameters *not* determined with training). This is the so called **model selection** procedure, which is used to compare different design patterns and architectures to find the most appropriate for the task.

With this final remark we close the introduction on the deep learning basics of the section. In the next pages we will explain in more details the functioning of **CNNs** and **recurrent neural networks (RNNs)**. We will then close the chapter illustrating the «quintessential example of a representation learning algorithm»[1], the **autoencoder**.

1.2.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks are a specialized kind of neural networks for processing data with a grid-like topology. Examples of this kind of input are digital signals, since they are the result of a **sampling** process, and we can think of the sampling process as the conversion of a continuous signal into a sequence of discrete values (samples) gathered by an imaginary regularly spaced grid lying in the physical dimensions where the signal lives. Obviously images, which can be thought of a

2-dimensional grid of pixels, or any time-series data like audio files or videos, the latter which can be considered as a sampling over a 3D lattice of a function $s(x, y, t)$[4], are examples of this kind of data.

As we said before, CNNs are constituted by the stacking of several layers (see figure 1.8 for an example), these being simple mathematical mappings, and in particular CNNs take their name by the use of an operation in each of their layers, which is indeed the convolution. Convolution is a mathematical operation defined between two functions of the same variables. Taking two functions $x(t)$ and $w(t) \in \mathbb{R}$ of a single real variable, convolution $c(t)$ is defined as:

$$c(t) = \int x(a)w(t-a)da, \tag{1.6}$$

and often indicated with an asterisk as $c(t) = (x * w)(t)$. When we deal with discrete functions, meaning that our independent variable (in this case t) does not assume values in a continuous range but only fixed integer ones (in other words, it has been sampled like happens for time in audio signals), we can define the **discrete convolution** as

$$c(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{1.7}$$

With functions of more than one discrete variable we can still define discrete convolution by simply adding a summation taking into account each one of their other variables. This is what usually happens in deep learning: the first term of the convolution, which is referred to as **input**, is a multidimensional array (a **tensor**) of data, like an **image** I, and the second term, which is referred to as the **kernel**, is a multidimensional array of parameters.

Arrays are a common implementation in computer science for storing the values of a sampled signal. Monodimensional arrays are used for audio signals, bi-dimensional arrays for images, three-dimensional arrays (tensors) for videos, etc. Obviously, these data structures are not infinite in size. This implies that our terms are not infinite functions, and therefore we can implement our convolution as a finite summation over the array/tensor elements. As an example, we can write the convolution C between a 2-dimensional input I and kernel K as

$$C(i,j) = (I * K)(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n), \tag{1.8}$$

[4]As we have stated previously, recall that the function $s(x, y, t)$ represents the evolution of **luminance** along the time axis in a spatial location defined by the cartesian coordinates x, y. Video signals in reality are 4-dimensional tensors, with the last dimension representing the different color channels relative to the components of light filtered by the CFA. Our convention of considering the video signal as the evolution of the luminance only will be explained in Chapter 3.

where we have exploited the **commutative** property of the convolution to write it in a form called without **flipped kernel**, which is more straightforward to implement in a machine learning library[5].

Convolution alone is rarely used in deep learning. Very often, it is immediately followed by some **activation** function. A common example of activation function is the **rectified Linear Unit (ReLU)**, defined as

$$a(z) = \max(0, z) \tag{1.9}$$

but others exist and are used depending on the type of task encountered. For instance hyperbolic tangent or sigmoid are other commonly used activation functions. They all share the fact of working in a non-linear fashion on the input values, adding non-linear behaviour to the output of the convolution. This role is crucial in determining the **feature map**.

The feature map is the name given to the output of the convolution after the processing executed by the activation function (in what is called the **detector** stage of the layer). Referring ourselves to the our previous reasoning regarding representation learning, the feature map is nothing but the abstract representation created by the convolutional layer, which is then used as an input by the successive layers of the CNN (see figure 1.6 for an example).

The feature map is usually followed by another mathematical operation called **pooling**. Pooling consists in the substitution of the values of the feature map at a certain location with a summary statistics of its neighbour locations. Example of pooling functions are the **max pooling**, where we select the maximum value of a neighbourhood of locations of the map, like a small rectangle or square, or some average of this neighbourhood. This operation causes a **downsampling** of the feature map, since it reduces the dimensionality of the output by selecting only a statistics of portions of it.

Finally, convolution along with activation and pooling constitute what is commonly called a **convolutional layer**[6].

As we have stated in the beginning of the subsection, CNNs have become a specialized tool for the processing of grid-like data, such as digital signals. The motivations lie in the specific nature of the layers: convolution is the core operation of **filtering** in digital signal processing (DSP), and the structure of convolutional layers makes the feature maps

[5] Note that convolution shows some close similarities to the mathematical operation of **cross-correlation**, which, in two dimensions, is defined as

$$C(i,j) = (I * K)(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

. This operation is almost equal to convolution without kernel flipped, and in fact many machine learning libraries implement this operation calling it convolution [1].

[6] Please note that there are two different terminologies when dealing with convolutional layers. One, called *complex* by [1], considers the convolutional layer as the sequential union of the three operations described before. The other, the *simple*, considers each operation as a separate layer. See figure 1.5 for a graphical representation.

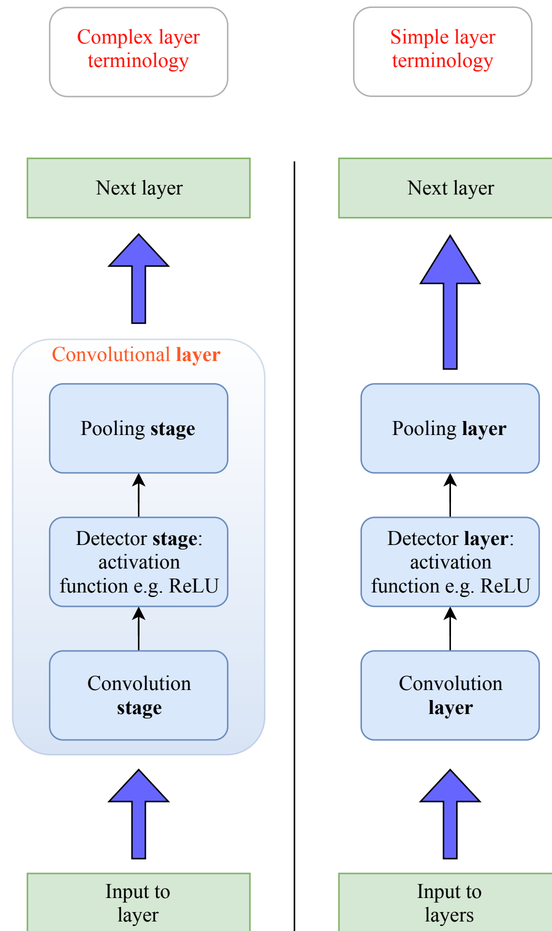

Figure 1.5: *Graphical representation of what is considered a convolutional layer by the two terminologies. Throughout this thesis, we will use the "complex" one.*

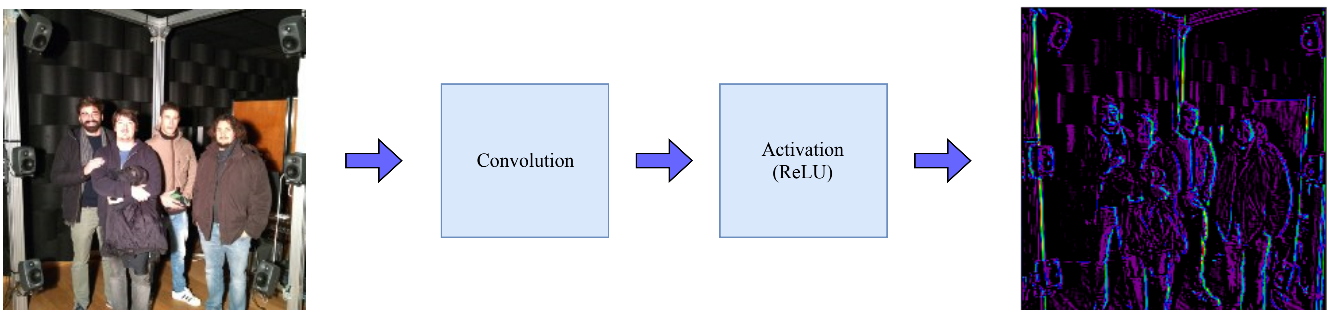

Figure 1.6: *Example of a feature map produced by the* 64th *filter of the first layer of the VGG16 network [2]. In this case, the activation function is a ReLU. We can see how some details of the image (like the edges) are more accentuated. This feature map will be then used by the successive layers of the network to produce a more abstract representation of the input.*

produced by CNNs have some really nice properties. Some of them derive from the characteristics of the convolution operation stage, such as the use of **sparse weights** and of **parameter sharing**, which produce **invariance to translation**; others from the pooling stage, which causes properties like the **invariance to local transformations**.

With *sparse weights* we indicate the fact that convolutional layers use a kernel that is much smaller than the input. Often one or more kernels are employed, so that the convolution stage implements what in DSP is called a **bank of filters**: it executes several convolutions/filtering in parallel. By keeping the number of parameters of these filters much lower than the dimension of the input, these allow to detect small meaningful features while still maintaining low memory requirements and achieving good execution performances. With respect to the MLP[7], besides the aforementioned improvements (convolutions are way faster than linear transformations), we have a complete different semantic of the operation applied to the data, that allows different interactions between the layers and the construction of abstractions that are more purposeful with respect to the nature of the input and of the task executed.

Moreover, while with the MLP we apply a linear transformation instead of the convolution, meaning that we have one parameter applied to each element of the input, the nature of convolution is such that the same kernel (or kernels in case we are implementing a bank of filters) is applied to the different portions of the input. This is the so called *parameter sharing* of the convolution stage. Besides the already cited advantages in memory requirements, the parameter sharing property makes the convolutional layer also *invariant to translations.*

Saying that a function c is equivariant to another function means that if the input is modified by some transformation expressed by a certain function g, the same modification appears in the output after applying c. In other words, this means that $c(g(x)) = g(c(x))$. Convolution has this property for some transformations, in particular for translation. If we take for instance as input an image I, and compute a translation of one pixel to the right $I' = I(x-1, y)$, the result of a convolution C would be the same if applied to I' or if applied to I and then translating the output. Convolution produces a sort of timeline that detects when particular features in the input show independently of their position. Moving a feature/event in the input would not make it undetectable, but it would only postpone its identification. However, please notice that convolution is not equivariant to all functions, like local transformations such as rotation or scaling.

Fortunately, the last stage of the convolutional layer introduces equiv-

[7] Again, here we will not deepen the description of the MLP. For the sake of the discussion, the reader can simply imagine a MLP as a CNN where each layer is a **perceptron**. The perceptron instead of performing a convolution in its first stage, applies a linear transformation of the type $y = Ax + b$, where $y \in \mathbb{R}^m, x \in \mathbb{R}^n$ are respectively the output and the input of the stage, and $A \in \mathbb{R}^{mxn}, b \in \mathbb{R}^m$ are the parameters of the transformation.

ariance to some other transformations. Pooling executed over spatial regions causes the layer to become invariant to translation exactly like the parameter sharing property; moreover, pooling over the outputs of different parametrized filters, each one producing a feature map of the same features but with different transformed inputs, the layer can become equivariant to the transformation applied to the input[8]. An interesting example is the invariance to rotation, as depicted in figure 1.7. In this case, three filters have been parametrized to learn the hand-written 5 digit with different rotations. Using a max-pooling over the outputs of the filters, therefore taking the output with the highest activation value in the detector stage, makes the layer detection of the digit invariant to rotation.

This last example gives us an idea of what kind of features CNNs are able to learn. As we said in the introduction of the section, deep learning tools learn meaningful features while learning the task, and in the years CNNs have been applied to different problems with encouraging results. Starting from handwritten digit recognition as done in [23], to the astonishing results obtained by the VGG16 network [2], CNNs are now a common tool in image processing, and recently the image forensics field too has started to use them with success. Examples are the work done for the detection of median filtered images done in [35], image manipulation detection in [21], and device identification done in [19].[9]

1.2.3 Recurrent Neural Networks (RNNs)

Just like CNNs are specialized networks for processing grid-like data, **recurrent neural networks (RNNs)** are a family of networks for processing **sequential** data. For sequential data we mean a sequence of values $x^{(1)}, x^{(2)}, \dots, x^{(\tau)}$ that are related through some temporal process generating it, where the apexes are the indexes indicating when the elements have been produced.

From CNNs, they take the idea of *parameter sharing*, with the difference that instead of using it for processing different portions of the input, the sharing happens across the different time steps of the sequence. This is obtained by making each output a function of

- the **input** at time index t;
- the **output** or **outputs** at some previous time step $t' < t - 1$;

and using the same function to compute the output at each sequence index. The networks therefore implement a **recurrent** formulation (from

[8]The equivariance property of the convolution and pooling stages can be thought as the assumption of an infinitely strong prior probability distribution that the function the layer learns must be invariant to some transformations. We address again the reader to [1], chapter 9 section 4, for a complete discussion of this idea.

[9]For what concerns video signals, our object of interest, and other works in the image splicing localization task, we address the reader to Chapter 2.

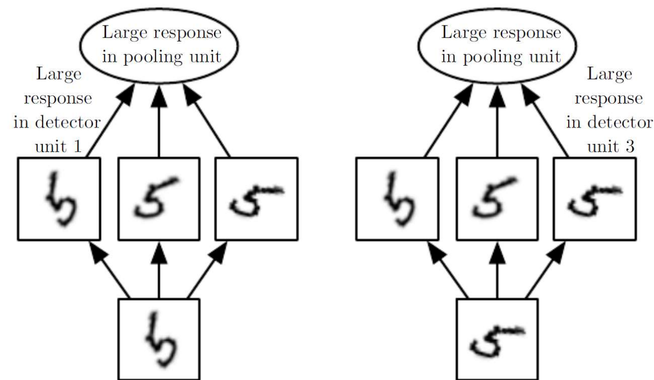

Figure 1.7: *Example of how pooling over the outputs of filters learning the same features with different parameters allow the layer to detect features invariantly with respect to the transformation applied to the input. Figure taken from [1], page 344.*

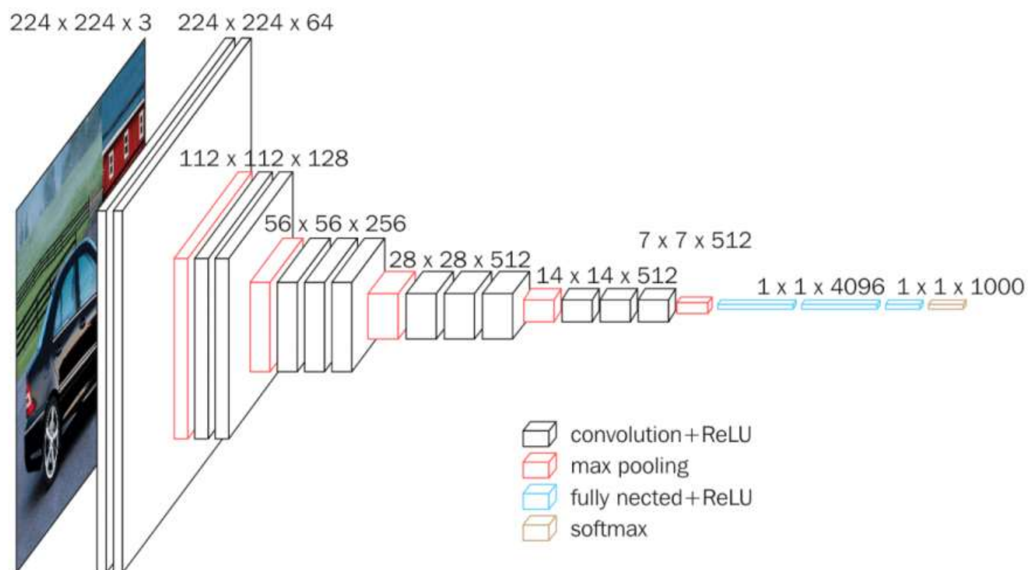

Figure 1.8: *Schematization of the architecture of the VGG16 model [2]. The model won the first and the second places in the localisation and classification tracks respectively at the ImageNet Challenge 2014. Figure taken from* https://neurohive.io/en/popular-networks/vgg16/.

here their name), resulting in a sharing of the parameters through a very deep computational graph.

Various architectural variations of RNNs exist, but almost all of them share the idea of computing the output relative to the next element in the sequence taking into account the whole sequence history seen by the network. This task is accomplished by using a special variable, called the **state of the network**, which is nothing more than the value of the outputs of the intermediate layers (the so called **hidden units**). Mathematically speaking, we can write the state/value of the hidden units $h^{(t)}$ at time index t as a vector

$$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, x^{(t-2)}, \dots, x^{(2)}, x^{(1)}) = f(h^{(t-1)}, x^{(t)}, \theta) \quad (1.10)$$

with $x^{(\tau)}$ being an input **vector** of features at time step τ and θ the parameters of the network. Equation 1.10 shows two possible interpretations of the computation of the state $h^{(t)}$: from one point of view, we can see it as the application of a function $g^{(t)}$ different at each time step t, since it considers sequences of different length; from another, $h^{(t)}$ is computed using the same function f with the same weights θ at each time step t.

In the latter case, RNNs use the state as a sort of lossy summary of the history of the inputs/outputs, since sequences of any length are always mapped to the same variable of fixed length, the hidden state $h^{(t)}$. However, this mapping in reality can be considered as an ability of the network to generalize information from sequences of any length, since we learn a single model f operating on all indexes instead of many functions $g^{(t)}$ as time steps t the sequence is long.

Besides this common trait among all RNNs, design patterns for this type of networks differentiate in many aspect such as the connections between hidden units, recurrence of the output, mismatch dimension between input and output sequence length, etc. Figure 1.9 shows two examples of a computational graphs for a general RNN architecture for input classification, where we have recurrent connections only between hidden units or between outputs and hidden units.

The figures highlight one operation which is essential in the use of RNNs, which is the **unfolding**. On the left side of both pictures in fact, we can see that the recurrent connection from previous states/outputs is implemented as a delay element. However, this type of representation does not allow to actually implement numerically the graph, for which an unfolding or unrolling, meaning a repetition of the graph across the different time steps, is therefore necessary.

Only through unfolding is it possible to actually perform training, through an adapted version of the backpropagation algorithm called **backpropagation through time (BPTT)**. This is not really a specialized algorithm, but only the application of the backprop in the unfolded computational graph of the RNN, taking into account each time step of the sequence. Using BPTT, it is possible to perform a normal training

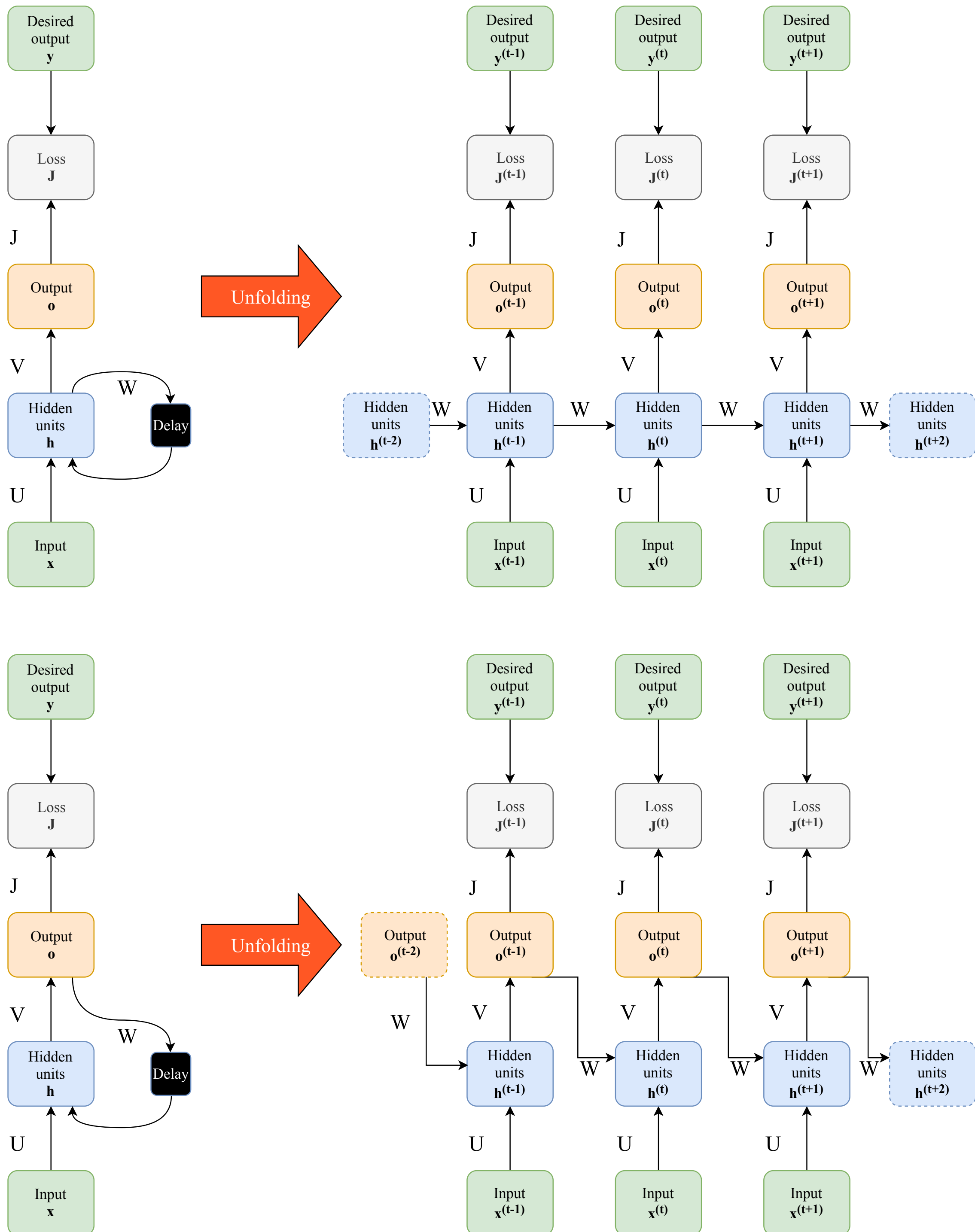

Figure 1.9: *Computational graphs from the folded and unfolded point of view of two generic RNNs used for a classification task where the input and output have the same length. The two design patterns differ for the use of the previous state or of the previous output for the computation of the next state. Therefore, they differ in the function W used for the computation of the next state.*

of the networks: however, computing the gradients with respect to the unrolled graph reveals to be complicated somehow.

Two major problem in fact arise in the training of RNNs. The first is the fact that the forward propagation of the input in the network is inherently sequential, and this affects the network performances (but with some design patterns this problem is not a major concern); the second is defined as the **vanishing gradient** problem.

We have seen in the first subsection that the function implemented by a neural network derives from the composition of the functions characterizing each layer, like the example of equation 1.4. The same situation happens for RNNs, where the unfolded computational graph involves the composition of the same function multiple times (once per time step), leading to an highly non-linear behaviour (see figure 1.10 for an example). The vanishing gradient problem manifests for this reason: when learning long-term dependencies (when we are processing very long sequences), the gradients propagated over the steps of the sequences tend either to vanish or to explode.

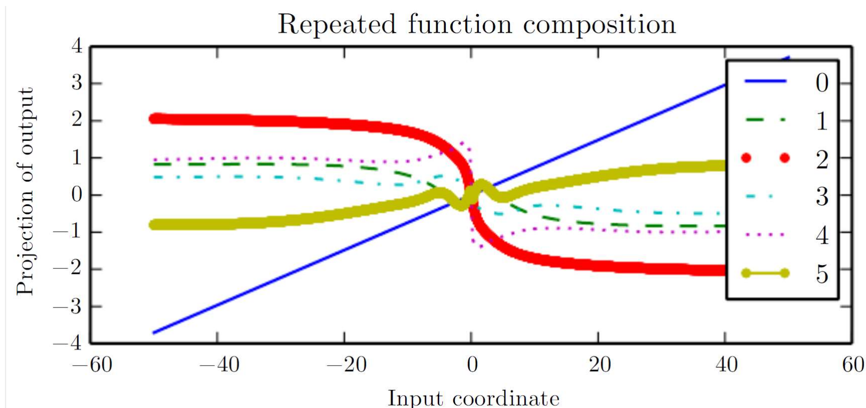

Figure 1.10: *1D linear projection of a 100-dimensional hidden state of a RNN, resulting from the (multiple) composition of a non-linear activation function (an hyperbolic tangent in this case) due to the unrolling procedure of the computational graph. On the y axis we have the projection, on the x axis the coordinate of the initial state along a random direction in the 100-dimensional space. The figure shows the behaviour of the hidden state up to 5 composition of the* tanh *function, so up to 5 steps in the unrolling graph. Taken from [1], page 403.*

Taking as an example the first RNN of figure 1.9, let's suppose that the computation of the state $h^{(t)}$ at step t is described by the recurrent function W, which in this case is a simple linear transformation with no activation function, something along the line of

$$h^{(t)} = W^T h^{(t-1)}, \tag{1.11}$$

where the matrix W with its elements represents the parameters of the recurrent layer and where we are omitting the input $x^{(t)}$ for simplicity.

Clearly, taking the initial state $h^{(0)}$ we can rewrite $h^{(t)}$ as

$$h^{(t)} = (W^t)^T h^{(0)}. \tag{1.12}$$

Let us assume that W admits an eigenvalue decomposition of the type $W = Q\mathbf{\Lambda}Q^T$. Equation 1.12 becomes equal to

$$h^{(t)} = Q^T \mathbf{\Lambda}^t Q h^{(0)}. \tag{1.13}$$

The eigenvalues of vector $\mathbf{\Lambda}$ are raised to the power of t, causing those whose magnitude is less than 1 to vanish, and those whose magnitude is greater than 1 to explode, discarding any component of the hidden state $h^{(0)}$ which is not aligned with them[10] [11].

This problem is common to CNNs and MLPs which show a very deep architecture, but indeed it is really particular to RNNs due to the nature of the unfolded computational graph. For this reason, while for CNNs some fixes have been found only recently [37], from the discovery of the issue [38] [36] in the RNNs field several solutions have been proposed.

One idea is to not learn the weights of the recurrent connection's function: learn only the parameters of the output functions, and fix the recurrent weights so that they capture a rich history of the past inputs/outputs. In this way therefore the problem of vanishing gradients is avoided. This is the design pattern followed by **echo state networks (ESN)** [39] and **liquid state machines** [40], which are also generally referred to as *reservoir computing*.

Another solution is to not let the gradient flow freely in the unfolded computational graph. The main problem of vanishing gradients indeed is that, pursuing the goal of learning long-term dependencies, we let

[10]Here we have simplified the discussion in order to show the basic motivation behind the phenomenon of vanishing gradients. A more complete demonstration based on the power method can be found in [1], page 289.

[11]Please note that the vanishing gradient problem *does not* prevent the network from learning the right parameters for the task. The problem lies in the fact that gradients deriving from a long-term interaction (a very long sequence) will have a very small magnitude compared to the those of a small-term interaction. Therefore, it would take a very long time to learn something from them, since the variations of the weights would be constituted of very small fluctuations easily hidden by those caused by shorter-term interactions. However, what happens in practice is that gradient-based optimization becomes increasingly difficult, with success in training almost impossible for even very short sequences (10-20 elements) [1] [36].

the gradients of the recurrent connections[12] flow freely without neither controlling their values nor checking which of the weights are useful for learning the temporal information. **Gated RNNs** starting from this idea, by using mechanisms called **gates**, control the flow of data representing the history of inputs through the different time steps of the unrolled computational graph.

As for reservoir computing networks, different variants of gated RNNs have been devised during the last 20 years. One model in particular has gained a lot of attention: the **Long-Short Term Memory (LSTM) model**, first published in 1997 by Hochreiter et al. [27], has been widely adopted with good results in a great variety of tasks involving sequences of data [28]. LSTM uses three types of gates, which can be considered like layers from a certain point of view, since they are composed by a linear transformation, followed by an activation function (a sigmoid in this case), and finally by a point-wise element multiplication. The three gates control different parts of the variables used by the function implemented by a recurrent layer, and specifically they are called:

- **forget gate**, which is used to control which elements of the **old state** $h^{(t-1)}$ might or might not be preserved;
- **input gate**, which instead controls which elements of the **new input** $x^{(t)}$ are useful for the task;
- **output gate**, which finally decides which elements of the **new output** $y^{(t)}$ are relevant.

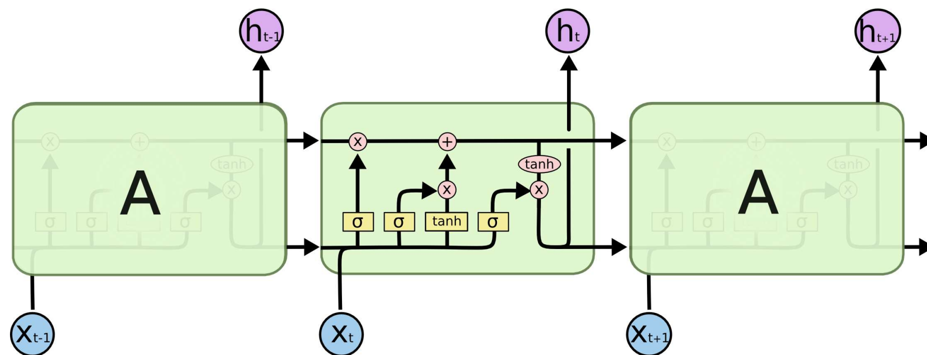

Figure 1.11: *Example of a unfolded computational graph of a RNN implementing the LSTM model. We can see the internal structure composed by the three gates. Picture taken from [3].*

In figure 1.11 we can see the internal structure of a LSTM layer with the three gates, which is also usually referred to as **cell**.

[12]Here we refer to recurrent connection as recurrent connections' weights, meaning the weights of the functions realizing the connections in the computational graph.

To understand how the gates work, we can take a closer look to the forget gate. This layer takes as input the previous output $y^{(t-1)}$ and the actual input $x^{(t)}$, concatenates them and multiply them by a matrix W_f. It then produces as output a vector $f^{(t)}$ of the same dimension of the cell state according to the formula

$$f^{(t)} = \sigma(W_f \cdot [y^{(t-1)}, x^{(t)}] + b_f). \tag{1.14}$$

with σ being a sigmoid activation function, $\cdot$ a matrix multiplication and $[\quad]$ a concatenation operation. The **forget vector** $f^{(t)}$ can be considered as a linear transformation of the previous input and output, whose values are forced by the sigmoid to lie in the range 0-1. Since the vector has the same dimension of the "old" state $h^{(t-1)}$, we can view it as a simple vector of weights which, indeed, is multiplied element-wise to $h^{(t-1)}$ to decide which elements of it to preserve (see equations 1.16).

The input and output gate behave similarly, with their outputs described respectively by the following formulas

$$\begin{aligned} i^{(t)} &= \sigma(W_i \cdot [y^{(t-1)}, x^{(t)}] + b_i), \\ o^{(t)} &= \sigma(W_o \cdot [y^{(t-1)}, x^{(t)}] + b_o), \end{aligned} \tag{1.15}$$

with separate matrices and biases vectors W_i, W_o, b_i and b_o. The new cell states and outputs are then produced according to the following formulas

$$\begin{aligned} \tilde{h}^{(t)} &= \tanh(W_h \cdot [y^{(t-1)}, x^{(t)}] + b_h), \\ h^{(t)} &= f^{(t)} \circ h^{(t-1)} + i^{(t)} \circ \tilde{h}^{(t)}, \\ y^{(t)} &= o^{(t)} \circ \tanh(h^{(t)}). \end{aligned} \tag{1.16}$$

where $\tilde{h}^{(t)}$ is the **candidate state** for the time index t, $\circ$ denotes a element-wise multiplication and tanh is a simple hyperbolic tangent.

While the whole set of equations describing the functioning of the LSTM model might be quite cumbersome (and appalling in some sense...), and we suggest anybody who is interested to deepen the argument in the really accurate and intuitive walk through by Christopher Olah [3], we would like to focus the reader's attention on two main points:

1. the gates, each one with their separate sets of parameters, allow to control the data flow in different points of the unrolled computational graph. This is accomplished by generating vectors of weights in the range 0-1 which are multiplied element-wise to the variables of interest of the recurrent function $f(x^{(t)}, h^{(t-1)})$ implemented by the network. The network therefore is able to learn during training to discriminate between which information are more useful between the new input or the previous history of the sequence to accomplish the task;

2. the presence of the gate functions makes the computational graph have gradients that do not explode neither vanish, but with respect to reservoir computing networks the weights of the recurrent connections are not set manually, but learned during training.

The description carried out until now is the one of the "classic" LSTM model presented by Hochreiter et al. [27]. We have seen how each gate can be considered as a sort of MLP layer, composed by a linear transformation followed by a sigmoid activation function. However, this kind of design somehow limits the capabilities given by the gates' mechanism to use the sequence history depending on the type of input data given.

In the previous subsection on CNNs we have analyzed some of the benefits that the convolutional stage gave in the analysis of grid-like data. But what about data with both grid-like and sequential topology, just like **video files**? It would be really convenient to have the benefits of both recurrent and convolutional architectural patterns.

Aiming at this goal, Chen et al. [35] presented in 2015 the **convolutional LSTM (convLSTM) model**. We can see the convolutional LSTM as an LSTM model where instead of having simple linear transformations in the input-state and state-state transition in the gates, presents convolutions, resulting in the full model described by the following set of equations:

$$
\begin{aligned}
i^{(t)} &= \sigma(W_{xi} * \mathcal{X}^{(t)} + W_{yi} * \mathcal{Y}^{(t-1)} + W_{hi} \circ \mathcal{H}^{(t-1)} + b_i), \\
f^{(t)} &= \sigma(W_{xf} * \mathcal{X}^{(t)} + W_{yf} * \mathcal{Y}^{(t-1)} + W_{hf} \circ \mathcal{H}^{(t-1)} + b_f), \\
o^{(t)} &= \sigma(W_{xo} * \mathcal{X}^{(t)} + W_{yo} * \mathcal{Y}^{(t-1)} + W_{ho} \circ \mathcal{H}^{(t)} + b_o), \\
\tilde{\mathcal{H}}^{(t)} &= \tanh(W_{xh} * \mathcal{X}^{(t)} + W_{yh} * \mathcal{Y}^{(t-1)} + b_h), \\
\mathcal{H}^{(t)} &= f^{(t)} \circ \mathcal{H}^{(t-1)} + i^{(t)} \circ \tilde{\mathcal{H}}^{(t)}, \\
\mathcal{Y}^{(t)} &= o^{(t)} \circ \tanh(\mathcal{H}^{(t)}).
\end{aligned}
\tag{1.17}
$$

Here the letters have the same meaning used for the previous description of the classic LSTM model, the $*$ means a convolution operation and the $\circ$ the Hadamart product (element-wise multiplication), and the matrices of parameters instead of being multiplied in a linear transformation are used as kernels in convolutions.

The main remarkable difference is in the nature of the input and output data $\mathcal{X}$ and $\mathcal{Y}$, as well in the state $\mathcal{H}$, which are no more vectors, but **3D tensors** where the first dimension is the temporal (sequence) one, and the last two represents row and columns of a matrix-like structure (like an **image**). The convolutional LSTM therefore, due to the use of convolutions in its gates, determines the future state of a certain cell in the grid by inputs and past states of its local neighbours, combining the characteristics in handling sequences of inputs of the LSTM model with those of CNNs in processing grid-like data (see figure 1.12).

Even if the authors used histories of radar maps for precipitation forecasting, this kind of network can be easily adapted to process 3D data like **videos** as well with simple modifications.

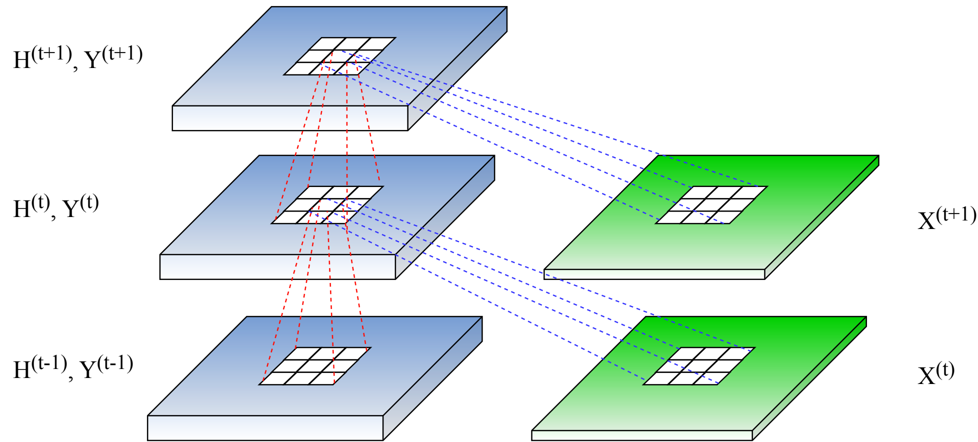

Figure 1.12: *Use of the convolution operation inside the gates of the convLSTM for the computation of the output $\mathcal{Y}^{(\tau)}$ and state $\mathcal{H}^{(\tau)}$. From this point of view is clearer the nature of the variables as 3D tensors.*

1.2.4 Autoencoders

We close this chapter introducing **autoencoders**, a type of neural networks specialized in the task of **copying their input to their output**. Autoencoders can be thought as composed by two parts:

1. an **encoder network**, which maps the input x to an output $h = f(x)$ called the **hidden or internal representation**, also referred to as the **code** of the autoencoder;

2. a **decoder network**, which executes the inverse operation $g(f(x)) = x$ of mapping the code to the original input.

While this task might seem trivial, there are several reason behind having such kind of tool. Autoencoders in fact are trained *not* to reproduce the input perfectly. In doing such an operation, the model is therefore constrained in selecting, inside the hidden representation, which aspects of the data to preserve for the decoding operation, in this way implicitly learning useful properties about the input.

Recalling our previous discussion about representation learning, we can see autoencoders as neural networks whose main and only task is that of learning representative features of the input, which are "encoded" in the hidden representation indeed, in different ways which are defined by the deep learning practitioner again with the **loss function** used during training.

One of the way through which autoencoders are able to learn useful properties about the data is to force the hidden representation to have smaller dimensions than the input. Such family of autoencoders is called **undercomplete**, and comprehends all kinds of networks that map the input to its output having a code of smaller dimensionality with respect to the input. Learning an undercomplete representation forces the network to learn only the distinctive features of the training data, with the loss function defining which features are distinctive or not.

A simple example of such kind of architecture is the *linear autoencoder*. In this case, encoder and autoencoder are both composed by a perceptron without activation function, and the loss function is simply defined as the MSE between input and reconstructed output: in this case, the operations executed are the same of the **Principal Component Analysis (PCA)**, and in fact the hidden representation is nothing more than the projection of the input on the same span of the PCA.

Obviously, by using a normal perceptron with a nonlinear activation function more salient features can be learned. In fact, any kind of architecture, even deep, can be used to create an autoencoder, comprehending CNNs or RNNs as well. However, another way to capture useful features about the data is to use more complicated loss functions than the MSE. This is the case of **regularized autoencoders**, where by using a regularization term in the loss function we encourage the model to have other properties besides the dimensionality reduction of the input.

Regularized loss are loss functions of the type

$$\tilde{L}(\mathbf{y}; f(\mathbf{x}; \mathbf{w})) = L_1(\mathbf{y}; f(\mathbf{x}; \mathbf{w})) + L_2(\mathbf{w}), \tag{1.18}$$

with $\mathbf{x}, \mathbf{y}$ and $\mathbf{w}$ being respectively the input and relative output of the training set and the parameters of the network. They are composed by two terms where the first is relative to task, so in how the input should be processed with respect to its corresponding output in the training set, and the second is a term relative only to the model, often defined with respect to its parameters $\mathbf{w}$.

Regularization terms are usually employed for avoiding the phenomenon of **overfitting**, which describes the behaviour of a network which is not able to generalize to unseen data (which is at the base of the dataset splitting into training and validation set too). However, regularization terms can be also used to encourage properties of the learned model. One of them is the **sparsity**, which leads in the case of the autoencoders to **sparse autoencoders**: the model tunes to only determined features of the data, so rather than learning an identity mapping from input to output, responds only to unique statistical features of the dataset.

Another desiderable characteristic for models is the ability of processing noisy data, so to be robust to noise in the input. Autoencoders with this property are called **denoising autoencoders**, and their aim is to learn the underlying distribution of the data regardless of the noise affecting them. A specialized version of denoising autoencoders are the

contractive autoencoders, whose feature extraction process is able to resist even to the tiniest variation of the input.

In literature, many other different types of regularized autoencoders have been proposed. However, we can state that all of them base their effectiveness on the idea that their input data concentrates around a low-dimensional manifold, or a small sets of them. Manifolds are connected regions, sets of points with a neighbourhood associated to each of them, that locally appears as an Euclidean space. A common example is the surface of the Earth, a spherical manifold in a 3D space that appears as a 2D plane.

In machine learning usually manifolds tend to be used to designate a connected set of points that can be approximated well by considering only a small number of degrees of freedom, or dimensions, embedded in a higher-dimensional space, with each dimension representing a direction of local variation. What machine learning actually do, for training data defined in $\mathbb{R}^n$, is effectively to learn manifolds of interests: instead of learning functions with variations across all $\mathbb{R}^n$, we assume that most points of $\mathbb{R}^n$ are invalid inputs, and that the interesting inputs lie in a manifold or sets of manifolds with their local directions of variation being the variations of interest of the function approximated by the algorithm. From a probabilistic perspective, we can see **manifold learning** as the process of modeling the regions of the probability density/mass function of the input data which are most concentrated.

Regularized autoencoders during training need to balance two important factors: learn a representation such that the input can be approximately recovered through the decoding function, and satisfy the additional constraints imposed by the regularization terms of the loss. Since the training points are drawn from the training set, so a set of points that are probable in the domain of the function, this results in the fact that autoencoders can afford to learn only variations that are essential to reconstruct the training samples. If the data generating distribution concentrates near a low-dimensional manifold, this yields representations that implicitly capture a local coordinate system for this manifold.

Autoencoders therefore are able to learn in their hidden representation information regarding the local manifold, or probability distribution, generating the input data. We would see that these concepts are essential in the use of autoencoders in **unsupervised learning** tasks, such as **anomaly detection**.

2
State of the Art (SOA)

This chapter introduces a review of the **state of the art (SOA)** techniques proposed in the literature for **splicing detection and localization**, either using deep learning instruments or not, applied to **video signals**. As we have stated previously, techniques in video forensics have their roots in those developed for images. For this reason, we will first dedicate a section on methods for image tampering detection, then a section on methods for video tampering detection. In both we will initially analyze methods that do not make use of deep learning techniques, and then focus on works that exploit them. Since the literature for video forensics has fewer contributions for what concerns splicing localization, we will list briefly some other works of interest for our task that are, however, not immediately related to our work.

2.1 Images tampering detection overview

When dealing with image tampering detection, the first point to make is the differentiation of the types of forgeries we might encounter. The most common categories of forgeries to detect are the **copy-move**, **splicing**, **inpainting** and the **broad-scope image operations**.

Copy-move refers to taking a part of the image and copy it on another portion of it, either to add false information or to hide the content of the portion covered. Splicing, our matter of study in this thesis, is a very similar forgery that differs from copy-move only for the fact that the portion of image copied is taken from a different source. Inpainting is used instead to refer the "drawing" on the image with a software editing tool, such as a "brush", with the purpose of reconstructing missing or damaged parts of it: we can see it basically as an interpolation. Finally

with broad-scope image operations we indicate all those operations such as cropping, filtering, rescaling or histogram adjustments that can be performed even without malicious intent.

We can easily see that both copy-move and splicing cause some disturbance in the local image structure, and in some sense the copy-move forgery can be considered as a "specialized" technique of splicing. For this reason, splicing detection algorithms are usually able to recognize copy-move forgeries too [41].

Although almost all algorithms dedicated to image splicing are based on the assumption that the portion of image spliced will differ for some of its characteristics from the rest of it, many of them are able to only **detect** the splicing, therefore to say if the image is spliced or not, without localizing where. In this work instead we focus on the **localization** of splicing, and for this reason in the following pages we will review methods dedicated to this task.

In the first section of Chapter 1 we have outlined the three major steps in the lifecycle of a multimedia object, and saw how each of them leaves some traces on it that can be used to reconstruct the processing chain it has undergone. All of the traces we have seen previously can be used to determine where an image has been spliced.

We have seen **PRNU** noises being a unique trace left by the APS of the camera during the acquisition stage, and that it can be used to assess the authenticity of an image under analysis. Several authors exploit this footprint, and the basic idea shared by them is that having at disposal a set of varied images taken by the same device, it is possible to estimate the device's PRNU. Thus, having different sets of images taken by different devices, it is possible to have a set of PRNU noises of several cameras. They can then be used for evaluating whether the image under question conforms to any of them. If local deviations appear, the presence of a splice in the corresponding region is posited. Works along these line are [42] [43] [44].

Other authors exploit the fact that the combination of capturing device, the capture parameters of each image, and any subsequent image postprocessing or compression, create unique **noise patterns** that differ from image to image. In [45] the authors localize splicing by identifying noise patterns wavelet-filtering the image, assuming that the local variance of the high-frequency channel will differ significantly between the splice and the recipient image. In [46], local noise is isolated by observing that in the frequency domain, each sub-band of the frequency content of an image has positive kurtosis coefficient, and allows to discriminate between spliced and clean regions. A significant work done by Cozzolino et al. in [47] constructs synthetic features from co-occurrences of residuals of the image filtered with a linear high-pass filter: assuming that spliced regions will exhibit different statistical properties of these features, the authors developed an algorithm, **Splicebuster**, that works both in an unsupervised and semi-supervised scenario. In the latter, the user indi-

cates a region that it is possibly tampered; the algorithm then calculates the natural statistical properties of the co-occurrences of that portion of the image, and consecutively evaluates regions of the rest of the image with respect to their conformance to the constructed model. In the unsupervised case instead we are not sure of which areas of the image have not been tampered: therefore, the method uses an Expectation-Maximization algorithm to fit two different distributions (tampered and non-tampered) to the local descriptors.

CFA patterns are exploited too. Splicing in fact can easily disrupt the original interpolation pattern of the image, mainly because different cameras use different CFAs and even different interpolation algorithms. A spliced image would therefore highly probably present discontinuities, but not only. Indeed, spliced regions are often rescaled or filtered, therefore they change heavily the original pixel interrelation; moreover, even simply misaligning the splice with the rest of the image disrupts the CFA pattern. Typical CFA array patterns follow a 2x2 grid: thus, there is a 75% chance that any splice (or even copy-move) forgery will alter it. The authors of [48] [49] [50] use this information to detect spliced regions. In [49], they use two methods. In the first, they try to detect the CFA pattern used during the image acquisition stage by subsampling the image using various possible selection patterns; then, they re-interpolate it, and finally compare it to the original, looking for discrepancies between the reconstructed and the original image in order to detect spliced regions. In the second, by filtering the image with a de-noising filter they compute the image noise pattern, and then calculate a measure of relationship between the noise variance of interpolated and natural pixels. They then show that there is a high probability that pixel values have been disrupted by tampering if the two variances in a region are highly similar. In [50] the authors compute the same measure of the variance relative to natural and interpolated pixels, but in this case they estimate on a block per block basis a probability that the image has been tampered.

The main drawback of methods exploiting CFA patterns is that they tend to be easily covered by simple scheme of compression, like the JPEG. Fortunately, almost all lossy compression formats leave recognizable artifacts too, and in fact another prosperous field in the forensics literature of splicing detection is based on **JPEG compression footprints**. These can be broadly divided in two categories: **quantization artifacts** and **grid discontinuities**.

The first family of footprints is relative to the process of quantization of the DCT coefficients in the JPEG processing chain. What usually happens when a splicing is performed, is that the malicious attacker would process the spliced region, add it to the tampered image and then usually saved it in a compressed format. Assuming that the original image was saved in a JPEG format too, and that the spliced portion does not show compression artifacts relative to its previous history, the tampered image would exhibit traces of a **double compression**.

It has been observed that the DCT coefficients' distribution exhibits some periodicity when multiple compressions have been executed on the image. This observation lead therefore to the family of splicing localization methods denoted as **Double Quantization**, which attempt to model the periodic DCT coefficients' patterns caused by a double compression, and detect local regions that do not fall into this model. Works based on this concept are [51] [52] and [53]. In [53], the distribution of the DCT coefficients is modeled for the entire image, and then the image is divided in blocks of the JPEG format dimension and each of those is evaluated with respect to its conformance to the overall image model, evaluating the probability that it has originated from a different distribution. In [51], the authors extend this method, making the model significantly more robust by considering that the DCT coefficients' distribution estimation may be influenced by the presence of both tampered and non-tampered blocks. The work in [52] is based instead on another observation: the value of the distribution of the DCT coefficients follows the Benford's law, meaning that in the first digits low values are significantly more probable with respect to high values. When multiple JPEG compressions are performed, this characteristic fades. Therefore, the authors trained a set of Support Vector Machines (SVMs), a machine learning algorithm for binary classification, for assigning the values of the DCT coefficients' distribution to a single or double compressed one. Dividing the image in blocks, they are therefore able to indicate as a strong evidence of tampering the presence of single-compressed portions of the image.

For what concerns grid discontinuities methods, they are based on the quantization performed by the JPEG format, which is based on a 8x8 block grid. Based on the previous assumption that the spliced content does not show evident signs of its previous history, traces of splicing therefore simply derive by the absence or by the misalignment of the JPEG grid on original and tampered regions (in case these last ones show remaining sign of a previous compression). In [54], descriptor vectors which model the presence of such JPEG blocking artifacts are extracted from the image, and then used for training a SVM for detecting, but not localizing, image-level inconsistencies. In [55] instead the authors use as feature the local intensity of the blocking pattern. The features variations indicate local absence or misalignment of the grid, but they can depend also on the image content.

Other two methods for splicing localization based on JPEG artifacts are the **JPEG Ghost** and the **Error Level Analysis**. Both methods rely on two considerations on multiple JPEG compressions. The first, proposed in [56], considers the fact that compression with different **quality factors** leads to portions of the image that have different range of values. By taking the image under analysis and re-compressing it with different factors, then subtracting the image to itself, it is possible to construct a residual map where areas with a different compression factor

from the test ones are highlighted. This provides evidence that those regions might be spliced on the original content and come from a different quantized source. The second relies on the evidence that if an image undergoes multiple JPEG compression, it starts losing its high-frequency content. This means that if we assume the multiple compression hypothesis of splicing, if we subtract the image to itself and then filter it with an high-pass filter, spliced regions, which have undergone less compressions, will present an higher residual than the original portions of the image. This idea is exploited in [57] and [58].

In the last years some works dedicated to image forensics using **deep learning tools** have come to the attention of the research community. The most notable ones are the work done in [19], where CNNs have being used for device identification, and a successive work by the same authors [4]. This last one proposes a splicing localization method in the scenario of spliced regions and original image coming from different cameras. The proposed method exploits CNNs for features extraction, and then analyzes them by means of iterative clustering techniques in order to detect whether and where the image under analysis has been forged. The pipeline used by the authors is depicted in figure 2.1.

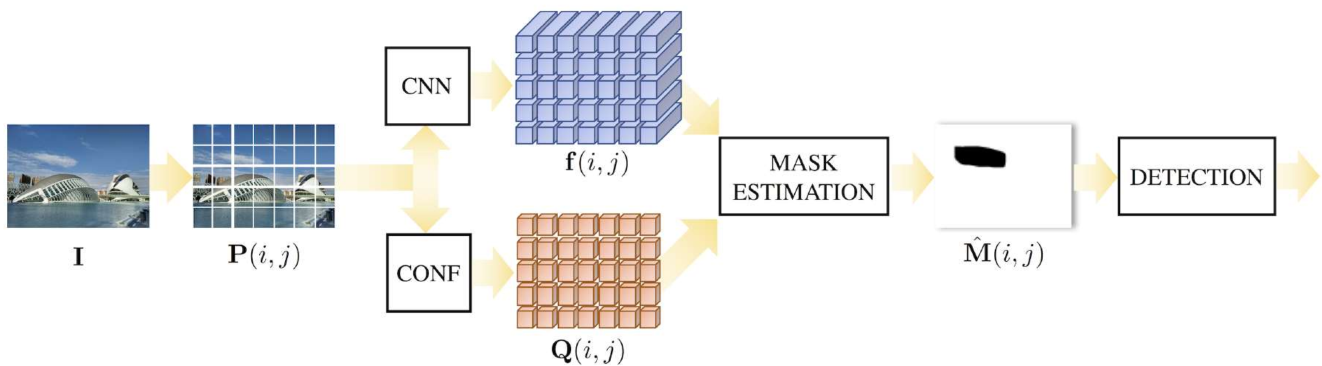

Figure 2.1: *Pipeline used in [4] for splicing localization.*

Another recent work related to splicing localization has been presented by Yarlagadda et al. in [10]. The authors tackled the task of splicing localization in images coming from satellite technology. Using a completely unsupervised approach, the authors exploit CNNs as feature extractor and generative adversarial networks (GANs) for creating a feature representation of pristine images. They then train a SVM to determine their distribution and localize splicings of different dimensions and shapes.

A similar work exploiting a unsupervised approach is the one by Cozzolino and Verdoliva in [5], with their pipeline shown in picture 2.2.

The authors worked using the same features presented in [47], this time employing **autoencoders** to construct an internal representation of the distribution of these features for clean frames. They show that using an offline training procedure with **discriminative labeling** and adding a **regularization term** in the loss function, the autoencoder at last is able to distinguish the spliced regions of the image with promising results.

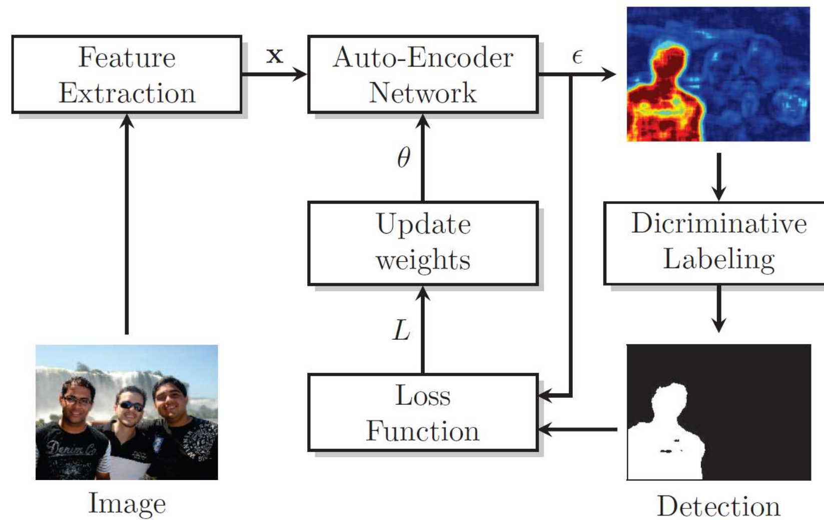

Figure 2.2: *Block diagram of the algorithm proposed in [5].*

Next we will see some works dedicated to the area of video splicing localization.

2.2 Video splicing localization overview

Video splicing localization literature has less contributions with respect to the images' one. The reason, as we have stated several times previously, lies in the nature of video signals, which, for their dimensionality, on one hand are more difficult to manage, and on the other are exposed to a wider range of manipulations for the forger that makes the forensics analysis more difficult.

A forensics analysis based on methods developed for images is possible for videos too, if we consider videos as a simple collection in time of frames [18]. All methods seen in the previous section are therefore appliable. However, precedently we have discussed how the presence of the temporal dimension in video signals gave to the forger the possibility of altering or hiding the content of the signal without modifying

the frame in its singularity. We can call this type of attack as **temporal splicing**, or **video-based attack**, and it mainly consists in **frame deletion** or **addition** (see figure 2.3). More often, what happens is that the forger modifies the video both in *space* and *time*, either by using a spatio-temporal region of the same video (a sort of video copy-move attack), or by employing a region coming from a different source, or by replacing it with a still image. In this case we call this type of splicing **image-based attack**.

Specific algorithms for video splicing detection and localization, which therefore can take place in the temporal dimension only without altering the frames' content, are harder to find. Thus, in the following we will simply list all works related to video forensics, trying to focus our attention on the most specific on both temporal and spatio-temporal splicing localization.

In [59], the authors study the effect of frame deletion and addition and the resulting fingerprints left by the operation in video signals. They also study the effect of **anti-forensics techniques**, methods aimed at hiding or deleting fingerprints left by a forging operation, developing an approach for detecting the fingerprints left by these anti-forensics techniques too. In [6], the authors develop methods for the detection of both temporal splicing and spatio-temporal splicing (video-based and image-based attacks), using an algorithm based on iterative morphological operations and clustering which is completely unsupervised (no information about spliced regions of the video sequences are given in advance).

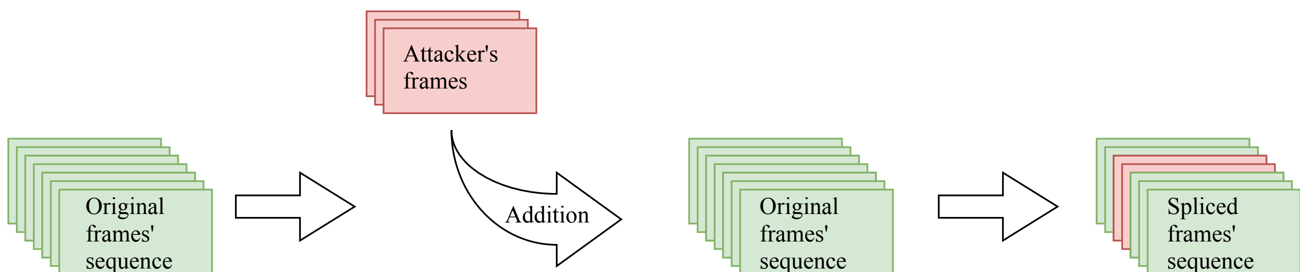

Figure 2.3: *Representation of temporal splicing with frame addition.*

In [60] the problem of re-encoding artifacts is studied. For re-encoding artifacts we refer to the problem of re-encoding of low-quality bitrate videos with high-quality factors. The authors have analyzed videos in MPEG-2 format, developing a 16-dimensional feature vector through which are able to recover the original bitrate of the video. Interestingly, some of the artifacts noticed by them are similar to the double quantization factor of double JPEG compression, in particular those related to the DCT coefficient's distribution (since the MPEG-2 employs a coding procedure for each frame similar to the JPEG's). In [7] the authors similarly analyze re-encoded videos, aiming at blindly discover the original codec adopted in the first compression along with some of

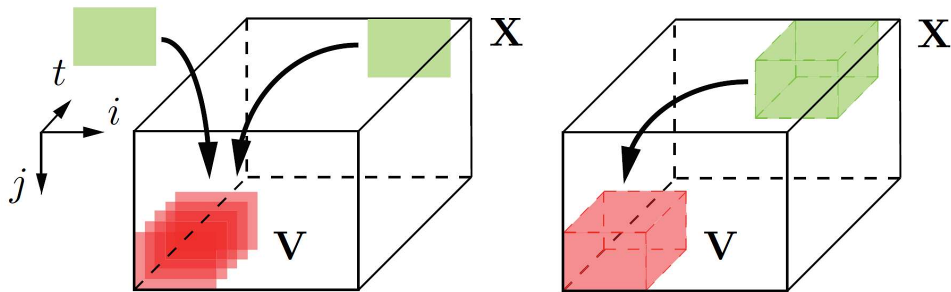

Figure 2.4: *Representation of image based attack taken from [6]. The forger can perform this operation by repeating a still image in time or by inserting a whole 3D volume, in this case taken from another region of the video.*

its parameters, specifically the original **Group Of Pictures (GOP)**. The functioning of the proposed method relies on the fact that coding is almost an idempotent operation, meaning that re-encoding a sequence using the same parameters would produce a sequence with very small difference from the original. Let's imagine a simple processing chain like the one pictured in figure 2.5.

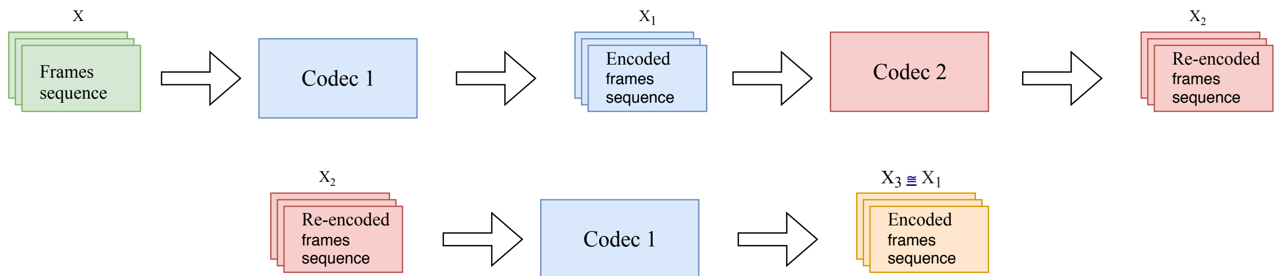

Figure 2.5: *Compression chain assumed by the authors of [7].*

By exploiting the idempotency of the encoding operation, it is possible to re-encode the double compressed video with different parameters, until the original codec configuration is found by looking at the maxima in the **Peak Signal to Noise Ratio** of the residual maps obtained by subtracting the re-encoded video X_2 to the "triple" encoded video X_3. The core idea is the exploitation of the artifacts left by the quantization step of the encoding, and it might considered as an extension of the JPEG Ghost method.

For what concerns video splicing localization, a pioneering contribution is the one presented in [61]. The authors use a **Gaussian Mixture**

Model (GMM) to estimate the distribution of the correlation of temporal noise residuals of forged videos on a block per block basis. They then use the estimated parameters of the model for training a Bayesian classifier to determine if a block of the video under analysis is tampered or not. A more recent work [62] extracts statistical descriptors from noise residuals on a block per block basis too, computing then a statistical model that allows: given a sequence, to determine if and where it has been spliced; given two, to determine if they come from the same acquisition device. Noise descriptors are computed also by the authors of [63], in this case to localize temporal splicing in video sequences obtained by the composition of clips coming from different devices. The idea is that the noise residuals (combination of PRNU and other noise characteristic of the acquisition device) can characterize a sequence of frame, so the temporal correlation of the residual noise of the shots are different if coming from separate devices, and can be used to localize the splicing.

In [64], a specific type of spatio-temporal splicing is studied, the **copy-move**, consisting in the splicing of a portion of the video with a region coming from the video itself. Copy-move has been exhaustively studied in the image forensics field, and almost any SOA method exploits a very simple idea: divide the image in patches; extract some features from them; finally, look for patches which are almost identical in two separate regions of the image. In executing these operations they usually compute what is called a **nearest-neighbour field (NNF)**, which is the set of nearest pixels with best matching features. The computation of such field can also be applied to video signals, but it is extremely expensive from the computational point of view (even for images, since the complexity grows quadratically with their size). For this reason, the authors exploit the idea of PatchMatch [65], an algorithm producing a dense approximated NNF over an image which is orders of magnitude faster than general-purpose techniques, adapting it for 3D signals (videos). They develop a method capable to detect copy-move splicing which is also resistant to some common post-processing operation like rotation, rescaling, etc., thanks to a set of noise-resilient rotation-invariant features. Using some *ad hoc* operations, the authors are also able to remove some false alarms.

Some very recent works tackle some new type of splicing attacks which employ **deep learning tools**. This is the case of the Face2Face [66] and **DeepFake** softwares, the last one which is able to splice a video with artificially generated faces using a combination of convolutional autoencoders and transfer learning. Matern et al. [67] exploit the different visual artifacts left by these two manipulation techniques to expose forgeries in videos. Works done by [68] and [69] are able to detect spliced videos with DeepFake by using CNNs as feature extractor, and then creating sequences of features for each time index analyzed to classify the video as tampered or not using recurrent LSTM classifiers.

The use of RNNs is quite appealing when processing video data, since

theoretically are well suited in handling patterns in the temporal dimension of video signals. Another work besides the ones cited in the previous sentence is the one done by D'Avino et al. [8]. Following an **anomaly detection approach**, which was used by the authors in a previous work on image splicing localization [5], the authors construct an autoencoder trained on reconstructing perfectly sequences of features extracted from non-tampered video frames. These features are extracted on a block per block basis, and are essentially the same used by the authors in [47], thus residuals of high-pass filtering. The main assumption is that the internal representation of the autoencoder would implicitly represent a local system of coordinates for the manifold of the distribution of clean frames' features blocks, and therefore spliced blocks would be detected since the autoencoder would not be able to reconstruct it perfectly[1]. Measuring the reconstruction error between original and reconstructed features' blocks, an heatmap is produced which leads to splicing localization and detection. The complete pipeline followed is reported in figure 2.6.

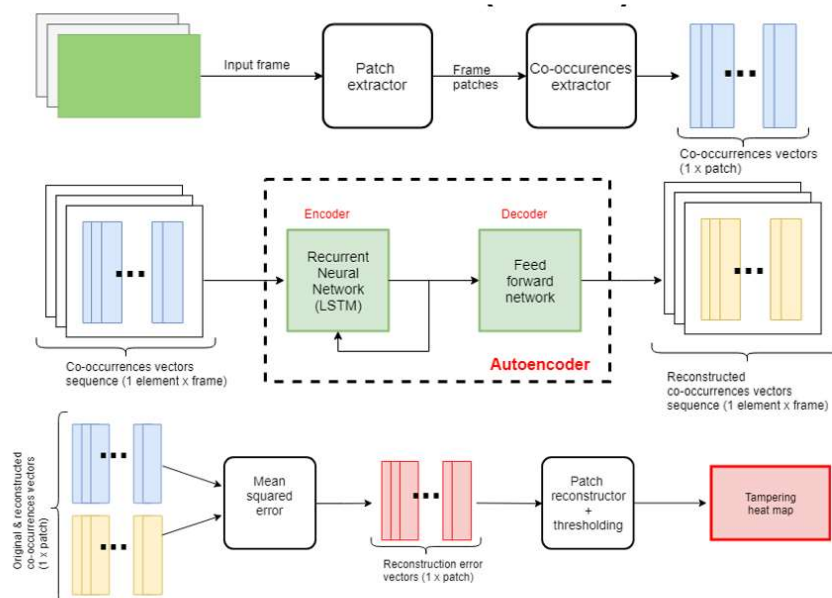

Figure 2.6: *Pipeline for the video splicing localization followed by the authors in [8].*

This method is of great interest since avoids the problem of using **labeled** datasets for the training of the autoencoder. In the procedure followed by the authors, it is possible to perform a classification task

[1]How autoencoders are able to represent a manifold for the distribution of the input data is described in the last part of the subsection dedicated to autoencoders in Chapter 1.

(decide if an area is spliced or not) while not having a labeled dataset of videos. The availability of such data is in fact an issue for multimedia forensics researchers, since it employs hours of manual work to be done for categorize each tampered video in a dataset.

The challenges posed by this problem and the way the authors faced them encouraged us to pose ourselves in the same scenario.

3
Problem formulation and proposed methodology

In this chapter we will formulate the problem of **splicing localization** using a multimedia forensics passive technique. We will then illustrate our proposed methodology, based on an approach taken from the **video anomaly detection field**, which in turns exploits autoencoders as main processing tool. Each stage of our pipeline will be then deepened in a separate section.

3.1 Formulation for splicing localization

In Chapter 2 we have described in what consists a splicing forgery and the types of forgeries that can be executed on a video signal in order to alter its content. We have furthermore described the differences between the *detection* and *localization* of splicing. Here we will briefly formulate them from a signal processing perspective introducing the notation necessary to illustrate our methodology.

Consider a digital video signal $\mathbf{V}$ of which we want to assess the integrity. This video can be considered as multidimensional tensor coming from a 3D lattice, which samples the temporal evolution of the light coming from a natural scene and which is captured by an acquisition device (a digital camera). Specifically, a video signal can be considered as a 4-dimensional tensor, where the first dimension represents **time**, the second and third **cartesian coordinates** where each element is used to represent the **light intensity** in a point (pixel) of the captured scene, and finally the last one the different **color channels** of the signal.

We have in fact previously described how the CFA allows each pixel

to capture only one component of light, and how a color image or video is obtained by the interpolation of three different signals which are relative to the components filtered by the CFA (red, green and blue, the primary colours). Most image and video coding formats however consider not these three components, but a separate **color space**, composed by the **luminance**, the simple brightness of the image/frame (the "black and white" achromatic portion of the image), and the **blue** and **red chrominances**, simply defined as the difference between the blue and red components of the image/video and its luminance content.

The reason behind such operation is that the human visual system is more sensitive to luminance variations, therefore allowing to use lower bitrates in coding for the chrominance components. In this way it is possible to reduce the final amount of information to encode while preserving the perceived quality of the object, since the luminance channel, which is the most informative, is encoded with higher quality. For multimedia researchers, this translates in the possibility of "ignoring" the chrominance channels, and to consider, for their analysis, luminance only, "dropping" the last dimension of the signal.

From now on, our video signal $\mathbf{V}$ therefore will be considered as a 3D tensor, with the dimensions having the semantic described before: $\mathbf{V}$ will be a discrete function $f(t, x, y)$, representing the temporal evolution of the luminance channel only in the cartesian coordinates x, y. As we have pointed out several times before, our video can be considered also as a sequence of frames, or images, ordered along the temporal dimension in order of acquisition.

Now, let us consider a single frame $\mathbf{F}$. We can represent the integrity of the frame by creating a **mask** $\mathbf{M}$, of the same dimensions of $\mathbf{F}$, where each pixel takes a binary value 0 or 1, such that a pixel belonging to a spliced region has value 1, and a "clean" pixel instead has value 0. An example is given in figure 3.1. The overall video mask is made of the collection in time of the masks of each frame.

For splicing in our work we refer to the **image-based attacks** described in Section 2.2, therefore to the addition of regions of video coming from a different source object. Within this context, our main goals of detecting and localizing the splicing translates in computing $\hat{\mathbf{M}}$, an estimate of the mask for each frame, where any pixel with value 1 represents a forged position (splicing localization), and therefore if the frame presents any pixel with value different from 0 is detected as spliced (splicing detection).

Figure 3.1: *Example of a frame taken from a spliced video and relative mask used to represent the spliced region. In this case the man walking is added through an editing software. The mask belongs to the dataset presented in [8].*

3.2 Proposed methodology

Our proposed methodology is based on different works in the field of **video anomaly detection** which rely on **deep learning tools**. Anomaly detection is a well known subdomain of machine learning and data mining research, whose main goal is to identify and detect **abnormal patterns** or **motions** in data that are by definition infrequent or rare [29]. Applied to video signals, it is mainly used in surveillance systems in order to detect "strange" events analyzing the motion and appearance patterns of data coming from CCTV cameras. The main issue encountered in accomplishing this task is that, imagining of tackling it as a classification problem for instance, datasets with labels for anomalous samples are hard to be found. In fact, while the "normal" class can be well characterized, the "anomalous" one comprehends a wide set of rare events or unseen objects that, for the sake of the goal, can be considered consistent, but in fact might be very different from each other.

Therefore, the objective of anomaly detection is, given a dataset of training samples containing *no* anomalies, to design or learn a feature representation that captures their behaviour, and allows to detect anomalies by measuring the **deviation** of samples from this "normal" behaviour.

Deviations can be considered as the approximation error of the projection of the sample in a geometric space, for instance, or the probability of observing the anomalous sample given a distribution modeling the "normal" ones. Thus, when dealing with video anomaly detection, to the issues which we can say are inherent to the task of anomaly detection, we have to consider the well known problems relative to the dimensionality of video signals too.

We can see therefore that there are a lot of analogies between the anomaly detection and video forensics field. The first of them can be considered the very low availability of labeled datasets; the second, the problems in handling video signals; finally, the last one the difficulties in designing a set of characteristic features to extract from the data in order to accomplish the task at hand.

It should not surprise then if deep learning techniques gained popularity in the video anomaly detection field too. The work of Kiran et al. [29] makes an exhaustive overview of the different approaches followed in the field. Two in particular, namely the **representation learning for reconstruction** and the **predictive modeling** approaches, caught our attention.

The first aims at constructing a representation for normal video samples using **autoencoders**, and detecting anomalies as samples that are poorly reconstructed. The second considers video frames as temporal patterns or time series, and therefore aims at constructing the conditional probability distribution $P(\mathbf{F}_t|\mathbf{F}_{t-1}, \dots, \mathbf{F}_0)$, so at predicting the current frame given a history of past frames. Again, deviations are measured as frames that are badly reconstructed, and in this case the main tool used

are **RNNs** such as the **LSTM** model. Combining the two approaches, the work by Chong and Tay [9] uses the **convLSTM** model described in subsection 1.2.3 for creating a **spatio-temporal autoencoder**.

It is trivial to see that if we consider as our "normal" samples **frame sequences** without splicing, and as our "anomalous" **spliced frames sequences**, we can translate the problem of splicing detection and localization as an anomaly detection one[1]. Our pipeline, which is represented in figure 3.2, can be briefly summarized into the following steps:

- we **pre-process** the incoming video signal, dividing it into smaller **sequences** of video frames. These sequences are smaller both in the spatial and time dimensions;
- we train an **autoencoder** to reconstruct sequences with **no splicings**: our aim is to make the internal representation of the autoencoder capture the local coordinates, as explained in subsection 1.2.4, of the manifold where splicing-free videos lie in the input space;
- after the training procedure has stopped, we **test** the autoencoder on **videos forged** with splicing attacks;
- on the assumption that the hidden representation of the autoencoder is well suited for clean sequences only, spliced regions of the video are detected since they are badly reconstructed: therefore, we measure the **reconstruction error** between the original spliced video and the reconstructed one, producing a **splicing heatmap**;
- finally, after producing the heatmap we binarize it to have our estimate of the mask $\hat{\mathbf{M}}$ for the sequence of frames.

Splicing localization is possible since each sequence is extracted from small portions of the video, also called **patches**, so that the original video is subdivided into spatio-temporal regions analyzed separately. For the sake of our discussion, we will divide patches into two types: **sequence patches**, that we can consider as a collection of few small frames coming from the original video[2], and which are extracted from **both** the spatial and time dimension with a process we call **spatio-temporal cropping**; and **frame patches**, which instead are extracted from the spatial dimensions only considering a single frame at a time (see figure 3.3 for an exemplification).

[1] The same idea has been used by D'Avino et al. [8], but our works differs in different aspects. First, the authors work with hand-designed features extracted from each frame, that in a second moment are organized in sequences and analyzed by a recurrent autoencoder based on the classic LSTM model. Our pipeline works directly with the pixel data of the video, and employs convolutional structures in all the architectures developed. Moreover, our pipeline includes a pre-processing operation, that we call **volume rotation**, that is absolutely novel in the field.

[2] Each sequence patch can be considered approximately as a very short and low resolution video "cropped" from the original one.

In the next sections we will describe in details each step of our pipeline, the different versions of autoencoders used along with the **training procedure** executed, and finally our testing procedure with the heatmap computation and mask estimation. Interested in how the information present in a sequence of frames could help the splicing localization procedure, we wanted to compare two different design architectures for our autoencoder:

1. a **recurrent** architecture, based on the convLSTM model and inspired by the work of Chong and Tay [9];
2. a **non-recurrent** architecture, based on convolutional neural networks and taken from part of the work done in [10], which proved to be effective in the splicing localization of attacks executed on images coming from satellites services.

Both have been trained on different versions of patches, using a **rotation procedure** which will be explained in details in the next section.

3.3 Video pre-processing

As we have explained previously, images and videos are usually encoded using the luminance-chroma color space. A common operation in multimedia forensics research, based on the motivations described in the previous section, is to work directly with the luminance component only. This operation indeed has the advantage of reducing the dimensionality of the object at hand (we consider a single color channel instead of three), while still providing enough information for the forensics analysis.

As the first step of our pipeline therefore, we convert the range of the pixels' values of the video frames from the standard 8-bit encoding of almost all video formats (from 0 to 255), to a 0-1 range. We then normalize each video by subtracting the mean computed averaging the pixels' values at each location for every frame, resulting in a final range between -1 and 1. Finally, we take the luminance channel only.

From the original video, representable as a 4-dimensional tensor $\mathbf{V} \in \mathbb{N}^{t \times H \times W \times c}$, whose elements are integer values from 0 to 255, where t represents the temporal dimension, H and W the height and width of the resolution of the video, and c the number of color channels, we then arrive to a 3D tensor $\hat{\mathbf{V}} \in \mathbb{R}^{t \times H \times W}$, whose elements are float values from -1 to 1 representing the temporal evolution of the luminance channel.

The next operation in the pipeline is the sequence patch extraction. Each patch is extracted from **cropped** regions of the video, with the cropping taking place both in the time dimension t and in the spatial dimensions H and W. This leads to the creation of a dataset of sequence patches cropped from each video, that for the sake of training our networks can be considered as independent samples taken from an uniform

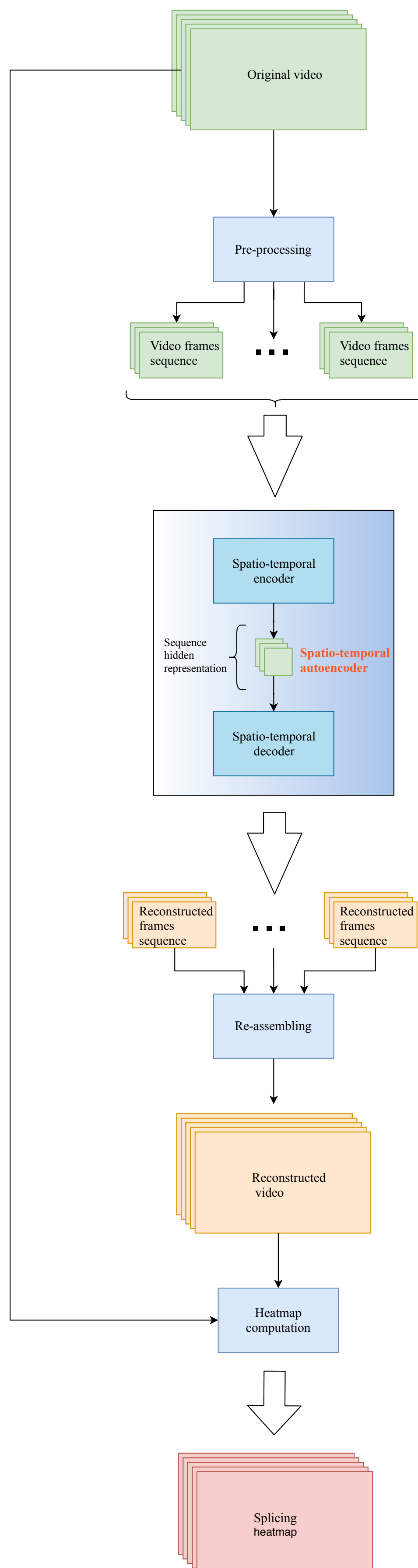

Figure 3.2: *Pipeline followed by our methodology.*

probability distribution. This procedure can be seen as an "extension" in time of the simple "cropping" procedure used for data augmentation by many deep learning practitioners (see again figure 3.3), that in our case is essential for having a fine granularity in detecting the splicing in both the temporal and spatial dimension.

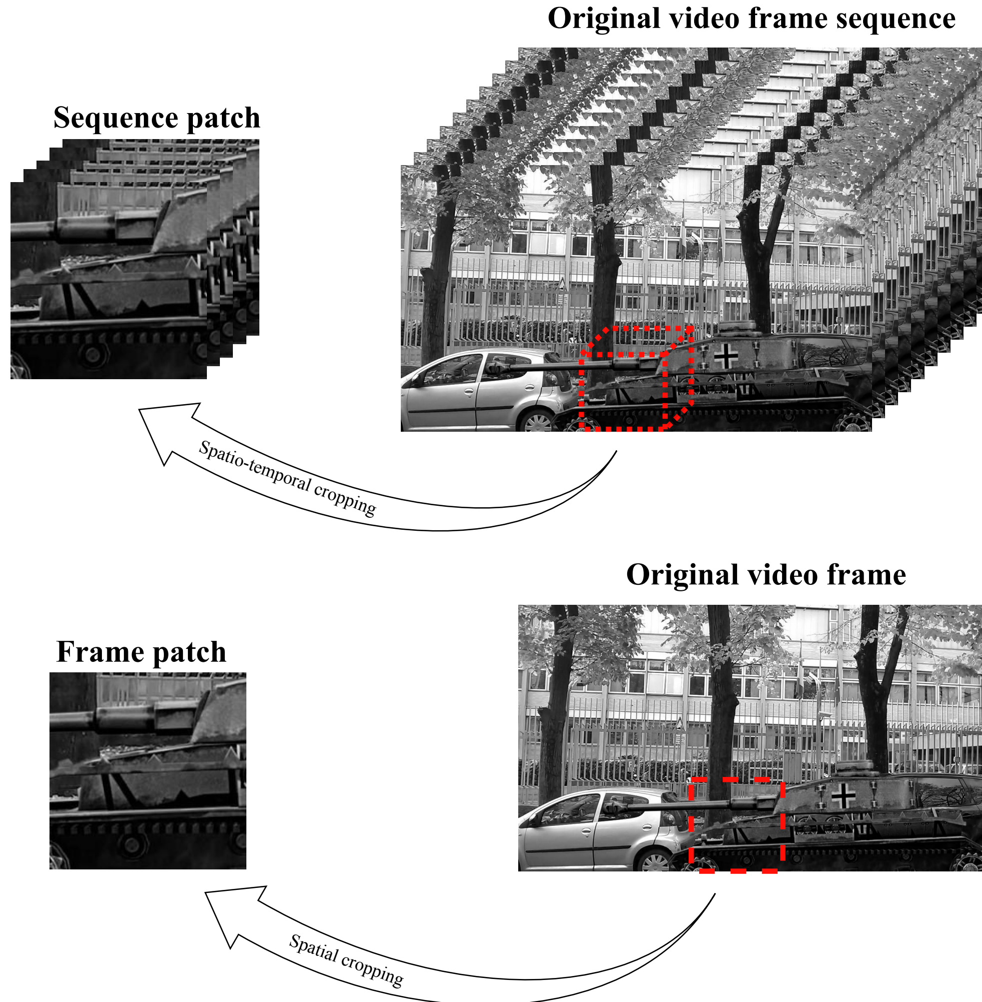

Figure 3.3: *Spatio-temporal and spatial cropping on the normalized video signal. We can see how the sequence "patch" extracted is smaller in the height, width and temporal dimension, as an "extension" in time of a normal frame patch. The frames are taken from a video of the dataset presented in [8].*

A novel aspect of our work is that sequences are not only extracted considering videos as a collection of frames. Taking a step backward, let us look at our 3D tensor $\hat{\mathbf{V}}$ as a 3D volume. This volume is nothing more than the digital signal resulting from the sampling of a multidimensional

function. Looking at it as a sequence of frames, we have what we can call a "frontal" view of the signal. This frontal view can be interpreted, as we have already done, as the temporal evolution of luminance in the points defined by a cartesian coordinate system. However, if we take this volume, and rotate it, we would observe the same evolution from a different perspective. Let us take as an example a volume $\mathbf{R}$, which we rotate along the y axis by 90°, as depicted in figure 3.4. The resulting "view" of the volume would be different from the one we are used to: instead of viewing the luminance sampled at different time instants in a cartesian point $p = (x, y)$, in the rotated volume $\tilde{\mathbf{R}}$ we would observe the **whole** temporal evolution of the luminance in a cartesian point y sampled at different positions in the x axis.

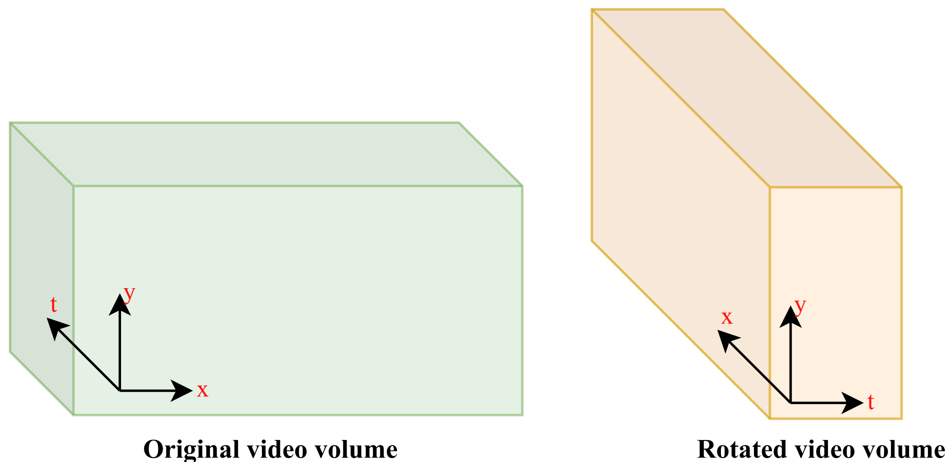

Figure 3.4: *Example of volume rotation along the y axis.*

While the overall semantic of our 3D object does not vary considerably, what really changes, accordingly to the paradigm of the "views" explained before, is the semantic of the patches of sequences given to our autoencoder. Considering our proposed methodology, this operation can provide it a more robust perspective for constructing an hidden representation of clean video sequences. Looking at the volume from three different angles, our hypothesis is that splicings which are less evident in the "classic view" of a video signal can be spotted from the rotated perspectives.

Image-based attacks indeed can be considered as a factor introducing discontinuities in the evolution of the sampled signal. Rotating the volume can give different insights for several reasons, of which one can be simply the fact that often the resolution of video signals is much higher in the spatial dimensions rather than in the temporal one. The temporal dimension can be considered as the most "sacrificed" in the encoding procedure of many video formats, due to reasons relied to the perception of motion by the human visual system. Rotating the volume therefore, from

a simple signal processing point of view, gives us a signal with higher resolution with respect to the classic "frontal" view, such that image-based attacks are easier to be noticed.

Another motivation, is that by looking at the volume from the frontal perspective only, our networks have an high chance of learning to represent the observed scene from an object-detection point of view. Interestingly indeed, the purpose for which the convLSTM model has been developed in the first place was to extend the basic LSTM model in order to have an hidden representation capturing the movement of objects in the scene[3]. Such semantic however is too specific for the splicing localization task. By rotating the volume, we are avoiding the networks to concentrate on these aspects of the scene, and forcing them to have a more general representation of the analyzed volume.

This approach is independent of the type of networks used. We can still perform our patch extraction procedure (as an example see spatio-temporal cropping executed on a rotated volume in figure 3.5), and eventually use both recurrent and non recurrent networks and compare their performances in detecting image-based attacks. In practice, the rotation procedure simply translates in **permuting** the elements of our final 3D tensor $\hat{\mathbf{V}} \in \mathbb{R}^{t \times H \times W}$, so that it results in a tensor $\tilde{\mathbf{V}} \in \mathbb{R}^{H \times t \times W}$ or $\tilde{\mathbf{V}} \in \mathbb{R}^{W \times H \times t}$, therefore rotating the volume of 90° along the x or y axis.

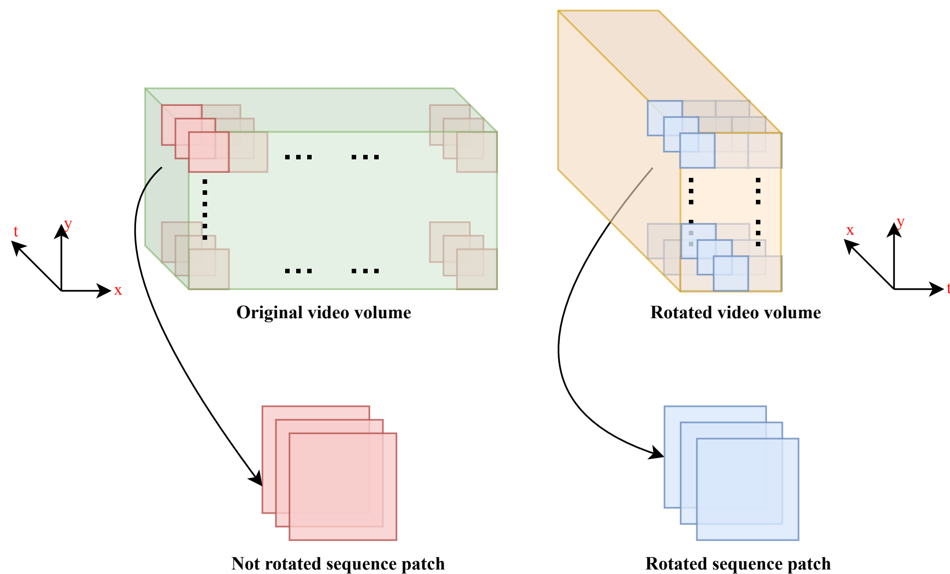

Figure 3.5: *Example of patches extraction with spatio-temporal cropping from the volume rotated by* 90° *along the* y *axis.*

[3]Specifically, the author proposed it for predicting radar maps for precipitation nowcasting tasks.

3.4 Autoencoders architectures

As we have stated previously, part of our work consisted in the comparison between a recurrent and a non-recurrent architecture for the autoencoder employed in our pipeline. The reason lies in the curiosity of understanding if models that process data with a dimension that represents the sequentiality of the process generating it, might have more insight in the localization of splicing attacks.

Our recurrent autoencoder is inspired by the work done by Chong and Thay in [9], and in figure 3.6 we have represented both schematically. The architecture by Chong and Thay takes as input frame sequences of 10 frames 227×227 pixels wide, and it can be divided into three modules:

1. a **spatial encoder**, which is used to reduce the dimensionality of each frame *singularly*, and is composed by two convolutional layers;

2. a **temporal autoencoder**, composed by three layers based on the convLSTM model, which constructs the **hidden representation** of the overall sequence of frames;

3. a **spatial decoder**, which takes the hidden representation in output from the temporal autoencoder and outputs the **reconstructed frame sequence** with the same dimensions of the input.

All the parameters of the convolution operation are described in the picture (number of filters and dimension of their kernels, eventual stride). Please notice that all layers are activated using an **hyperbolic tangent**, so that their output still lies in the range -1 to 1 of the input sequence, and that no pooling has been used. Our architecture mirrors the one developed by the authors, with two major differences:

- between each layer, we employ a layer of **batch normalization** [70]: it is indeed a common practice between deep learning practitioners to add it in convolutional chains, since it allows to control the dynamic of the inputs given to intermediate layers;

- the sequences gave as input to the autoencoder are extracted with the spatio-temporal cropping procedure described in the previous section, while the authors resized full videos to the dimension specified in figure 3.6.

The use of batch normalization and the dimension of the patches, in particular the length of the sequences, has been decided in a specific stage of **model selection** of our work explained in Chapter 4.

For what concerns our non-recurrent autoencoder, it is mainly inspired by the work of Yarlagadda et al. in [10]. Here the authors worked with images coming from satellite services, and used a convolutional autoencoder as a feature extractor for a successive classification task in a splicing localization context.

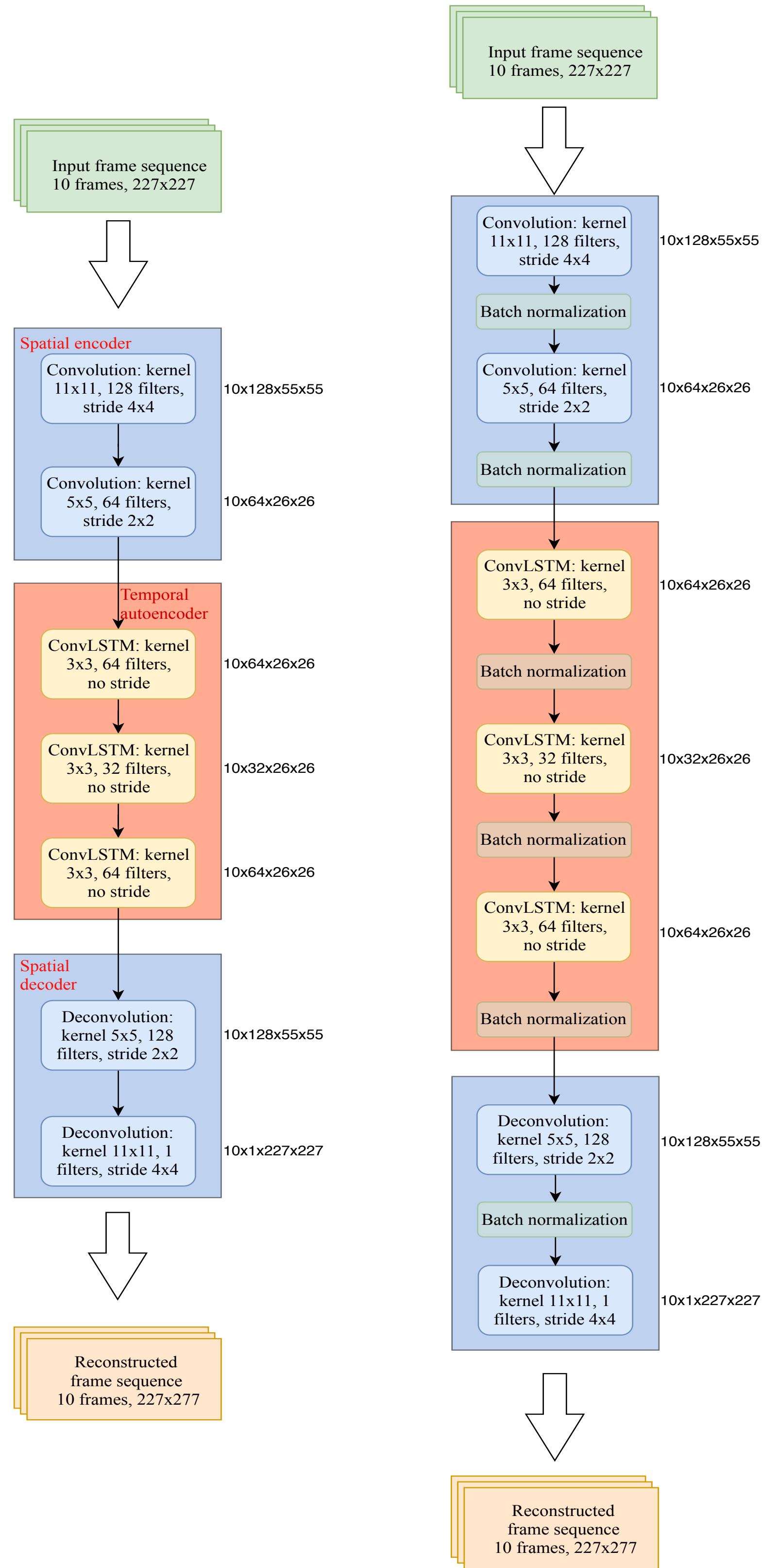

Figure 3.6: *The spatio-temporal autoencoder proposed in [9] (on the left) and our basic recurrent architecture (on the right). On the right of each layer we can see the dimensionality of the tensor in output.*

In our work instead we rely on the ability of the autoencoder to capture the manifold of splicing-free frames, therefore we simply use the architecture of the convolutional autoencoder proposed by the authors for the feature extraction process, depicted in figure 3.7, for our anomaly detection task.

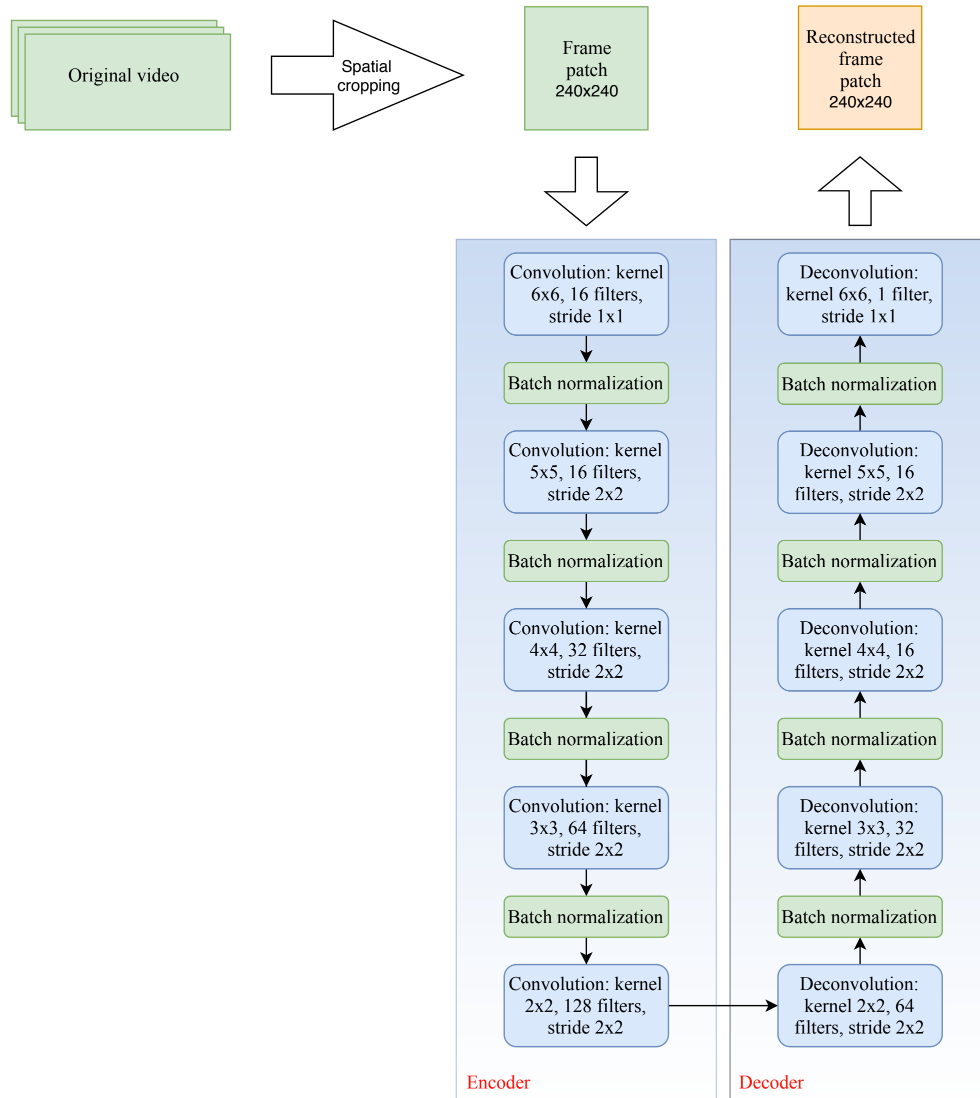

Figure 3.7: *The convolutional autoencoder proposed in [10]. Please notice that also in this case no pooling is used, and that the activation of each layer is linear.*

The autoencoder works with **frame patches** as input, 240×240 pixels' wide. Hence, differently from the recurrent autoencoder previously described, when training this network we perform a simple spatial cropping and reconstruct the patches extracted from each frame singularly. This operation still allows us to localize the splicing in time, since we work on a frame per frame basis, but also to compare if the presence of a sequentiality dimension, being time or space depending on the rotation

of the volume, could give an advantage in localizing image-based attacks.

The autoencoders described so far can be defined as our "basic" versions. Regardless of the architecture used, both of them have been trained to reconstruct the input given using as a loss function the **mean squared error** computed between the reconstructed and the input sequence/frame patch. The mean squared error (MSE) can be defined statistically as a measure of the performance of a **predictor**. Specifically, taking a vector of N predictions $\hat{\mathbf{x}}$ and a vector of N observations $\mathbf{x}$ of the predicted variable x, the MSE is defined as

$$\mathbf{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{\mathbf{x}}_i - \mathbf{x}_i)^2. \tag{3.1}$$

Algebrically, we can view the MSE as the **norm of the distance** between the two vectors in an Euclidean space where they are defined ($\mathbb{R}^N$ in this case). In both interpretations, it gives us a measure of the "goodness" of the approximation of the "real" vector $\mathbf{x}$ using $\hat{\mathbf{x}}$. We can easily extend this measure for multidimensional tensors by simply using the correct norm for the space where they are defined. In our case, we will use it to verify the correctness of the reconstruction of the input given to our autoencoders, therefore the sequence patches for our recurrent autoencoder and the frame patches for our non-recurrent one.

However, our patches are defined over of an high dimensionality Euclidean space. Assuming that the two "classes" of spliced and clean patches are distinguishable, it would be easier to observe their separation in a lower dimensional space[4]. Since the loss is the only means for the autoencoders to learn the differences between the original and reconstructed patches, computing the MSE in an high dimensional space might not give the networks enough clue to construct an effective hidden representation. For this reason, we have devised another training procedure, where instead of computing the loss between the actual and the reconstructed input, we do so between the **hidden representations** of the original and of the reconstructed patch, as depicted in figure 3.8.

Using this loss function, which we can call $\mathcal{L}_2$, we trained different autoencoders, where we have tried to balance the reconstruction errors measured by the MSE computed for both the hidden representation and the reconstructed patch. In fact, defining instead the MSE between the input and reconstructed patch as $\mathcal{L}_1$, we can use as performance measure during our training the overall loss $\mathcal{L}_{enc}$[5] equal to

$$\mathcal{L}_{enc} = \alpha\mathcal{L}_1 + \beta\mathcal{L}_2, \tag{3.2}$$

where α and β are hyperparameters which could be determined during a stage of model selection.

[4]This is a well known problem in machine learning theory defined as **curse of dimensionality**.

[5]The *enc* subscript refers to the name given to the autoencoders trained with this loss function, encoding loss autoencoders indeed, as we explain in the following.

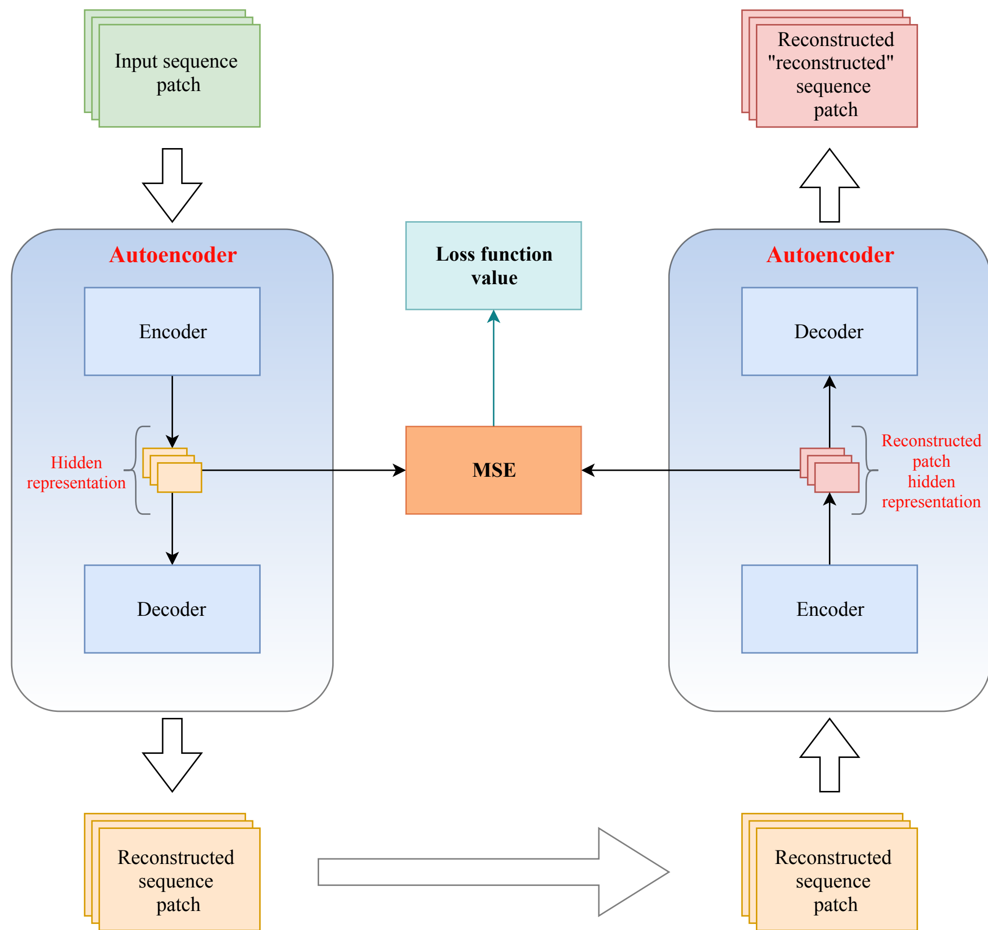

Figure 3.8: *In order to compute the loss between the hidden representations, we first make the autoencoder reconstruct the input patch, and then reconstruct the "reconstructed" patch another time. We then take the hidden representation of the two patches and compute the MSE between them. In this figure we use as an example the reconstruction of a sequence patch by our recurrent autoencoder.*

In our work we have simply tested three pairs of values for them, as we will explain in the following chapter, in order to evaluate the impact of both MSEs during the training of our networks. We will refer to this group of autoencoders, both recurrent and not and working with rotated and not rotated volumes, as **encoding loss autoencoders**, for the use of the MSE computed on the hidden representation during their training.

Based on the experiments conduced for this last types of networks, as a final step of our work we trained another model of autoencoder. Using again as loss function during training the MSE computed on the hidden representation, we insert as an additional term a **regularization term**. Inspired by the work of Cozzolino and Verdoliva [5], we insert a term trying to mimic the class intra-variance used by the authors. After some reflections, our choice fell on the use of the **mean of the energy** of the patch, computed as the mean of the squared elements of the reconstructed patch.

Our assumption is that this regularization term would constrain the autoencoder to reproduce content with "smoother" transitions. Considering image-based attacks as a source of discontinuities in the signal, such a property would therefore limit the autoencoder in reproducing spliced regions of the video with an high overall quality, leading to their localization following our pipeline.

This choice revealed indeed to be the most effective approach for splicing detection, as we will highlight in the following chapter. As we did for the encoding loss autoencoders, defining the regularization term as $\mathcal{L}_3$, we tried to balance it with the term representing the MSE computed on the hidden representations according to the following formula

$$\mathcal{L}_{reg} = \alpha\mathcal{L}_2 + \beta\mathcal{L}_3. \tag{3.3}$$

From now on, we will refer to the autoencoders trained in such way, that are both recurrent and non-recurrent, and work with both rotated and non-rotated volumes, as **regularized autoencoders**.

Please note that for both the encoding loss and the regularized autoencoders, the procedure of splicing localization does not change. Both typologies of networks return as output the reconstructed input patch: what changes is only the procedure of training in the form of the loss function used. Therefore, both types allow us to follow the pipeline defined before, of which only the last step, the **heatmap and mask computation**, needs to be described.

3.5 Heatmap and mask computation

The last step of our pipeline comprehends the computation of an **heatmap**, through which we aim at constructing an estimate of the **mask** $\hat{\mathbf{M}}$ to localize the spliced regions of the video under analysis.

After having reconstructed all the patches extracted from the original test video, our procedure simply consists in the following steps (represented in figure 3.9):

1. compute the **squared difference** between the elements of the original and reconstructed patches, creating what we can call a patch of **quadratic residuals**;

2. from the quadratic residuals, apply a multidimensional **mean filter**, of the following dimensions: for the reconstructed sequence patches, a 3D tensor of dimension t and 8×8 pixels wide, with t being the sequence length (10 frames in our case); for the frame patches, a simple window 8×8 pixels wide. This leads to the creation of patches of MSE between the original and reconstructed ones;

3. convert the MSE patches' values to a 0-255 range, obtaining an **heatmap** for each patch;

4. binarize the heatmap by imposing a threshold above which the pixels are considered **spliced**, obtaining a **mask estimate** for each patch.

The use of the mean filter allows us to localize very small splicings in the spatial dimension, while keeping into account all the frames of the sequence. The performance of our pipeline can then be evaluated in two ways:

1. by reconstructing a mask for the overall video, and comparing the estimated mask with the original;

2. by extracting patches from the mask video and then comparing the estimated masks for each patch.

The performance of the networks are then evaluated by changing the threshold computed at point 4 and creating **Receiver Operative Characteristic (ROC)** curves, as we will explain in the following chapter. Figure 3.10 shows a representation of the mask estimation procedure.

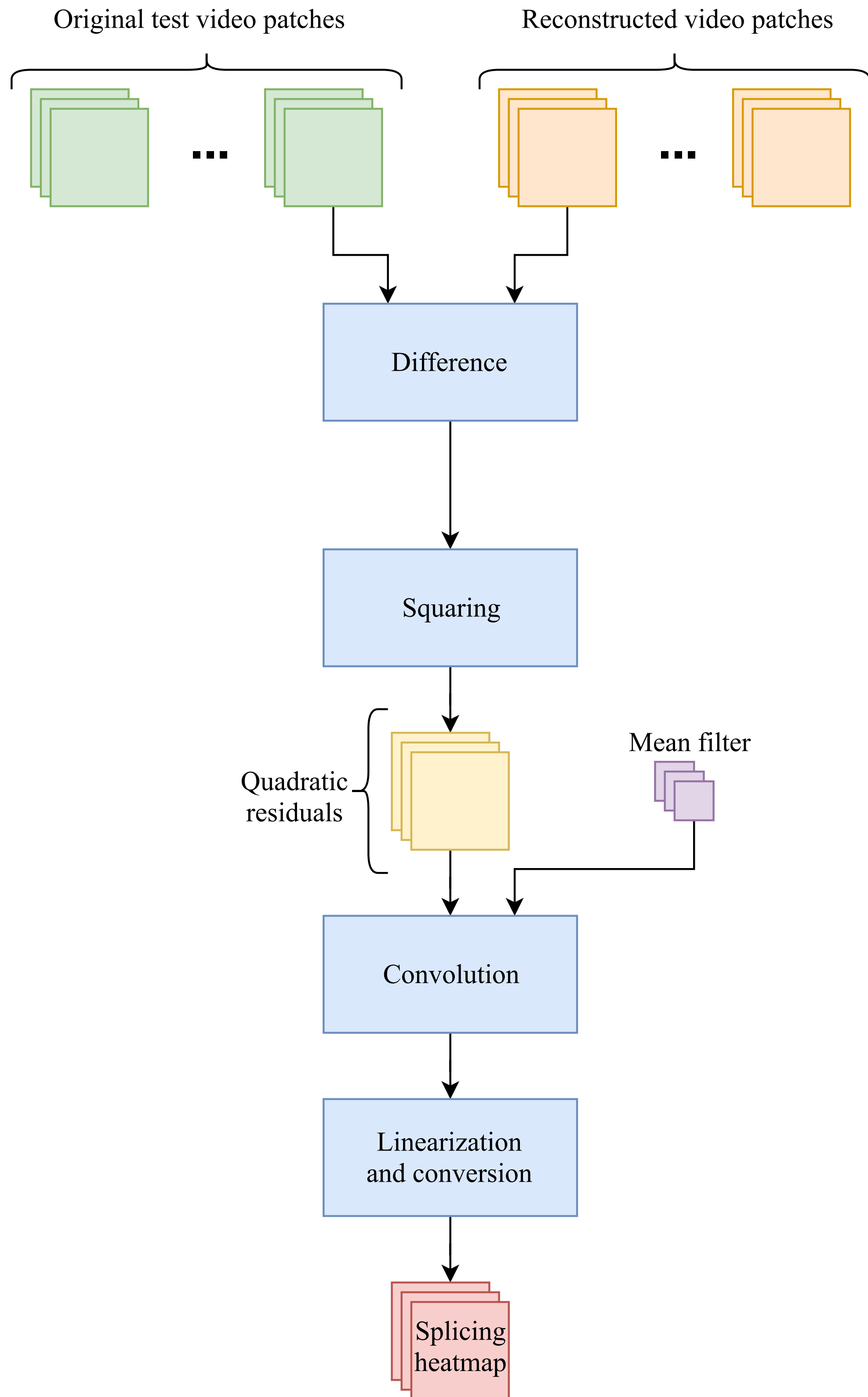

Figure 3.9: *Heatmap computation procedure executed on sequence patches. The process is almost identical for frame patches, changing only the size of the mean filter.*

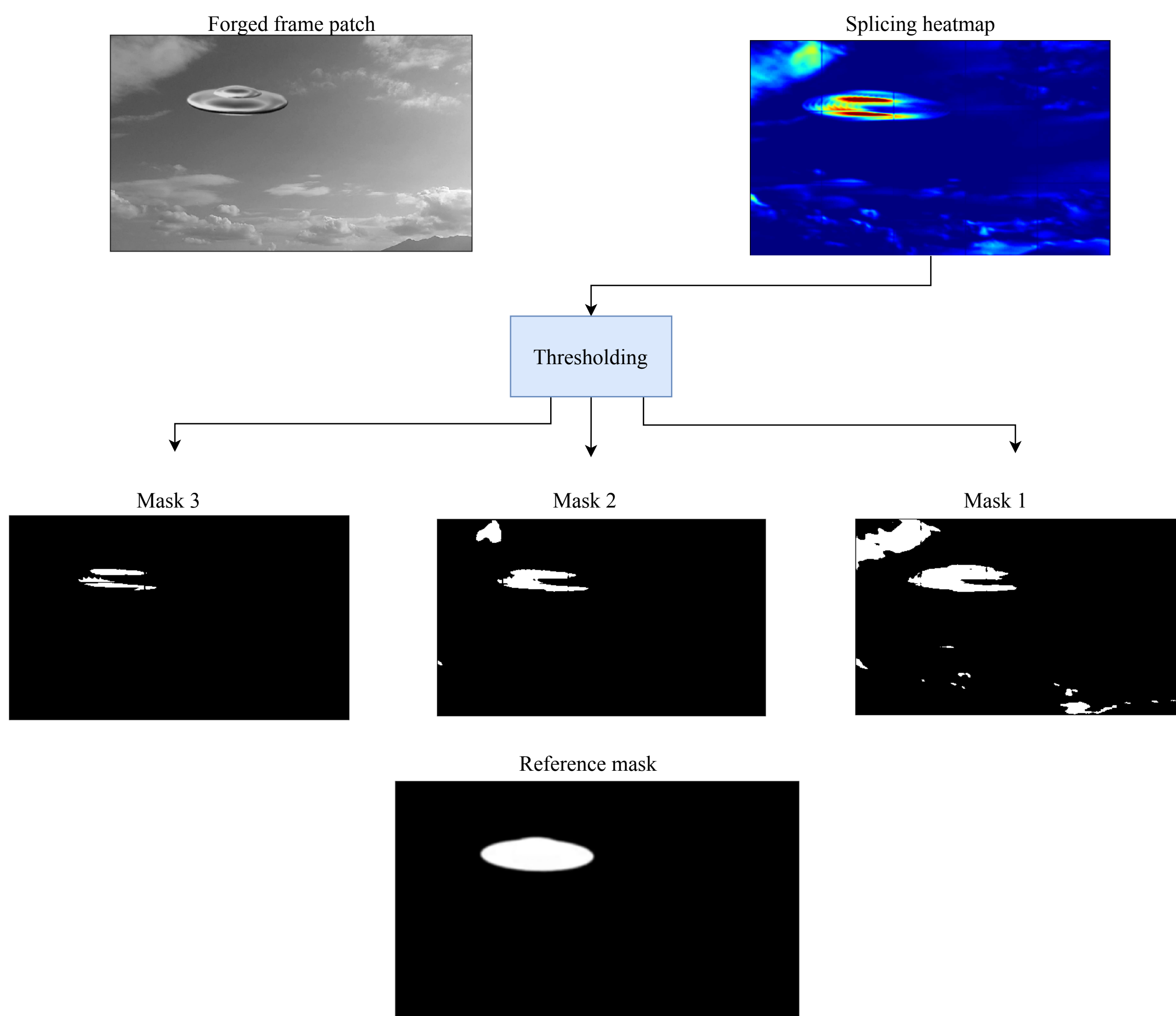

Figure 3.10: *Thresholding of the heatmap in order to create different mask estimates. Each of them are then confronted with the reference mask to compute ROC curves as explained in Chapter 4.*

4 Experimental results

The present chapter is going to illustrate the tests carried out to validate our proposed methodology. We will begin describing the **dataset** used for our experiments. We will then outline the training and validation procedure used for our autoencoder models, closing with a detailed report on the tests executed. In particular, a section will be devoted to explain the **preliminary experiments** used for determining some of the **hyperparameters** of our models and methodology, and a section dedicated to the description of our **final tests** will close the chapter.

4.1 Dataset description

Several times we have pointed out how the lack of labeled datasets is one of the major difficulties encountered by multimedia forensics researchers. Fortunately, D'Avino et al. [8] made publicly available the dataset of videos used for their experiments. In order to have a reference and performing our tests under known and controlled conditions, our choice fell over this set of videos too.

The overall dataset consists in two separate groups of **10 videos** each. The first one, namely the **clean set**, is made of videos acquired using nine different smartphones. All of them have the same resolution of 720×1280 pixels and a frame rate of 30 frames per second, but differs for their overall length, which is around ten to twenty seconds. The second group instead, namely the **forged** one, is constituted by the same 10 videos, which however have been altered through **image-based attacks** created using Adobe After Effects CC®. In particular, using **chroma-key composition** (see figure 4.1 for an example), the authors have spliced the original videos using as external sources **green-screen acquired** clips

available on YouTube under the Creative Common License. Therefore, for each of the videos of the clean set, a spliced counterpart exists, together with a **mask** indicating the spliced regions (which we have shown previously in Chapter 3, figure 3.1).

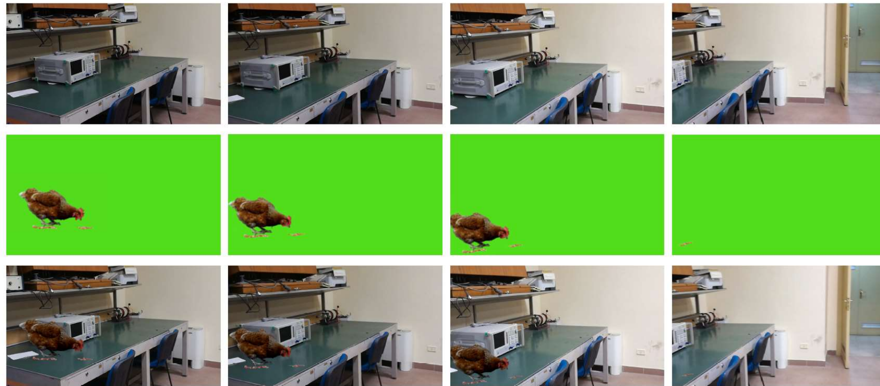

Figure 4.1: *Illustration of the splicing process executed by the authors in [8]. In the top row we can see frames of the original video, in the middle the green-screen clip used for forging, and in the bottom the final spliced video. Image taken from [8].*

All files are available in the **Audio Video Interleave (AVI)** format, and are encoded using the **Advanced Coding Video (AVC)** codec according to the High Profile of the **H.264** standard.

4.2 Models training and validation

In the previous chapter we have described the different typologies of autoencoders devised in this work and the motivations behind their use. In the following we will describe the procedure used for their **training**.

First of all, the primary goal of our approach is the creation through the autoencoders of an effective hidden representation of splicing-free patches. It is immediate therefore that the training of our networks uses the **clean videos set** only. We take each element of the group, normalize it, and then crop it individually according to the type of autoencoder trained (recurrent or not). The patches, as we have pointed out previously, can be considered as independent samples taken from an uniform distribution: therefore, after having processed all the videos, we split them in a **training** and in a **validation** set. Specifically, we insert in the training set the patches extracted from the first seven videos, and in the validation set the patches coming from the last three. The patches of each set are then randomly shuffled before training begins. In this way, we are avoiding to sequentially show the autoencoders patches coming from the same source, ideally speeding up the learning process, and at

the same time avoiding to overfit their hidden representation with the content of the videos from the training set.

Training is then carried out using a variant of the stochastic gradient descent algorithm named **Adam** [34], arranging the patches obtained in **mini-batches** of **16** elements each. The deep learning framework used for our experiments, based on the programming language **Python**, is **Keras** [71], with the hyperparameters chosen for our optimizer being:

- **learning rate** equal to 0.001;

- **exponential decay rates** for moment estimation $\beta_1 = 0.9$ and $\beta_2 = 0.999$;

- $\epsilon = 10^{-7}$.

Training has been executed for a maximum of **50** epochs, early stopping the procedure if the reconstruction error on the validation set did not change after **10** consecutive epochs. We then kept the best model according to the performances on the validation set.

As for the **loss functions** used during the process, we have already illustrated them in the previous chapter. We also specified how each of them characterizes one of the three families of autoencoders developed.

We will recap them briefly here. Recalling that the MSE between the original and reconstructed input is defined as $\mathcal{L}_1$, and the MSE between the hidden representations of the original and reconstructed patches as $\mathcal{L}_2$:

- for our **basic** autoencoders, the loss function is simply $\mathcal{L}_1$;

- for our **encoding loss** autoencoders, the loss function $\mathcal{L}_{enc}$ is defined in equation 3.2 as $\mathcal{L}_{enc} = \alpha\mathcal{L}_1 + \beta\mathcal{L}_2$;

- for our **regularized** autoencoders, the loss function $\mathcal{L}_{reg}$ is defined in equation 3.3 as $\mathcal{L}_{reg} = \alpha\mathcal{L}_2 + \beta\mathcal{L}_3$, with $\mathcal{L}_3$ being the regularization term described in Chapter 3.

The parameters α and β have not been tuned, but simply tested with three sets of values to observe their effects on the performances of the last two families of autoencoders. The three set of values chosen are 0.25 and 0.75, 0.5 and 0.5, and 0.75 and 0.25.

Other significant choices for the positive outcome of our experiments regarded for instance the **length** of the sequence patches processed by our recurrent autoencoders, the way the elements are extracted in the temporal dimension, and the use of batch normalization between their layers. All of them have been made during a specific set of **preliminary experiments**, to be explained later in a following section.

4.3 Evaluation metrics

In the previous section we have already described how we evaluated the performance of our networks during training. By simply measuring the reconstruction error on the patches extracted from the clean video set, we are able to stop the procedure and then proceed to **testing**.

The **overall test** performances of the trained networks instead have been evaluated by reconstructing the spliced videos and measuring the reconstruction error as described in the last section of Chapter 3. As for the training, the reconstruction error considered has been the MSE, from which a splicing heatmap is constructed through linearization and conversion into an 8-bit encoding scale of the pixels' values. We finally obtain an estimate for the masks of each video by imposing a threshold above which the pixels are considered spliced.

We can create a mask either for each patch reconstructed, or, by reassembling the whole video and then creating the heatmap, for the whole reconstructed video. In any case, by varying the threshold and comparing our estimates with the reference masks, we can measure the number of pixels which are classified as spliced while being really spliced, or **true positives**, and the number of pixels which are classified as spliced while in reality not being so, also defined as **false positives** (see figure 4.2 for a representation). These two numbers defines the **statistical measures** of:

- **True Positive Rate (TPR)** $= \frac{\text{\# of true positives}}{\text{total \# of samples classified}}$;
- **False Positive Rate (FPR)** $= \frac{\text{\# of false positives}}{\text{total \# of samples classified}}$;

By plotting the TPR and FPR as a function of the threshold, we obtain what is called a **Receiver Operative Characteristics (ROC)** curve. For each of our experiments, we plotted the ROC curves for all the test videos, computing altogether the **Area Under the Curve (AUC)**, which is usually adopted as a summary measure of the overall performance of the algorithm in the task.

We can roughly view it as the probability of our networks of correctly assigning a spliced pixel the positive value, and it gives an useful metric for comparing all the different solution proposed in our work.

Now we will start describing our initial set of preliminary experiments.

4.4 Preliminary experiments

After the initial phase of delineation of our methodology, the very first step of our work consisted in a preliminary test of the **basic** architectures described in Chapter 3. In particular, we wanted not only to have an early response on whether we chose our networks correctly, but we needed some clues in deciding two fundamental aspects of our recurrent autoencoders.

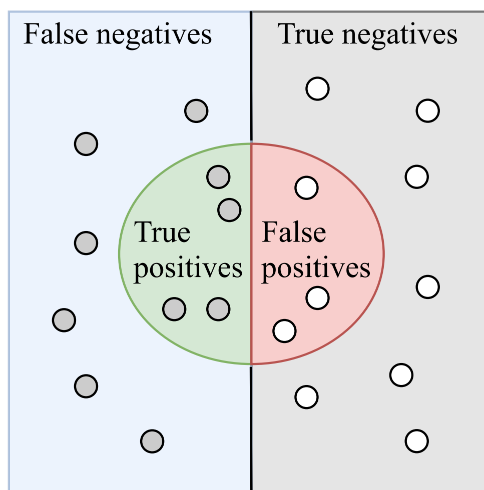

Figure 4.2: *Illustration of the division of samples in the feature space into true positives and false positives according to a binary classification algorithm. Samples inside the circle are classified as positives, those outside as negatives.*

These aspects are:

- the **length** of the sequence patches given as input;
- the **modality** of extraction of the patches in the temporal dimension.

The length of the sequence can be considered almost directly proportional to the depth of the "view" of the recurrent network in the temporal or spatial dimension. Using longer sequences therefore would, at least in theory, provide our autoencoder more information, or context, in recreating the patch in input.

All our experiments have been executed on an Intel Xeon E5-2687W with 48 Cores, having 252 GiB of RAM and using CUDA accelerated computations when possible on a TITAN V GPU with 12 GiB of RAM. Despite the good amount of resources for our computations, the fact of working with multidimensional tensors forbade us to work with very long sequences, which could not be easily loaded into the GPU's memory. For this reason, the maximum number of elements in the sequence has been set to **20**.

However, using longer sequences is not the only way to provide the network with a more informative input. Another strategy consists simply in not taking elements sequentially, but with some **dilation** (e.g., 1 element every 2 or 3) in the dimension of interest. Such an approach should be used carefully, since it might result in the loss of some information: for instance, taking elements with dilation when the volume is rotated would imply to "skip" some pixels of the frame, while in temporal dimension would imply to ignore some frames entirely.

In the light of this, we studied the behaviour of our networks changing the length of the sequence, while taking elements in the **temporal** dimension only (so with no volume rotation) with different dilation ratios too. Our first test thus forecast the training of different network prototypes, trained and validated with the videos of the clean set divided as explained before in training and validation set and tested on those of the forged group. However, for the sake of brevity, we report here only **four** of them, which are:

1. a first prototype which has been trained on patches extracted from a **single** region of interest for each video, of dimensions $10 \times 227 \times 227$;

2. a second prototype trained on patches extracted from 4 regions of each video, again of size $10 \times 227 \times 227$ but with the frames in the temporal axis selected with a dilation ratio of 1/3, meaning that a frame is selected every 3 (so that a sequence of 10 frames is extracted from a volume of 30);

3. a third prototype which was also trained on patches extracted from 4 regions of each video, but without dilation;

4. a final prototype trained on patches cropped from 2 regions, but with size $20 \times 227 \times 227$, and same dilation ratio of the second prototype (therefore a sequence of 20 frames is extracted from a volume of 60[1]).

Each of these prototypes has then been tested on the whole set of 10 forged videos, but extracting patches from a single region of interest. The reason of such procedure lies simply in the time efficiency of training and testing the prototypes on reduced portions of our dataset. Each reconstructed patch has been then reassembled into the original video, from which we obtained the splicing heatmap and finally the mask estimation by varying the threshold and obtaining the different ROC curves for each of the test clips.

The values of the AUC obtained by each of the prototypes is reported in figure 4.3. As we can see, the overall performance of the networks do not show relevant differences, independently of the sequence length or of the dilation ratio used. On one hand, this motivated us in keeping the size of the patches as the one originally used by the authors in [9], so $10 \times 227 \times 227$, and on the other suggested us that we could work in the temporal dimension using dilated sequences without loss in performances, therefore easening the overall burden on the machine used for our computations.

As a second experiment, we considered the insertion of **batch normalization** [70] layers in our recurrent architecture. The benefits of

[1]Please note that with a framerate of 30 frames per second, these volumes of 30 and 60 frames coincided with lengths of 1 and 2 seconds respectively.

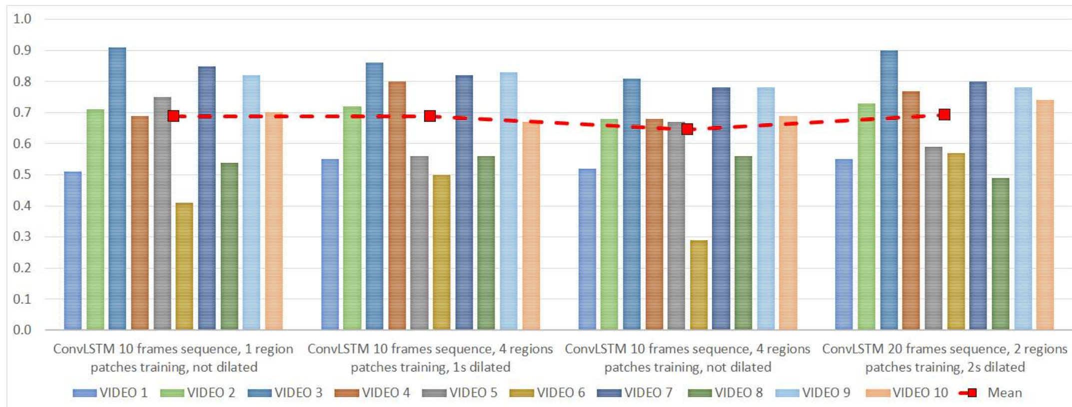

Figure 4.3: *AUC values for the ROC curves of the first four prototypes.*

using batch normalization are well known in the deep learning field, especially for what concerns the mitigation of the phenomenon noticed as **covariance shift**, and the independence from the distribution of the input values gained by each layer in their feature learning process. This last property is particularly desirable for our networks, in order to have an hidden representation which is less conditioned by the pixel content of the videos seen during training.

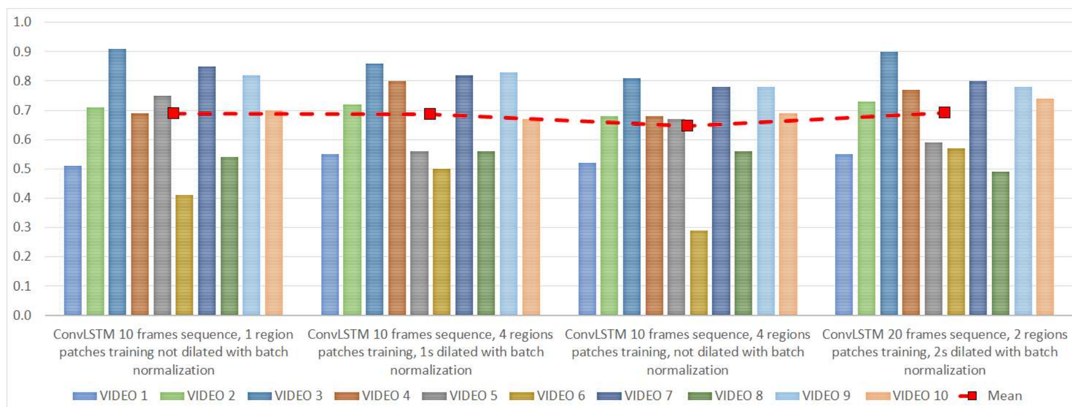

Figure 4.4: *AUC values for the ROC curves of the prototypes with batch normalization.*

In figure 4.4 we reported the results obtained testing again four prototypes, with the same parameters of the first test regarding the sequence length and the dilation, but inserting batch normalization between each layer of the architecture proposed in [9]. This experiment led to the design of our recurrent autoencoder represented in figure 3.6 of Chapter 3.

Even though we gained no sensible improvement in the overall performances of our networks, for the considerations made before for the rest of our experiments we preferred to maintain our recurrent architecture with batch normalization.

Finally, as the last preliminary experiment we tested our **volume rotation** procedure. In doing so, we used our batch normalized recurrent autoencoder, using as dimension for the patches $10 \times 227 \times 227$, as individuated in the first experiment, and developed two prototypes which differs from the fact that:

- in one case the autoencoder during training returns the patch **re-rotated**, so that the MSE is computed as if the patch has not been rotated at all;
- in the other, the autoencoder returns the patch without re-rotating it, so that the MSE is computed from the "rotated" view explained in Chapter 3.

The reason in such operation relies on the fact that, even though in the first case the autoencoder would still have an hidden representation which is based on the rotated view of the volume, the loss, and therefore the information provided to the network, would still be "encoded" in the classic perspective of the volume. Being interested in seeing if the network would still be able to localize splicing even if trained in such a way, we therefore trained four prototypes, two rotating the volume along the x axis and two along the y axis, according to the casuistry illustrated above.

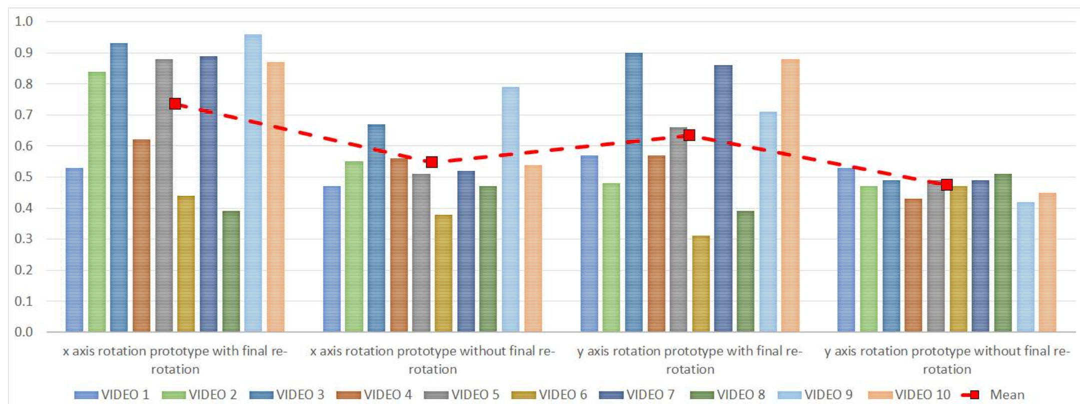

Figure 4.5: *AUC values for the ROC curves of the prototypes trained with volume rotation.*

As we can see in figure 4.5, the performances worsens with the only exception of the autoencoder trained with volume re-rotation along the x axis. This experiment therefore convinced us in applying the re-rotation of the volume even for the rotation along the y axis.

4.5 Main experiments

After the first stage of preliminary experiments, needed to set some details of the procedure and of the architecture of our networks, we have then proceeded in executing our main experiments. Basically, we have performed a group of tests for each of the families of autoencoders developed, with an experiment for each rotation of the volume. Moreover, for the two families of encoding loss and regularized autoencoders, we executed a test for each pair of values of parameters of the loss function proposed in the second section. Within this setup, every network has been trained on patches extracted from **all the regions** of the videos of the clean set, and tested on patches extracted from **all the regions** of the videos of the forged set. Masks have been estimated for each reconstructed patch, and therefore the ROC curves have been computed considering the total set of masks estimated by our networks for each singular video. For what concerns our recurrent autoencoder, the size of the patches is the same decided during the preliminary stage, therefore $10 \times 227 \times 227$, while the dilation ratio may change depending on the rotation of the volume. Whenever it is necessary, we will specify the different choice of this parameter.

4.5.1 Basic autoencoders

Concerning our "basic" autoencoders, the patches for our recurrent architecture have been extracted, for the non-rotated volume, with a dilation ratio of 1 frame every 2 in the temporal dimension. For what regards the networks working on rotated volumes instead, we preferred to not use any dilation ratio in any of the dimensions. Finally, in the case of our non-recurrent autoencoder, no dilation ratio has been applied in any dimension: we simply take one element of the volume at a time, in the order gave by the axis on which the sequence is considered ordered. The AUC values of the ROC curves scored by each type of network working on the different rotations of the volumes are depicted in figure 4.6.

As we can see, the mean value of the different AUC are only slightly larger than 0.5, suggesting that our networks are not performing better than random guessing. Among the videos, there are some exceptions, with values of the areas greater than 0.8. In any case, surely the most performant network for this group of tests revealed to be our recurrent autoencoder working on the x axis rotated volume, with a mean AUC value of 0.7. However, we must notice that this number is biased by some very good performances of the network on singular videos (specifically, the video number 7), while great part of the scores on the other clips of the dataset are around or below 0.5 (see figure 4.7 for an example).

Four videos in particular seems to be critical for all the networks tested, which are the videos number 1, 4, 6 and 8. The first and the last one share the fact that the spliced regions have a texture which is quite

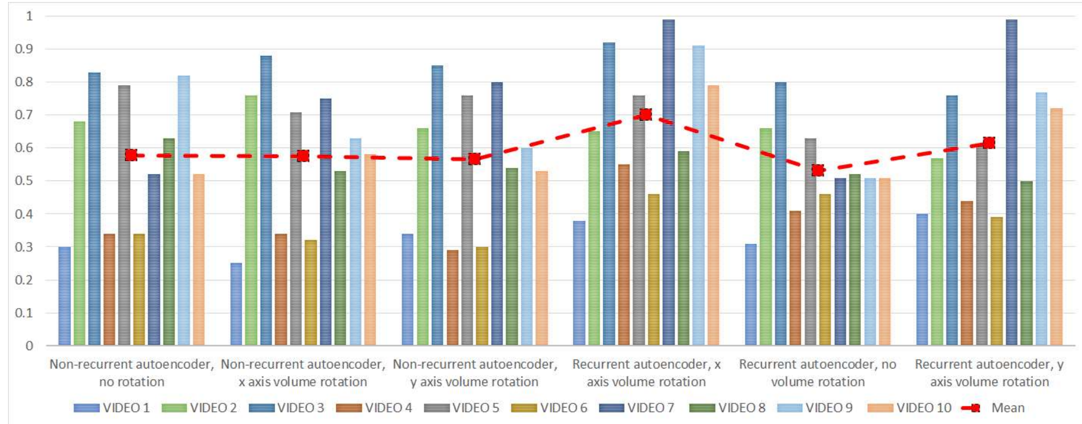

Figure 4.6: *AUC values for the ROC curves of the basic autoencoders, both recurrent and not.*

similar to the one of the non-spliced regions (see figure 4.8 for a clarification). Therefore, they might "trick" the networks, which reconstruct them, in spite of the splicing, with an overall good quality. Regarding the videos number 4 and 6, their splicings, while still quite evident, might nevertheless have some components that are mistakable as clean. The opposite reasoning might indeed be formulated for the very good results obtained by the networks on video number 7, where the splicing stands out quite evidently from the rest of the content of the video.

Therefore, our basic networks ability of spotting spliced attacks seems to be too dependent on the type of content of the spliced regions.

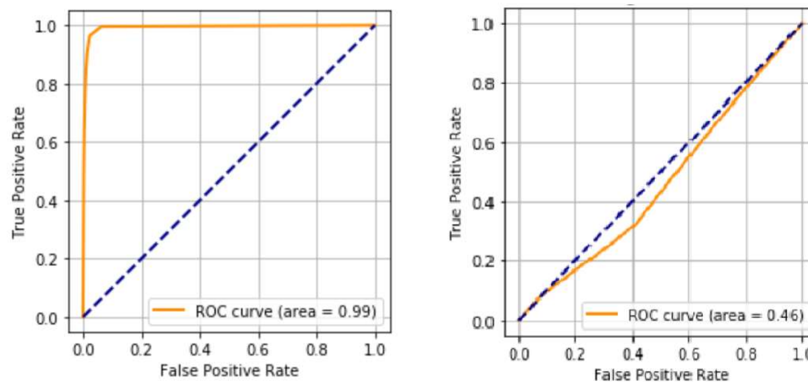

Figure 4.7: *ROC curves scored by the recurrent autoencoder working on the x axis rotated volume for the video number 7 (left) and the video number 6 (right).*

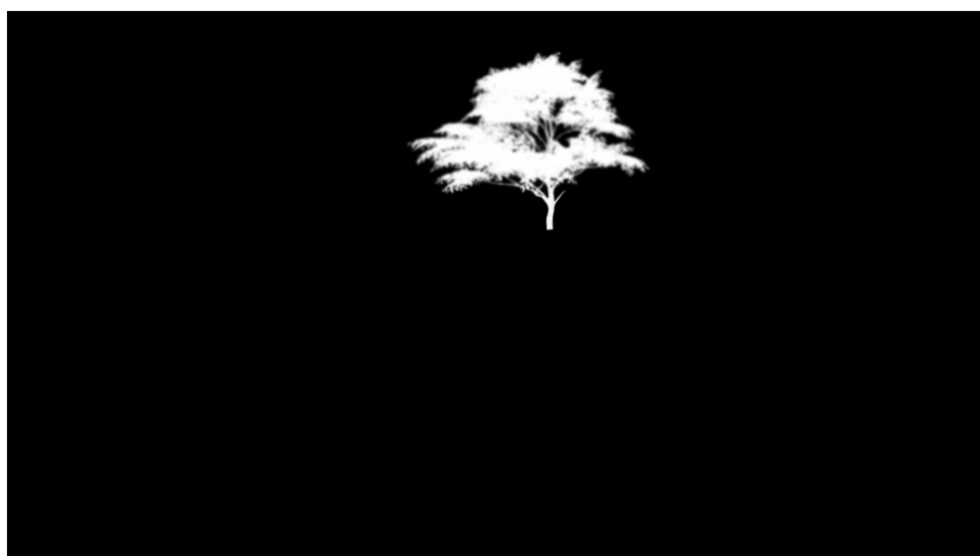

Figure 4.8: *Forged version of video number 8 and relative mask representing the spliced region. As we can see, the texture of the spliced area (a tree among other trees) is almost identical.*

4.5.2 Encoding loss autoencoders

For our second family of autoencoders, as we have previously said, we have executed a test for each of the three pair of values used as the parameters α and β of the training loss function, which again are $0.25 - 0.75$, $0.5 - 0.5$ and $0.75 - 0.25$. Moreover, we tested each combination with all the volume rotations tried previously, therefore resulting in three tests for each configuration of the parameters of the loss function.

The results for the experiment executed on each pair of values are depicted in figures 4.9, 4.10 and 4.11 respectively.

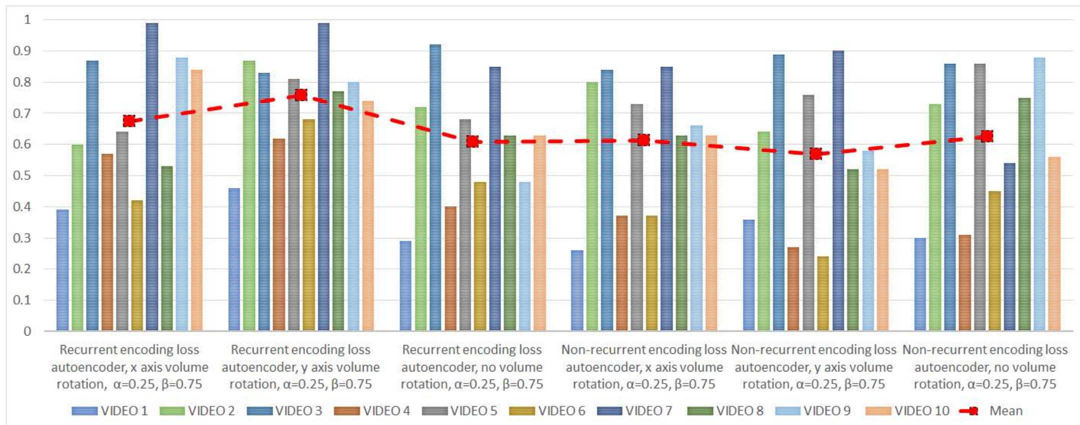

Figure 4.9: *AUC values for the ROC curves scored by the encoding loss autoencoders trained with $\alpha = 0.25$ and $\beta = 0.75$.*

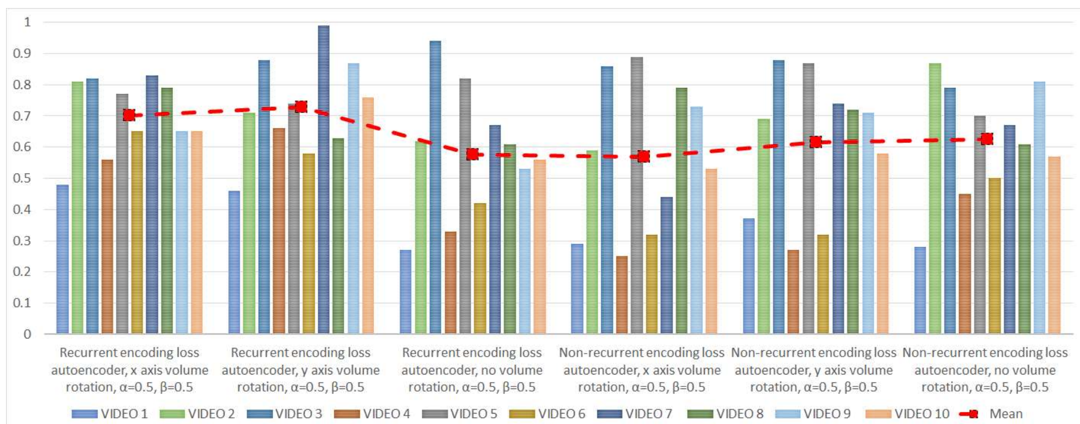

Figure 4.10: *AUC values for the ROC curves scored by the encoding loss autoencoders trained with $\alpha = 0.5$ and $\beta = 0.5$.*

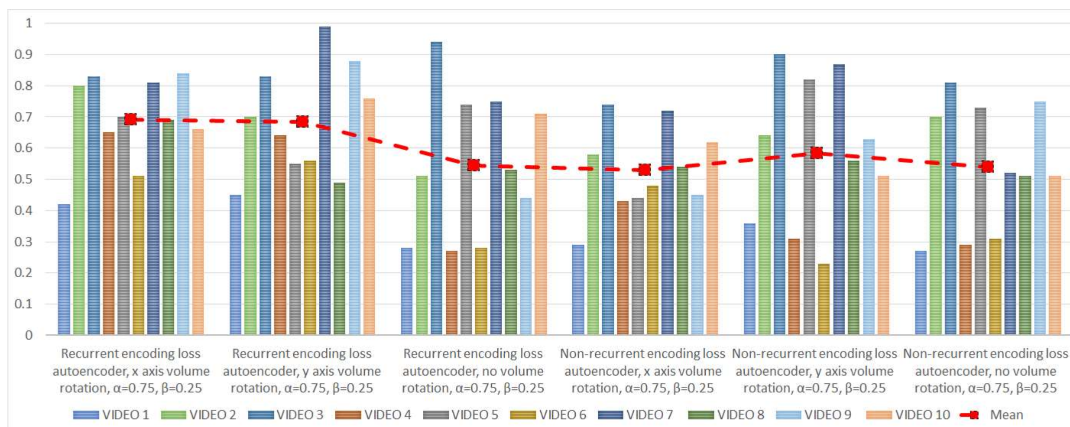

Figure 4.11: *AUC values for the ROC curves scored by the encoding loss autoencoders trained with* $\alpha = 0.75$ *and* $\beta = 0.25$.

For this last group of tests, we can observe immediately that for all the combinations of α and β the recurrent autoencoders seem to perform slightly better than their non-recurrent counterparts. Specifically, the recurrent networks working with the rotated volumes have better mean values for the AUC and more consistent performances across all videos.

We can take as the best example the recurrent autoencoder working on the y axis rotated volume, with $\alpha = 0.25$ and $\beta = 0.75$. This network, while reaching the same performance of our basic autoencoders in videos such as the number 7, improved its results on other more critical videos such as the number 8 (AUC=0.77) and number 6 (AUC=0.68, see figure 4.12). Similarly, the recurrent autoencoders working on the y axis but with $\alpha = 0.5$ and 0.75, and $\beta = 0.5$ and 0.25, improved their performances on video number 4, reaching as AUC values respectively 0.66 and 0.64.

These results suggest us that the combination of recurrency, volume rotation and working with the loss function on the hidden representation, makes our autoencoders' performances less dependent on the content of the splicing, therefore making its hidden representation more able to generalize the manifold of splicing free patches.

Anyway, we must notice that this solution is not yet generalizing well on all videos, with again the video number 1 being the most critical and resistant in the localization of the splicing. Finally, no sensible difference is observable by changing the parameters α and β, probably due to the magnitude of the values considered in our test.

4.5.3 Regularized autoencoders

We close this chapter illustrating the last group of tests executed on the family of regularized autoencoders. The procedure followed is the same

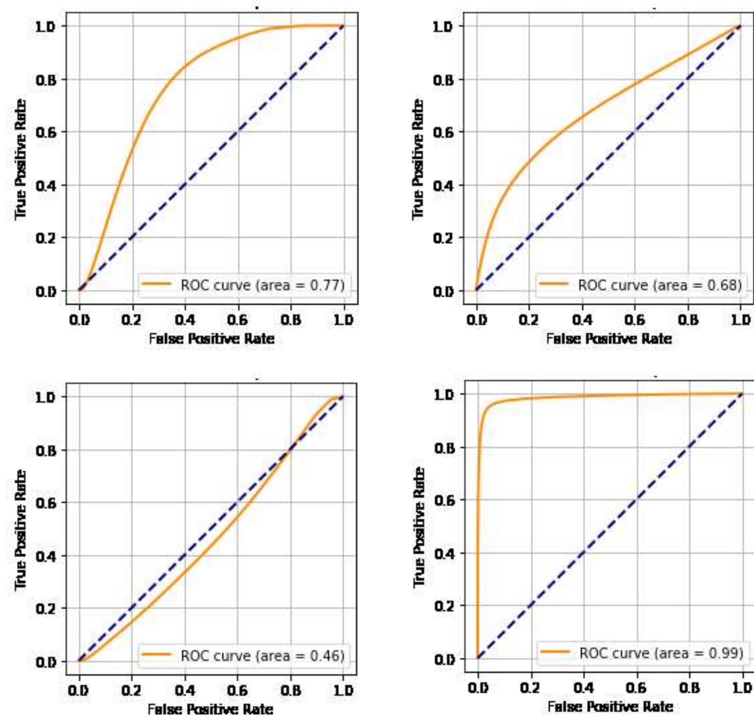

Figure 4.12: *Some of the ROC curves scored by the recurrent autoencoder working with the y axis rotated volume and trained with $\alpha = 0.25$ and $\beta = 0.75$. On the top row we have: on the left, the ROC curve for the video number 8; on the right the one scored for video number 6. On the bottom row: on the left the ROC scored for the video number 1; on the right for video number 7.*

of the precedent subsection: we have tested each autoencoder, recurrent and non-recurrent, for each combination of the values α and β of the regularized loss function used during training. For each pair moreover, we have tested the network with all the possible rotations of the volumes.

We have reported the AUC values of the ROC curves scored by each network in the figures 4.13, 4.14 and 4.15. The results obtained are so far the best among all tests executed. We first notice that there is no significant difference in this case between the performances of the recurrent and non-recurrent autoencoders. All networks present an average AUC value greater or equal than 0.73, repeating the performances of the basic and encoding loss autoencoders in the precedent tests. Moreover, on videos 6 and 8, which showed to be quite critical, the performances are again improved with respect to the encoding loss autoencoders, since on both of them all the regularized autoencoders reached an AUC greater or equal to 0.75.

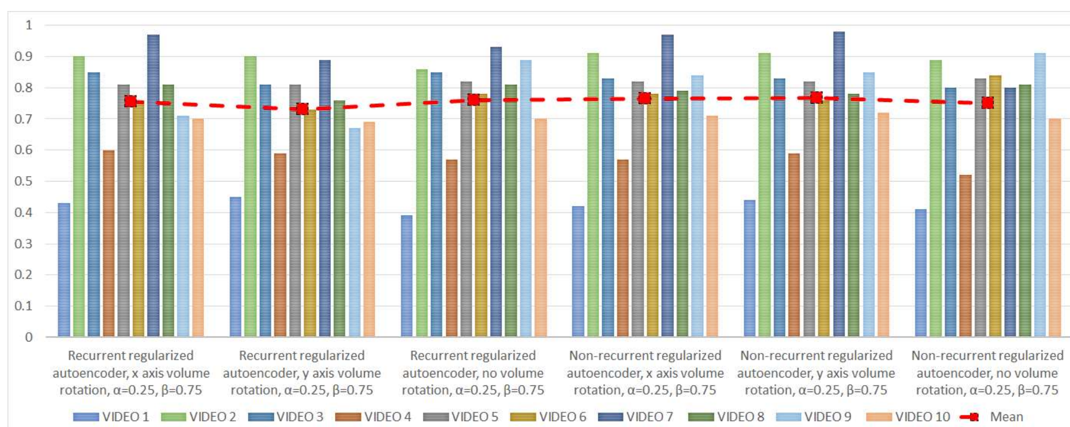

Figure 4.13: *AUC values for the ROC curves scored by the regularized autoencoders trained with* $\alpha = 0.25$ *and* $\beta = 0.75$*.*

Another important aspect regards the consistency of the performance of the networks. On almost all the videos the minimum AUC obtained is greater than or equal to 0.7, with the only exceptions of videos 1 and 4. This aspect denotes that the regularized autoencoders are the best networks trained in our work so far in creating an effective hidden representation of splicing free patches. However, we have to notice that the low performances on video 1 and 4 are probably an indication that the splicing localization process of our network is somehow still dependent on the content of the attack. Still, the improvements on videos 6 and 8 are encouraging, denoting that the presence of a regularization mechanism in the loss function during training is an essential factor in "forcing" the hidden representation to capture the local coordinates of the manifold of

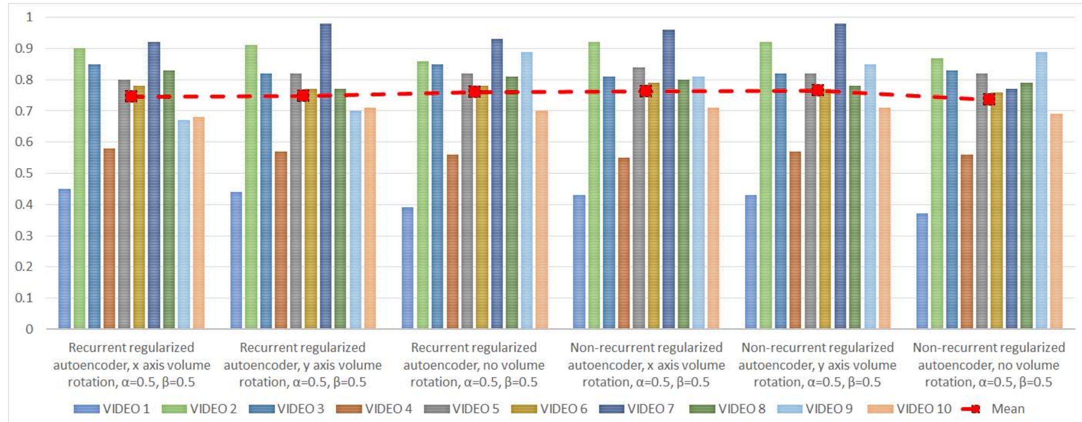

Figure 4.14: *AUC values for the ROC curves scored by the regularized autoencoders trained with* $\alpha = 0.5$ *and* $\beta = 0.5$*.*

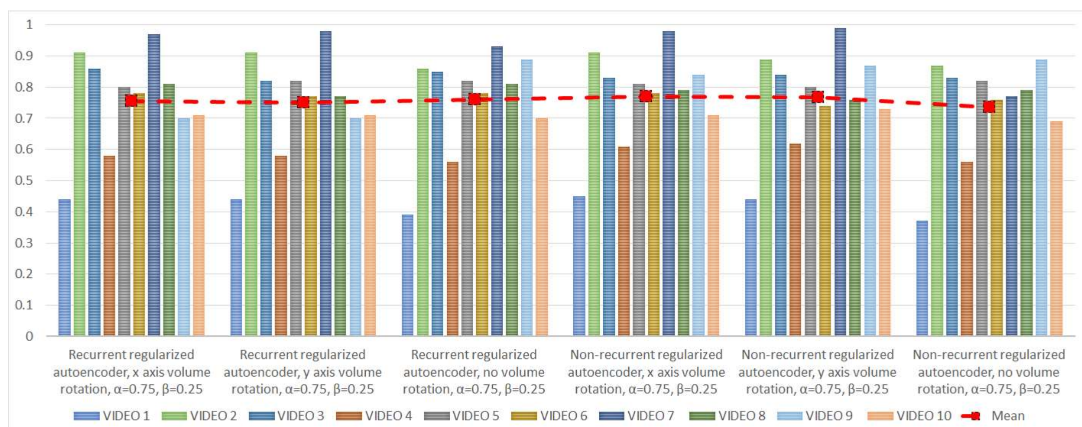

Figure 4.15: *AUC values for the ROC curves scored by the regularized autoencoders trained with* $\alpha = 0.75$ *and* $\beta = 0.25$*.*

clean patches[2].

Interestingly, besides the fact that even in this case there is no sensible difference in performance when using one pair of values for α and β with respect to another, we can also notice that the networks working on the rotated volumes in this case do not show any enhancement in their performance with respect to the others (see figure 4.16).

This is not bad news by the way. Indeed, this means that all rotated volumes bring enough information to the network for a correct tampering detection. Results obtained with these network can be potentially merged in a fusion framework exploiting the possible complementarity of the obtained tampering masks.

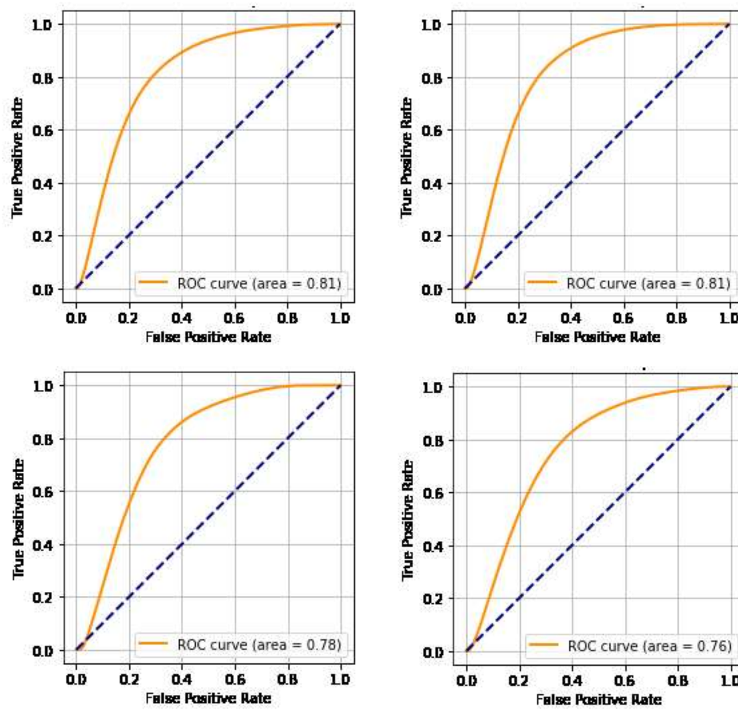

Figure 4.16: *ROC curves for the video number 8. On the top row we can see: on the left, the ROC scored by the recurrent autoencoder working with no volume rotation; on the right, the ROC obtained by working with the x axis rotated volume. On the bottom, we can see: on the left, the ROC scored by the non-recurrent autoencoder working on the y axis rotated volume; on the right, the recurrent autoencoder working with the x axis rotated volume. The values of α and β are equal to 0.25 and 0.75 for all networks.*

[2] This factor is often highlighted by Goodfellow et al. in [1], chapter 14 section 2.

4.6 Label-unaware results

All results shown up to now have been presented under the very strict assumption that a forensic analyst is interested in detecting which portion of the video is original, and which is modified. However, video splicing detection is an ill-posed problem by definition: when two videos are merged together, the definition of which sequence is the original and which sequence is the alien one is strongly ambiguous.

In other words, this means that we are not really interested in labeling one portion of the video as pristine, and one portion as forged. Conversely, we are mainly interested in detecting that two regions of the spliced sequence do not belong to the same class (i.e., pristine or modified).

In the light of this consideration, in this section we report the results achieved evaluating a selected series of the proposed networks in an anomaly detection framework: we consider flipped tampering masks (i.e., those with zeros and ones inverted) as correct. In doing so, we evaluate how good are the networks in recognizing that a video is a composition, and which regions of the video are not coherent, disregarding the fuzzy and ambiguous concept of forged vs. pristine.

Figure 4.17 shows these results for a selected subset of networks of all the three families of architectures. It is possible to notice that in this label-unaware framework, results improve up to 10%, especially for the non-recurrent models. Our best models instead repeated their performances, confirming the value of the solution proposed in the precedent setting.

In conclusion, this highlights that even some of the very simple and light proposed architectures can be used in this scenario.

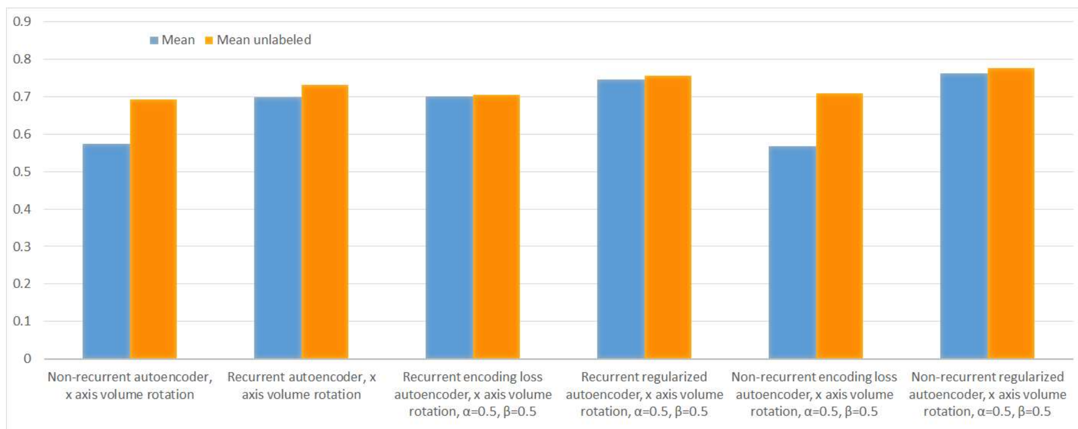

Figure 4.17: *AUC mean values for a selected set of networks. We took networks working only with volumes rotated along the x axis.*

5
Conclusions and future works

The fast and widespread diffusion of Internet-based media communication all around the globe, together with the predominant role that social media are assuming in every aspect of our life, including politics, has brought back to the attention of multimedia forensics researchers the problem of video splicing localization. Indeed this field has been somehow neglected due to the difficulties related to the processing of video signals in an efficient way. Goal of this thesis, therefore, has been to test deep learning tools, which offer several advantages for the forensics analysis such as great computational speed, easiness of configuration, etc., in the challenging scenario of video splicing detection and localization.

The use of convolutional and recurrent neural networks is at the base of this work. The first in fact are nowadays employed as a standard approach in many tasks related to computer vision and image processing, and have started to be explored by multimedia forensics researchers too. The latter, being specifically designed to handle sequential data, are particularly suited for elaborating time-series of objects such as video signals. For this reason, they have been consistently exploited in fields such as video anomaly detection, with some recent attempts done in the forensics literature too.

The analogies between the video forensics and video anomaly detection fields motivated us in using an anomaly detection approach for the localization of splicing. Considering as anomalies patches extracted from spliced regions of the videos, we built different models of autoencoders, fitting their hidden representation to the manifold of splicing-free patches, and localizing splicing as inputs badly reconstructed. The use of spatio-temporal and spatial cropping in the extraction of the patches gave us a fine granularity in spotting attacks, which take place in both

the spatial and temporal dimension of the signal.

After some preliminary tests, needed to decide some of the parameters of our pipeline and some architectural choices of our networks, we proceeded in evaluating the performance of our proposed method. We developed three families of autoencoders, declined in both a recurrent and a non-recurrent version. The recurrent version is based on the convolutional LSTM model, designed to handle temporal sequences of images. To the best of our knowledge, this has never been applied to the task of video splicing localization. The non-recurrent model instead is based on convolutional neural networks, and proved to be an effective feature extractor when applied to the task of copy-detection in images. Aim of our work was to verify if the use of networks that specifically take into account the sequential generating process of the input data, can improve the performance for the task at hand.

The three families have been defined as basic, encoding loss and regularized autoencoders, and differ for the type of loss function used during training. The basic family employed a simple MSE between the input and reconstructed patch; the encoding loss, in addition to the basic term, uses the MSE between the hidden representation of input and reconstructed patch; finally, the regularized autoencoders employ the MSE between the hidden representation of input and reconstructed patch plus an additional term that accounts for the mean of the energy of the reconstructed patch. For these last two families of networks, we investigated how the different loss terms contributed to the final detection performances.

All the networks have been evaluated using the dataset presented by the authors in [8]. It is composed by two sets of videos. The first contains only clean videos, the second the same clean videos but spliced adding content coming from other sources. Together with the "forged" set, all the masks used for the splicing addition are provided. We trained our autoencoders only on the clean video set, extracting patches from all the regions of each video and dividing them in a training and in a validation set to stop the procedure, and then tested them extracting patches from the forged video set and estimating a mask for localizing possible attacks.

A novel aspect of our work is the approach used in the patch extraction process. Considering the tensor representing the discrete video signal as a volume, by permuting the dimensions of this tensor we are able to gather the patches from a rotated version of the volume. Patches extracted in such way have a different semantic with respect to the ones that are cropped when considering the "frontal view" of the volume, therefore as a temporal sequence of frames. Such an approach on our assumption gives our networks a different perspective in observing their inputs, allowing them to spot splicing attacks otherwise not noticeable. We tested our hypothesis by training each network of each family on patches extracted from non-rotated, and 90° rotated volumes along the x, y axes.

The experiments with our basic family of autoencoders showed some interesting results. In general, the recurrent networks worked slightly

better than the non-recurrent ones, with the models analyzing rotated volumes having the best average and individual performances on most of the videos. We observed however that the evaluation metric used, the mean AUC of the ROC curves scored by the networks on each video, is biased by some very good performances of the networks on singular videos, with other examples of the dataset being critical. In particular video number 1, 4, 6 and 8 have been the most resistant to the splicing localization, with all of them being characterized by having the splicing content quite similar to the non-spliced regions. This suggests that our basic autoencoders are not really able to create an hidden representation which is specific to clean patches. Probably the representation is still quite general, robust and able to recreate spliced frames without much effort. It is interesting though to see that the convolutional LSTM model and the rotated volume still give good insights in localizing the splicing in some of the videos.

In the light of the aforementioned considerations, we expected the encoding loss autoencoders to show better performances with respect to their basic counterparts. Indeed, the experiments conducted on these networks confirmed and improved the results obtained by the recurrent autoencoders in their overall average performances. Again, the recurrent autoencoders working with rotated volumes performed best, just showing some troubles when processing some of the videos, in particular video 1 and 4. We observed improvements on video number 6 and 8, especially for the recurrent autoencoder analyzing the y axis rotated volume, but no sensible difference in behaviour for the use of one set of parameters for the terms of the loss function over another. Indeed, the presence of the MSE on the hidden representation is behaving like a regularization term on the representation itself, thus "forcing" it on the manifold of splicing-free patches.

This last consideration therefore brought us to our last set of experiments on regularized autoencoders. Working with the MSE on the internal representation, following the work of [5], while still remaining in our unsupervised approach we added a term in our loss function. Specifically, we chose the mean of the energy of the overall patch, on the assumption that such a regularization term would constrain the autoencoder to reproduce content with "smoother" transitions.

In fact, this last family of autoencoders revealed to be the most efficient on all videos. We still observed difficulties in the localization of splicing for videos number 1 and 4, but a good improvement on all the remaining 8 videos. Interestingly, there is no sensible difference in the performances between recurrent and non-recurrent autoencoders in this family of networks, and no difference when working with rotated and non-rotated volumes. The different set of loss weights seem to not have an huge impact on the results obtained by the autoencoders.

As a final remark, we would like to highlight that we considered video splicing detection as a binary classification task: determine, given one sample, if it belongs to the spliced class or not. However, we proposed a solution that follows an anomaly detection pipeline, not taking class labels into account. We have seen that, without a mechanism that implicitly, or explicitly, makes our model distinguish between the two typologies of samples, video splicing detection is not an easy task. However, if we relax the two-class constraint, and we simply focus on evaluating how good the network is in finding anomalies (no matters of which class), results improve significantly also with the worst performing networks.

Thanks to the performed tests and analysis, we have been able to outline some possible steps to further improve our method in the future. A possible future work in fact might consider the use of other types of neural networks, for which the paradigm of the rotation of the volume is more suited. An example are **3D convolutional neural networks** [72]. Recurrent neural networks surely provided to be effective, but using 3D CNNs we could directly work with the patches considering them as volumes instead of sequences, therefore changing the semantics of how the data is processed by the network.

Another viable and immediate future work might consider instead a more clever way of exploiting the paradigm with the deep learning tools already used. For instance, instead of simply making our recurrent autoencoders reconstruct the patches re-rotating them, we can think of making different networks see the same patch from the three different angles. Taking the reconstructed volumes of each one, we can merge or integrate them in some way, obtaining an output which is a weighted and summarizing view of the three perspectives of the rotated volume (see figure 5.1 for an exemplification). Such approach can be followed by using non-recurrent autoencoders too.

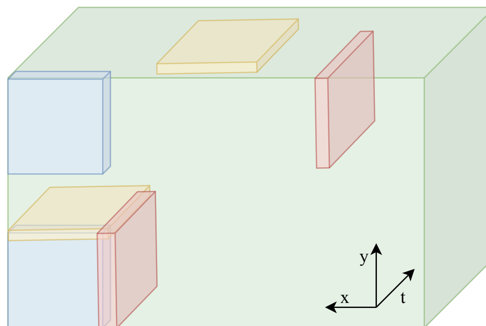

Figure 5.1: *Example of a possible procedure of volume merging. The three colored small volumes represent sequence patches extracted from the rotated volume separately. If instead they are cropped from the same region of interest, as it happens in the bottom left corner, we can construct an overall representation that is a weighted perspective of the three different angles.*

In general, we can say that the proposed methodology is enough flexible to be declined in several ways, with different tools and in different scenarios. Other interesting future directions of research may be the use of our pipeline in a semi-supervised context, developing a software which allows a user to indicate to the networks spliced areas for training and testing, or the comparison and integration of our work with more classical supervised approaches.

Living in a world where safety and trust are fragile and in constant danger, with the rising role of multimedia content in our everyday life, we hope that our work could provide some inspiration to other researchers in order to restore the attention which video forensics surely deserves in the field.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[3] C. Olah, "Understanding lstm networks http://colah.github.io/posts/2015-08-Understanding-LSTMs/," 2015.

[4] L. Bondi, S. Lameri, D. Güera, P. Bestagini, E. J. Delp, and S. Tubaro, "Tampering detection and localization through clustering of camera-based cnn features," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1855–1864, IEEE, 2017.

[5] D. Cozzolino and L. Verdoliva, "Single-image splicing localization through autoencoder-based anomaly detection," in *Proc. IEEE Int. Workshop Information Forensics and Security (WIFS)*, pp. 1–6, Dec. 2016.

[6] P. Bestagini, S. Milani, M. Tagliasacchi, and S. Tubaro, "Local tampering detection in video sequences," in *Proc. IEEE 15th Int. Workshop Multimedia Signal Processing (MMSP)*, pp. 488–493, Sept. 2013.

[7] P. Bestagini, S. Milani, M. Tagliasacchi, and S. Tubaro, "Codec and gop identification in double compressed videos," *IEEE Transactions on Image Processing*, vol. 25, pp. 2298–2310, May 2016.

[8] D. D'Avino, D. Cozzolino, G. Poggi, and L. Verdoliva, "Autoencoder with recurrent neural networks for video forgery detection," *Electronic Imaging*, vol. 2017, pp. 92–99, 01 2017.

[9] Y. S. Chong and Y. H. Tay, "Abnormal event detection in videos using spatiotemporal autoencoder," *Advances in Neural Networks - ISNN 2017*, Jan. 2017.

[10] S. K. Yarlagadda, D. Güera, P. Bestagini, F. Maggie Zhu, S. Tubaro, and E. J. Delp, "Satellite image forgery detection and localization using gan and one-class classifier," *Electronic Imaging*, vol. 2018, no. 7, pp. 1–9, 2018.

[11] "Internet & social media statistics https://influencermarketinghub.com/internet-real-time-social-media-statistics/," *Influencer Marketing*.

[12] R. Goldman, "Reading fake news, pakistani minister directs nuclear threat at israel. https://www.nytimes.com/2016/12/24/world/asia/pakistan-israel-khawaja-asif-fake-news-nuclear.html," *New York Times*, Dec. 2016.

[13] D. Venkatraman and A. Makur, "A compressive sensing approach to object-based surveillance video coding," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3513–3516, IEEE, 2009.

[14] J. Christian, "Experts fear face swapping tech could start an international showdown https://theoutline.com/post/3179/deepfake-videos-are-freaking-experts-out?zd=2&zi=mmjqprqj," *The Outline*, 2018.

[15] A. Rocha, W. Scheirer, T. Boult, and S. Goldenstein, "Vision of the unseen: Current trends and challenges in digital image and video forensics," *ACM Comput. Surv.*, vol. 43, pp. 26:1–26:42, Oct. 2011.

[16] M. C. Stamm, M. Wu, and K. J. R. Liu, "Information forensics: An overview of the first decade," *IEEE Access*, vol. 1, pp. 167–200, 2013.

[17] A. Piva, "An overview on image forensics," *ISRN Signal Processing*, vol. 2013, 2013.

[18] S. Milani, M. Fontani, P. Bestagini, M. Barni, A. Piva, M. Tagliasacchi, and S. Tubaro, "An overview on video forensics," *APSIPA Transactions on Signal and Information Processing*, vol. 1, p. e2, 2012.

[19] L. Bondi, L. Baroffio, D. Güera, P. Bestagini, E. J. Delp, and S. Tubaro, "First steps toward camera model identification with convolutional neural networks," *IEEE Signal Processing Letters*, vol. 24, pp. 259–263, Mar. 2017.

[20] M. Buccoli, P. Bestagini, M. Zanoni, A. Sarti, and S. Tubaro, "Unsupervised feature learning for bootleg detection using deep learning architectures," in *Proc. IEEE Int. Workshop Information Forensics and Security (WIFS)*, pp. 131–136, Dec. 2014.

[21] B. Bayar and M. C. Stamm, “A deep learning approach to universal image manipulation detection using a new convolutional layer,” in *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec ’16, (New York, NY, USA), pp. 5–10, ACM, 2016.

[22] G. Xu, H. Wu, and Y. Shi, “Structural design of convolutional neural networks for steganalysis,” *IEEE Signal Processing Letters*, vol. 23, pp. 708–712, May 2016.

[23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[24] Y. LeCun, K. Kavukcuoglu, and C. Farabet, “Convolutional networks and applications in vision,” in *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 253–256, May 2010.

[25] M. Barni, L. Bondi, N. Bonettini, P. Bestagini, A. Costanzo, M. Maggini, B. Tondi, and S. Tubaro, “Aligned and non-aligned double jpeg detection using convolutional neural networks,” *Journal of Visual Communication and Image Representation*, vol. 49, 08 2017.

[26] S. Verde, L. Bondi, P. Bestagini, S. Milani, G. Calvagno, and S. Tubaro, “Video codec forensics based on convolutional neural networks,” in *Proc. 25th IEEE Int. Conf. Image Processing (ICIP)*, pp. 530–534, Oct. 2018.

[27] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 2222–2232, Oct. 2017.

[29] B. Kiran, D. Thomas, and R. Parakkal, “An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos,” *Journal of Imaging*, vol. 4, no. 2, p. 36, 2018.

[30] M. Hasan, J. Choi, J. Neumann, A. K. Roy-Chowdhury, and L. S. Davis, “Learning temporal regularity in video sequences,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 733–742, June 2016.

[31] N. Srivastava, E. Mansimov, and R. Salakhutdinov, “Unsupervised learning of video representations using lstms,” in *Proceedings of the*

32Nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, pp. 843–852, JMLR.org, 2015.

[32] X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 802–810, Curran Associates, Inc., 2015.

[33] T. M. Mitchell, *Machine Learning.* New York, NY, USA: McGraw-Hill, Inc., 1 ed., 1997.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization,"

[35] J. Chen, X. Kang, Y. Liu, and Z. J. Wang, "Median filtering forensics based on convolutional neural networks," *IEEE Signal Processing Letters*, vol. 22, pp. 1849–1853, Nov. 2015.

[36] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, Mar. 1994.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

[38] Y. Bengio, P. Frasconi, and P. Simard, "The problem of learning long-term dependencies in recurrent networks," in *Proc. IEEE Int. Conf. Neural Networks*, pp. 1183–1188 vol.3, Mar. 1993.

[39] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *NIPS*, 2002.

[40] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: a new framework for neural computation based on perturbations.," *Neural computation*, vol. 14, pp. 2531–2560, Nov. 2002.

[41] M. Zampoglou, S. Papadopoulos, and Y. Kompatsiaris, "Large-scale evaluation of splicing localization algorithms for web images," *Multimedia Tools and Applications*, vol. 76, p. 4801, Feb. 2017.

[42] C. Li and Y. Li, "Color-decoupled photo response non-uniformity for digital image forensics," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 260–271, Feb. 2012.

[43] G. Chierchia, G. Poggi, C. Sansone, and L. Verdoliva, “A bayesian-mrf approach for prnu-based image forgery detection,” *IEEE Transactions on Information Forensics and Security*, vol. 9, pp. 554–567, Apr. 2014.

[44] M. Chen, J. Fridrich, M. Goljan, and J. Lukas, “Determining image origin and integrity using sensor noise,” *IEEE Transactions on Information Forensics and Security*, vol. 3, pp. 74–90, Mar. 2008.

[45] B. Mahdian and S. Saic, “Using noise inconsistencies for blind image forensics,” *Image and Vision Computing*, vol. 27, no. 10, pp. 1497–1503, 2009.

[46] S. Lyu, X. Pan, and X. Zhang, “Exposing region splicing forgeries with blind local noise estimation,” *International Journal of Computer Vision*, vol. 110, p. 202, Nov. 2014.

[47] D. Cozzolino, G. Poggi, and L. Verdoliva, “Splicebuster: A new blind image splicing detector,” in *Proc. IEEE Int. Workshop Information Forensics and Security (WIFS)*, pp. 1–6, Nov. 2015.

[48] A. E. Dirik, H. T. Sencar, and N. Memon, “Digital single lens reflex camera identification from traces of sensor dust,” *IEEE Transactions on Information Forensics and Security*, vol. 3, pp. 539–552, Sept. 2008.

[49] A. E. Dirik and N. Memon, “Image tamper detection based on demosaicing artifacts,” in *2009 16th IEEE International Conference on Image Processing (ICIP)*, pp. 1497–1500, IEEE, 2009.

[50] P. Ferrara, T. Bianchi, A. De Rosa, and A. Piva, “Image forgery localization via fine-grained analysis of cfa artifacts,” *IEEE Transactions on Information Forensics and Security*, vol. 7, pp. 1566–1577, Oct. 2012.

[51] T. Bianchi, A. De Rosa, and A. Piva, “Improved dct coefficient analysis for forgery localization in jpeg images,” in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2444–2447, IEEE, 2011.

[52] I. Amerini, R. Becarelli, R. Caldelli, and A. Del Mastio, “Splicing forgeries localization through the use of first digit features,” in *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 143–148, IEEE, 2014.

[53] Z. Lin, J. He, X. Tang, and C.-K. Tang, “Fast, automatic and fine-grained tampered jpeg image detection via dct coefficient analysis,” *Pattern Recognition*, vol. 42, no. 11, pp. 2492–2501, 2009.

[54] W. Luo, Z. Qu, J. Huang, and G. Qiu, "A novel method for detecting cropped and recompressed image block," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, vol. 2, pp. II–217, IEEE, 2007.

[55] W. Li, Y. Yuan, and N. Yu, "Passive detection of doctored jpeg image via block artifact grid extraction," *Signal Processing*, vol. 89, no. 9, pp. 1821–1829, 2009.

[56] H. Farid, "Exposing digital forgeries from JPEG ghosts," *IEEE Transactions on Information Forensics and Security*, vol. 4, pp. 154–160, Mar. 2009.

[57] H. C. Patel and M. M. Patel, "An improvement of forgery video detection technique using error level analysis," *International Journal of Computer Applications*, vol. 111, no. 15, 2015.

[58] W. Wang, J. Dong, and T. Tan, "Tampered region localization of digital color images based on jpeg compression noise," in *International Workshop on Digital Watermarking*, pp. 120–133, Springer, 2010.

[59] M. C. Stamm, W. S. Lin, and K. J. R. Liu, "Temporal forensics and anti-forensics for motion compensated video," *IEEE Transactions on Information Forensics and Security*, vol. 7, pp. 1315–1329, Aug. 2012.

[60] S. Bian, W. Luo, and J. Huang, "Exposing fake bit rate videos and estimating original bit rates," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, pp. 2144–2154, Dec. 2014.

[61] and Chih-Chung Hsu, Tzu-Yi Hung, Chia-Wen Lin, and Chiou-Ting Hsu, "Video forgery detection using correlation of noise residue," in *Proc. IEEE 10th Workshop Multimedia Signal Processing*, pp. 170–174, Oct. 2008.

[62] P. Mullan, D. Cozzolino, L. Verdoliva, and C. Riess, "Residual-based forensic comparison of video sequences," in *Proc. IEEE Int. Conf. Image Processing (ICIP)*, pp. 1507–1511, Sept. 2017.

[63] S. Mandelli, P. Bestagini, S. Tubaro, D. Cozzolino, and L. Verdoliva, "Blind detection and localization of video temporal splicing exploiting sensor-based footprints," in *2018 26th European Signal Processing Conference (EUSIPCO)*, pp. 1362–1366, IEEE, 2018.

[64] L. D'Amiano, D. Cozzolino, G. Poggi, and L. Verdoliva, "Video forgery detection and localization based on 3d patchmatch," in *2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pp. 1–6, June 2015.

[65] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, “Patchmatch: A randomized correspondence algorithm for structural image editing,” *ACM Trans. Graph.*, vol. 28, pp. 24:1–24:11, July 2009.

[66] J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. Nießner, “Face2Face: Real-time face capture and reenactment of rgb videos,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 2387–2395, June 2016.

[67] F. Matern, C. Riess, and M. Stamminger, “Exploiting visual artifacts to expose deepfakes and face manipulations,” in *Proc. IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pp. 83–92, Jan. 2019.

[68] Y. Li, M. Chang, and S. Lyu, “In ictu oculi: Exposing AI created fake videos by detecting eye blinking,” in *Proc. IEEE Int. Workshop Information Forensics and Security (WIFS)*, pp. 1–7, Dec. 2018.

[69] D. Güera and E. J. Delp, “Deepfake video detection using recurrent neural networks,” in *Proc. 15th IEEE Int. Conf. Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, Nov. 2018.

[70] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,”

[71] F. Chollet *et al.*, “Keras.” https://keras.io, 2015.

[72] S. Ji, W. Xu, M. Yang, and K. Yu, “3d convolutional neural networks for human action recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.