POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Deepfake detection using LSTMs, Transformers and video-level artifacts

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Nicola Rosetti

Advisor: Prof. Mark James Carman

Co-advisor: Prof. Paolo Bestagini

Academic year: 2020-2021

---

1. Introduction

Nowadays, media manipulation is widely spreading in many different fields of our lives; it is very easy, for example, to come across a modified video or a selfie with a filter. Another type of media manipulation are the so called *deepfakes*. Deepfakes are media modified via deep learning techniques whose peculiarity is their ability to quickly and cheaply generate pretty convincing crafted videos (or images). They are based on the replacement of a person with another similar person.
These powerful tools have been firstly developed by researchers, who were trying to come up with new efficient methods to manipulate media. Afterwards, these techniques started to spread among non-expert users. The diffusion of these videos has been huge in the last couple of years. We need to be well aware of this phenomenon, since most of these videos are used and spread for malicious deeds. Indeed, the threats these kind of video pose are serious: they can be used to defame people publicly, to spread fake news or to fool face recognition systems.
To tackle this problem, researchers have started studying ways to detect these deepfakes, mainly employing deep learning techniques.

The goal of our work is to further investigate the time domain in deepfake detection: in particular, we propose new architectures and a new approach based on artifacts detection.

1.1. Deepfake generation techniques

As of now, we can divide deepfake generation techniques in 5 main categories, which are based on face manipulation:

- face swapping, based on entirely swapping a face with another face;
- attribute manipulation, based on modifying face attributes (like gender, age) while leaving the rest unchanged;
- puppet mastery, based on the control of a face in a video with another face in the source video;
- lip syncing, based on modifying the lip movement to be coherent with a specific audio clip;
- face synthesis, based on creating from scratch not existing faces.

1.2. Deepfake generation

The architectures used to generate deepfakes are Deepfake Autoencoders (DFAE) and Generative Adversarial Networks (GAN).

For DFAE generation, 3 architecture blocks are used: a (shared) encoder, decoder A and decoder B. At first, the shared encoder with decoder A is trained on the source images. Then, the shared encoder with decoder B is trained on the target images. At generation time, an input image is fed to the autoencoder composed of encoder and decoder B, thus creating the modified face.
GANs have been recently employed to improve deepfakes quality, improving smoothness of the output features and creating more realistic eye movement.

2. Related work

Deepfake detection is a very popular research field. There are mainly 2 approaches to tackle this problem, both exploiting deep learning models: frame-based and video-based. Here we cite just some works on the topic.
Among the frame-based works, the CNN architectures are the current state of the art. Rossler et al. [5] showed that CNN-based architecture such as XceptionNet are very good in the Face Forensics dataset. As the winner of the Deepfake Detection Challenge, Selim Sef [1] showed that EfficientNet-B4 and B7 outperform any other method on the DFDC dataset.
Video-based approach has not been widely studied and the current state of the art is based on convolutional LSTMs. The most famous work on the topic is the one of Güera et al. [2], who proposed a convolutional LSTM with Inception-V3 as backbone, obtaining 99.7% accuracy on their own test set. Another famous conv-LSTM work is the one by Li et al. [3]. They proposed an unconventional way to detect deepfakes: inconsistent eye blinking patterns. The approach was very effective but it soon became obsolete after new deepfake techniques tackled this problem.
In the current state of the art, few researchers have tried to use other video-level architectures for deepfake detection. Some works proposed a way to detect deepfakes via 3DCNN, showing how these architectures have a better generalization capability with respect to other ones.

[1] https://github.com/selimsef/dfdc_deepfake_challenge

3. Dataset

The dataset we use for our work is the DFDC dataset, which is the largest deepfake video dataset available on the Web. The dataset is composed of 119,154 clips. These clips are 10 seconds long and most of them have a resolution of 1920x1080. We perform a train/validation/test split of 60%/20%/20%, considering the size of the dataset, using the first 30 directories as training set, from 31 to 40 as validation set and from 41 to 50 as test set.

4. Video-level deepfake detection

As a first time-related approach we propose video-level detection models. The employed pipeline for our work is showed in fig. 1. At first, we need to select a restricted number of frames to keep computation time low. Then, since the modifications are in the facial traits, from every frame we crop only a small region centered in the face area, using Blazeface. The sequence of faces of the selected frames is fed as input to our classifiers, which detects if a video is pristine or has been modified.

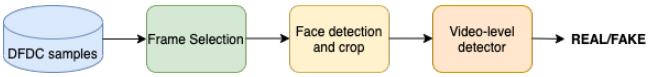


Figure 1: Proposed pipeline.

4.1. Preprocessing

For the preprocessing step we opt for a selection of 15 evenly spaced frames across the videos. We tried considering more frames, but the performance dropped. For every video, a crop of size 224×224 centered around the face is extracted, in order to be compatible with most architectures, to keep computational time low and to have enough information to correctly perform classification. The crops are extracted considering the Blazeface detection output and adapting the detection to a 224×224 final region.

4.2. Architectures

As video-level detection models, we propose 3 architectures: convolutional LSTM (as the one proposed by [2]), Video Transformer Network

(architecture similar to [4], code taken from [2]) and Timesformer [1] (code taken from the official repository). The convolutional LSTM archetype is studied to employ a state-of-the-art method with which we could compare the Transformers, state of the art for video classification but not yet used for deepfake detection.
We propose a convolutional LSTM with Inception-V3 pretrained on Imagenet as backbone, with a GAP layer to shrink the feature vector. The 2048-dimensional feature vector is fed into a 2048-units LSTM layer with 0.3 chance of dropout to reduce overfitting. The last hidden state of the LSTM is enth fed to 2 fully connected layers to finalize classification.
In addition, we propose another convolutional LSTM architecture, using EfficientB4 as backbone pretrained on the DFDC.
The VTN is composed by a CNN backbone and a self-attention module. As feature extractor, we choose EfficientNetB0 pretrained on Imagenet. We select it because we had memory constraints with other architectures and because it is a pretty effective network on the DFDC dataset. The feature vector is shrunk with a GAP layer and then fed to the 16-heads self-attention module.
The Timesformer architecture is based on dividing all frames into smaller crops and them performing spatio-temporal self-attention over all the crops. In our work we propose the result on the architecture pretrained on Kinetics-600 and with the divided space-time attention. d

5. Artifacts detection

As a second time-related approach, we perform a study on the video-level inconsistencies in deepfake videos. After having analyzed some videos we have come up with several potential inconsistency labels:

- Flickering: face features abruptly change across consecutive frames;
- Obfuscated Face: face blurred in consecutive frames of the video;
- Irregular facial traits: deepfake generation generates clearly artificial facial traits;
- Face proportions inconsistency: deepfake generation messes up right face proportions;
- Color inconsistency: deepfake generation messes up colors in the face;
- Glasses inconsistency: deepfake generation introduces glasses but some parts of them are missing.

5.1. Artifacts detection via Web Interface

Via a Web Interface, we collect data from users which are requested to detect above mentioned inconsistencies in deepfake videos. The dataset used for the survey is a collection of videos from the last 10 folders of the DFDC dataset. In total, we randomly select 200 video, 20 pristine and the others fake, divided in two 5-seconds chunks to make users' life easier.

5.2. Automatic flickering detection

As a further investigation, we propose a way to automatically detect flickering. We decide to detect flickering since it is the most frequent inconsistency and it can be, to a certain extent, synthetically replicated.

5.2.1. Synthetic dataset generation

To train a model to detect flickering we create our own ground truth for training and validation purposes. We start from DFDC dataset samples. We process every single sample of the first 40 directories of the DFDC. Every sample is treated as described in the following algorithm:

Algorithm 1 Samples selection

```
if label == FAKE then
    if 'original'_not_processed then
        'original' processed
        choice = random.binomial(0.5)
        if choice then
            create flickering sample
        else
            label 'original' as NO_FLICKERING
        end if
    else
        discard sample
    end if
else
    discard sample
end if
```

For the frame selection step, we opt for 30 consecutive frames from every video, taken ran-

[2] https://github.com/ppriyank/Video-Action-Transformer-Network-Pytorch-

domly from the entire sequence, because flickering is a more noticeable effect in consecutive frames. The creation of flickering samples is explained as follows:

Algorithm 2 Flickering samples generation

```
originalList = 'original' video stable cropping
fakeList = fake video stable cropping
index = 0
flickering_sample = [ ]
while index < 30 do
    choice = random.binomial(0.5)
    if choice then
        flickering_sample.append(fakeList[index])
    end if
    index +=1
end while
return flickering_sample
```

The basic idea of this approach is to try and force abrupt changes in consecutive frames by alternating real and fake frames. As mentioned in Algorithm 2 we perform a stable cropping of the samples. This cropping is performed since the face detector is not always correct or, if there are multiple subjects, it may fail at always recognizing the same face across consecutive frames, thus introducing abrupt changes that may mislead the classifier.
A video is considered stable if the standard deviation of the detection pixels coordinates (in terms of width and height) does not go above a certain threshold, namely the half of the average detection width (or height).
If the sample is not stable (in the case of flickering samples, both 'original' and fake video must be stable) then it is discarded. If the sample is stable, then a 224×224 crop is considered, fixed for every frame, symmetrically generated from the average detection pixel (in coordinates).

5.2.2. Flickering test set

The flickering test set is composed of 200 videos from the last 10 folders of the DFDC dataset. In particular, the dataset is composed of 180 FAKE videos and 20 REAL ones. We manually label these videos in order to have an available ground truth to evaluate our models.

6. Experiments

In this section we show the results of our work.

6.1. Video-level deepfake detection

To train our models we use Keras (for conv-LSTMs) and Pytorch (for Transformers). Considering the large amount of data used for training, models reach the validation minimum after few epochs (max 15 epochs).
Since the test set in unbalanced, we consider as additional evaluation metric the balanced accuracy together with accuracy and Area Under the Curve (AUC).
Results are shown in Table 1. As we can we can see, the conv-LSTM-IV3 and VTN approaches outperform all the other approaches. In fig. 2 and fig. 3 we can observe their confusion matrices.
Timesformer model does not really perform good on deepfake detection, having very poor performance on pristine samples.

Approach	Accuracy	Bal.acc.
conv-LSTM-IV3	**0.91**	0.86
conv-LSTM-IV3-bal	0.89	0.88
conv-LSTM-EB4	0.90	0.84
VTN-EB0	0.90	**0.90**
Timesformer	0.83	0.65

Table 1: Metrics of the models.

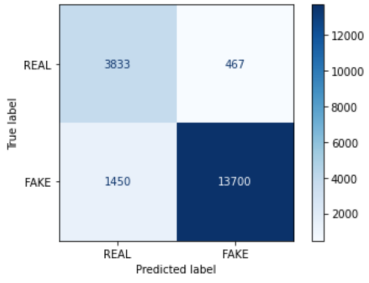


Figure 2: Confusion matrix of the VTN-EBO approach.

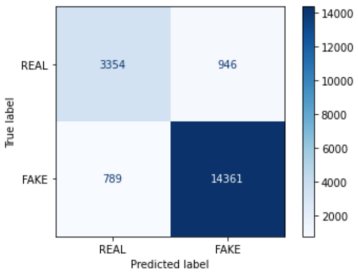


Figure 3: Confusion matrix of the conv-LSTM-IV3 approach.

6.2. Web Interface data

We have collected a total of 925 annotations, with 347 annotated samples. To evaluate human-level performance in detecting inconsistencies we make use of two statistics evaluating the agreement among several raters: inter-rater agreement and normalized inter-rate agreement.

Inter-rater agreement (IRA) The IRA is a metric that takes into consideration the number of samples with all coherent labels with respect to the total number of samples. The IRA is defined as the percentage of pair of coherent labels with respect to the total number of pair of labels. The equation of the IRA is the following:

$$IRA = \frac{N_{pos-pos} + N_{neg-neg}}{N_{TOT}} \quad (1)$$

where $N_{pos-pos}$ and $N_{neg-neg}$ are respectively the number of pairs of annotations with positive answers and negative answers and N_{TOT} is the total number of pairs of labels.

Normalized IRA The normalized IRA is a metric that gives a more precise evaluation of the agreement, since it takes consideration the possibility of raters giving the same answer by chance. The formula to compute the IRA is the following:

$$norm\_IRA = \frac{IRA - p_e}{1 - p_e} \quad (2)$$

where p_e is the probability that 2 raters will give the same answer by chance.
The statistics of the collected data are shown in Table 2.

Class	IRA(%)	p_e(%)	n_IRA
Flickering	65	52.4	0.27
Obf. face	80.6	66	0.43
Prop. incon.	75.1	73.5	0.06
Irr. traits	67.4	60.8	0.168
Color incon.	66.3	61	0.135
Glasses incon.	83.6	70	0.45

Table 2: Statistics collected from our Web Interface study.

6.3. Flickering detection

Here we show the results of some approaches we adopt for automatic flickering detection. The training is done similarly to the deepfake detection task.

6.3.1. Test set labeling

To label a test sample, we first crop the entire video with the stable cropping as described is Section 5. Then we consider 30 frames at a time, with a sliding window of 25 frames, from the beginning of the video until the end, and we label every single subsequence. If a subsequence is labeled as FLICKERING then the entire video is labeled as FLICKERING. The rationale behind this choice is that the flickering effect has no pattern and it may appear in a small portion of the video, therefore a flickering subsequence is enough to label the entire sample.

6.3.2. Results

The best results are obtained by the convolutional LSTM with Inception-V3 as backbone, using synthetic dataset generation as we have previously described. The approach obtains an accuracy of 76%. This method has problems with dark videos, therefore we tried to apply contrast enhancement, but this decreased the performance to 74%. Many other approaches were tried, like balancing the synthetic dataset or considering the difference among frames (with and without feature extraction), but no approach outperformed the most basic one. Table 3 shows the results of the 2 best approaches, adding Precision and Negative Predictive Value (NPV) as evaluation metrics.

Approach	Accuracy	Precision	NPV
stable crops	0.76	0.68	0.84
contrast+	0.74	0.68	0.80

Table 3: Flickering results.

7. Conclusions

In our work, we have explored different video-level approaches to deal with deepfake detection. As first approach, we have proposed state of the art models (conv-LSTMs) and unseen models for deepfake detection at video-level (VTN and Timesformer). VTN was able to reach LSTM performance in deepfake video detection showing pretty good results when exploiting finetuning of a convolutional feature extractor.
Unfortunately, presented video-level models are not able to reach frame-based classifiers performance. This result does not come unexpected since the winner of the DFDC came to the same conclusion, but it should not mislead. Indeed video-level detection is a much more challenging task. First, because it requires much more data than what is currently available on datasets. Second, because the employed architectures are aiming at spotting very little pixel discrepancies across frames and they are doing it by feeding to the LSTM (or self-attention, in our case) layer not the frames but their high-level features. It would be interesting as a future work to further explore video-level deepfake detection and in particular new Transformer architectures (or any combination of them), maybe with more available training data.
As second approach, we have analyzed deepfake videos and we have noticed some recurrent inconsistencies in them.
In particular, at first, we have focused on flickering and we have proposed a way to synthetically reproduce and detect it, showing the performance of our models. The overall achieved accuracy is 76% on our test set. With more available time, it would have been interesting to test other approaches and see if the performance could have been improved.
As a parallel study, we have built a Web Interface to collect user data, asking users to detect inconsistencies in videos. The annotations have been used to study human level performance on these inconsistencies and to compare it with our own models. Indeed our models accuracy (76%) has outperformed the average agreement of people on flickering (65%). As a research direction, it could be useful to use our interface to collect more data in order to directly use it as a training set; with this data we could train models on all inconsistencies we have come up with, evaluating and comparing the performance with respect to our models based on synthetic dataset generation.

References

[1] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? *arXiv preprint arXiv:2102.05095*, 2021.

[2] David Güera and Edward J Delp. Deepfake video detection using recurrent neural networks. In *2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS)*, pages 1–6. IEEE, 2018.

[3] Yuezun Li, Ming-Ching Chang, and Siwei Lyu. In ictu oculi: Exposing ai created fake videos by detecting eye blinking. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7. IEEE, 2018.

[4] Daniel Neimark, Omri Bar, Maya Zohar, and Dotan Asselmann. Video transformer network. *arXiv preprint arXiv:2102.00719*, 2021.

[5] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics: A large-scale video dataset for forgery detection in human faces. *arXiv preprint arXiv:1803.09179*, 2018.

POLITECNICO DI MILANO
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

Deepfake detection using LSTMs, Transformers and video-level artifacts

Supervisor: Prof. Mark James Carman
Cosupervisor: Prof. Paolo Bestagini

Master Thesis of:
Nicola Rosetti, matr. 940435

Academic Year 2020-2021

Ringraziamenti

Innanzitutto mi sembra doveroso ringraziare il mio relatore, Mark Carman, e il mio co-relatore, Paolo Bestagini, che mi hanno trasmesso gli strumenti, le conoscenze e la passione per gli argomenti di questa tesi.

Un ringraziamento speciale va fatto anche a tutto l'Image and Sound Processing Lab, per avermi messo a disposizione la potenza di calcolo senza la quale non avrei potuto ottenere ciò che ho ottenuto in questo lavoro.

Questo risultato, la tesi di laurea magistrale, non è solo calcoli, codice e stesura; è il culmine di tanti anni di un percorso a tratti tortuoso, portato a termine grazie ad una serie di persone che in ogni momento mi hanno accompagnato ed hanno contribuito alla realizzazione di questo traguardo.

Un enorme grazie ai miei genitori e mio fratello Lorenzo, che sono sempre stati dietro ad ogni mia decisione assecondandomi quando fosse necessario e mi hanno sostenuto per tutta la mia vita.

Ringrazio Lorenzo, Riccardo, Martina e Federica, che dal primo momento mi hanno sempre fatto sentire a casa seppur fossi distante.

Grazie anche a tutti i miei amici di triennale e magistrale, con cui ho condiviso intere giornate e interminabili sessioni.

Un ringraziamento va a Francesco, Enrico, Matteo, Gianluca, Dario, che dal liceo ci sono sempre stati; nonostante le strade si sono divise, sembra che con voi il tempo non passi mai.

Non posso poi non menzionare le amicizie più recenti, che in questo ultimo difficile periodo sono state in grado di rendere migliori le giornate.

Grazie infinite a Simone, Fabio e Michele, le mie colonne portanti, con i quali ho condiviso tantissimi bei ricordi e le serate giochi da tavolo.

Infine un grazie speciale a Giulia, per tutto. Con amore e certezze mi hai sostenuto e accompagnato in questo percorso e nella vita. Non sarei Nicola senza di te oggi.

Abstract

Deepfakes are swiftly becoming a reality. These media are generated by very powerful tools which allow fast and cheap media manipulation. Along with potential positive uses of such tools come malevolent uses and threats. Indeed, these videos can be used to defame people, spread fake news or impersonate others to access a bank account. Given the upcoming problems these media pose, researchers have started working on different ways to try and detect these types of media. As of now, several powerful frame-based detectors have been proposed, but the time domain has not been studied in depth.

In our work, we show two different approaches to study deepfake videos in the time domain. We first present several video-level deepfake detectors: from deepfake detection state of the art convolutional Long Short-Term Memory to previously untested architectures for this field, namely, Transformers. Afterwards, we focus on video-level artifacts in videos. These errors appear since the most used deepfake generation tools work frame-by-frame, causing inconsistencies across consecutive frames. We build a synthetic dataset of such videos by replicating the inconsistencies, and propose methods to spot them automatically using the models described above. At the same time, we build a Web Interface that allows users to label videos, asking them to spot inconsistencies. We use these labelled videos to evaluate our models and compare their performance with human annotators.

Sommario

I deepfakes stanno rapidamente divenendo realtà. Questi media sono generati da tecniche molto potenti di deep learning, le quali permettono di manipolare un video o un'immagine molto rapidamente e a basso costo. Il loro potenziale è altissimo, ma se vengono usati per scopi criminosi rappresentano un minaccia concreta: possono essere utilizzati per diffamare, spargere fake news o impersonare altre persone per riuscire a bypassare sistemi di riconoscimento.

Dati i problemi che stanno sorgendo, i ricercatori hanno iniziato a studiare tecniche per il riconoscimento di tali video. Ora come ora la maggior parte delle tecniche efficaci è costituita da classificatori frame-based, ossia che classificano un frame alla volta, mentre il dominio temporale (quindi cercare di studiare il video nel suo insieme) è ancora piuttosto acerbo.

Nel nostro lavoro presentiamo due approcci per studiare il dominio temporale nel riconoscimento di deepfakes. Prima di tutto, presentiamo vari modelli video-level: dalle Long Short-Term Memory convoluzionali, stato dell'arte per quanto riguarda il riconoscimento video-level, ai Transformers, nuovi in questo campo. Successivamente, come secondo approccio, studiamo artefatti temporali in questi video. Questi errori si verificano poichè la maggior parte delle tecniche di generazione dei deepfakes lavora frame per frame, causando inconsistenze tra frames consecutivi. Proponiamo quindi un modo di riconoscere queste inconsistenze, utilizzando i modelli descritti prima e il nostro dataset sintetico, basato sul tentare di replicare tali inconsistenze. Parallelamente, attraverso il crowdsourcing, raccogliamo una serie di annotazioni di diversi utenti, i quali devono riconoscere inconsistenze nei video e la cui performance viene utilizzata per valutare i risultati dei nostri modelli.

Contents

Chapter 1

Introduction

Media manipulation is something that is present in our life since the beginning of the 20th century. In the 1920s, the Soviet Union was performing visual manipulation of images and pictures for political purposes [1].

Throughout the 20th century, media manipulation has started to fill our lives in different contexts and for different tasks. During 1990s, video editing techniques were perfected in Hollywood, but they were employed only in a few movies, given the expensiveness of such tools. Nowadays, media manipulation is becoming cheaper and cheaper. When we surf the Internet or we are using a social network, it is rare to not come across any altered media - whether that be a simple selfie with a filter, a meme or a video edited to add certain effects. Media manipulation is something we have to deal with everyday of our life. An example of manipulated, widely spreading, media is the deepfake.

The word *deepfake* is used for videos or images that are synthetically created via deep learning tools by replacing a person in it with another similar person. Some of these deepfakes have become very famous, like the ones where Obama (Figure 1.1) or Putin appear to say something they never said or did. These tools to create deepfakes were first developed for academic purposes, aiming at reducing the cost and complexity of generating convincingly edited videos. Afterwards, these methods have spread across the Internet and have become very popular even among non-expert people, due to their easiness of use and their cheapness. FaceApp, the famous mobile app that everyone was using to post selfies on Instagram, is an example of these available techniques. To use this app you just need to have a smartphone

Figure 1.1: Selection of frames taken from the famous Obama deepfake. Image taken from [2].

and to install it. After that, with a simple click, users are able to have in their phone versions of their younger or their older self; these kind of pictures are none other than deepfakes.

The number of deepfakes available on the Internet is rapidly increasing day by day [3]. Just to give some numbers, consider that in February 2021 there were approximately 60000 of these synthetic videos posted online. These tools, as we will see later, can be potentially dangerous if used for malicious ends: they can be employed, for example, to spread fake news or defame people.

Along with their diffusion, a number of researchers started working on the topic in order to find good ways to detect these videos. In 2019, Facebook started a competition on deepfake recognition, giving money prize to the winners. A new and very large dataset was presented based on several state-of-the-art deepfake generation techniques. This dataset is currently the largest one available for research purposes. A number of brilliant minds joined the competition, given the prize money on offer, pushing deepfake detection forward by a large margin. Right now indeed, given also the help brought by the competition, we can rely on pretty good detectors based on machine learning techniques, but we cannot settle with what we already have. Deepfake generation techniques are improving at very high pace and soon the detectors we have will not be able to recognize deepfakes, whose quality is becoming higher and higher. An example of astonishing deepfake is the Tom Cruise one, shown in Figure 1.2. Almost every month someone comes up with new improved tools for fooling observers and recognition software. Our deepfake detectors will need to be updated soon before it is too late.

Nevertheless there is light at the end of the tunnel since, as experts said

[3], the vast majority of these forged videos/images are not very convincing and they are based on the manipulation of the face. This is because most of these videos are created using the same cheap open source tools which exploit algorithm based on Artificial Intelligence in order to forge videos.

As of now, there are mainly two ways to try and detect these videos: frame-based approaches and time-related (video-based) approaches. Frame based approaches are based on trying to classify single frames as modified or not. Time-related approaches, instead, classify the videos considering a sequence of frames and classify the sequence as deepfake or pristine video.

The goal of this work is to explore time-related approaches for deepfake detection. Since it has not been studied in depth, our work aims at discussing and comparing several video-level models. A further investigation is done concerning artifacts and inconsistencies in deepfake videos: since the datasets available for the problem are generated frame-by-frame, this may generate inconsistencies in the videos and we are aiming at trying to spot them. In order to help us in this task, we employ a Web Interface. The interface is used to collect data from users. We exploit this data to generate a new dataset, with our own custom labels. Finally, we discuss how this dataset has been useful in assessing the performance of our artifacts detection models and the human-level performance on artifacts recognition.

Figure 1.2: Crop of a frame taken from the famous Tom Cruise's deepfake.

1.1 Work description

The main focus of this work is to investigate the temporal aspect of deepfake detection. We propose several video-level models and approaches. The dataset used in this work, for train and test purposes, is the DeepFake Detection Challenge dataset. At first, we present some state of the art models, namely convolutional Long Short-Term Memory. We show different variations of them and different data preprocessings. We describe the training and testing of these models, comparing the performances of all the approaches. Then, some new models are presented: Transformers, which are applied for video classification tasks, and we compare their performance with the convolutional LSTMs. The proposed models are: the Video Transformer Network [4] and the Timesformer [5].

Afterwards, an investigation on the inconsistencies concerning the videos is shown. We identify several recurring inconsistencies. We talk about the flickering effect, how we propose to replicate it and how we detect it. In order to test the models, we perform a survey via a Web Interface to get people to label some ground truth videos. This labeling has been helpful for our final evaluation on flickering detection and in giving insight on what is human performance in detecting inconsistencies in videos.

1.2 Document structure

The rest of the work is composed of seven chapters.

Chapter 2 goes more in deep into the deepfakes field: here we present deepfakes generation techniques, a brief historical introduction and their possible applications and threats. Afterwards, we present a brief description on deep learning techniques.

In Chapter 3 we display the state of the art models for the deepfake detection task. We describe 2 categories: hand-crafted features and deep learning techniques (frame-based and video-based).

Chapter 4 shows the research problem we are tackling, proposing research questions and how we have answered to them.

In Chapter 5 we present the methods and the pipeline utilized, both for deepfake detection and for artifacts (in particular flickering) detection.

In Chapter 6 the employed dataset is presented. We talk about our own synthetic dataset generation for flickering detection and about the Web Interface we have built to collect user data.

In Chapter 7 we assess and compare model performance, both for deepfake detection and flickering detection. We show confusion matrices, Receiver Operating Characteristic curves and other metrics. At the end, we assess flickering detection performance via some indices computed on the collected data of the Web Interface.

Chapter 8 wraps up our work, drawing conclusions and explaining possible future research directions.

Chapter 2

Background

In this chapter we first describe deepfakes, their generation tools, their story and possible applications and threats. Then, we present an introduction on some useful deep learning techniques to understand our work.

2.1 Facial manipulation approaches

Media editing has always been a part of our lives. With the help of deep learning techniques, this field has been pushed forward, creating this new phenomenon of deepfakes.

Deepfakes are starting to spread across the Internet and, as time goes on, we need to familiarize with this kind of techniques. As of now, the vast majority of the employed techniques works on facial manipulation. According to [6], there are 5 different types of facial distortion:

- *attribute manipulation*, a technique that focuses on modifying specific attributes of the face, while leaving the rest unchanged. Examples of this approach can be skin retouching or age/gender modification. Some obtainable results with this approach are shown in Figure 2.1. A tool that performs this techniques is FaceApp[1];

- *face swapping* [7], or face replacement, a technique based on automatic replacement of the face of a person in the source video with another

[1] `https://faceapp.com`

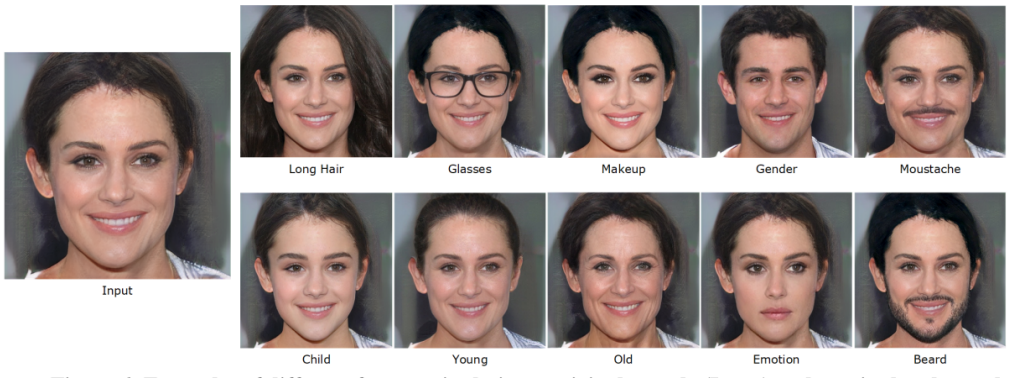


Figure 2.1: Attribute manipulation examples. The image on the left is the original one, while the other ones are modifications. Taken from [6].

face in the target video. Examples are shown in Figure 2.2. Two tools that perform this task are FaceSwap[2] and FaceSwapGAN[3];

- *lip synching* [8], a technique based on modifying a source video such that it generates a clip with a consistent mouth region using an arbitrary audio recording. An example is shown in Figure 2.3. The most famous tool is Wav2Lip[4];

- *puppet mastery* [9], also known as face reenactment, a technique in which the facial expression and movements of the person in the target video or image are controlled by the person in the source video. Examples are shown in Figure 2.4. A tool that performs puppet mastery is Face2Face [10];

- *entire face synthesis*, an approach that is based on creating from scratch faces not existing in real life. Examples are shown in Figure 2.5. A tool that allows to perform face synthesis is StyleGAN2 [11].

[2] `https://github.com/deepfakes/faceswap`
[3] `http://github.com/shaoanlu/faceswap-GAN`
[4] `https://github.com/Rudrabha/Wav2Lip`

Figure 2.2: Face swapping example. We can see the source image on the left (the face to modify), the target image (the image whose features are used to modify the source) in the middle and the result on the right. Image taken from [6].

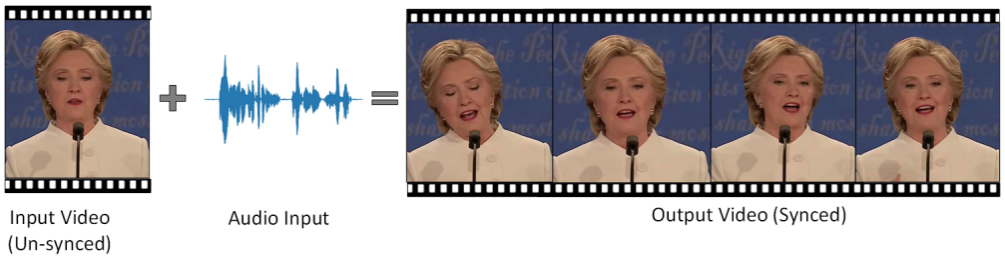


Figure 2.3: Lip syncing scheme. A source video and an input audio are used to synchronize lip with audio in the original video. Image taken from [6].

Figure 2.4: Face reenactment examples. In the picture, we can see the input pictures on top (source and target) and the modified ones in the bottom. Image taken from [10].

Figure 2.5: Face synthesis examples. The first row of pictures shows real existing faces, while on the second row there are synthetic ones. Image taken from [12].

2.2 Deepfakes

In this section we talk about deepfakes, how are generated, their story and we go through their possible applications and threats.

2.2.1 Deepfake generation

The first technique used to generate deepfakes was the deep autoencoder [13], employed in the first ever deepfake tool, named FakeApp.

The deep autoencoder is a deep learning architecture able to achieve very good performance in the image compression or denoising fields that has been adapted to the creation of deepfake videos. The name of the architecture used in this field is Deepfake autoencoder (DFAE). It is composed of two parts, the encoder and the decoder.

As stated in [14], two datasets are required to train these models. The first dataset is composed of original images, or rather the pictures that contain the faces to be replaced. The second dataset is composed of images containing the desired faces that will be swapped in the target video. In the training process, the weights of the encoder are shared between the two datasets (in order to capture common features in the two datasets) while instead two separate decoders, one for each dataset, are used. Encoder+Decoder A is trained on the first dataset and Encoder+Decoder B is trained on the second dataset. In the generation phase, an original image to be modified is given as input to the Encoder+Decoder B architecture, thus obtaining the face swapped picture. A scheme of this process is given in Figure 2.6.

Lately, GANs (short for Generative Adversarial Networks [15]) have become the de-facto standard for deepfake generation. The use of these models has improved the quality of videos generated, enhancing the smoothness of the forged features and creating more realistic eye movement [12]. FaceSwap-GAN is an example of a tool that makes use of GANs.

2.2.2 History

The term deepfake comes from a Reddit user named “deepfakes” that, in the late 2017 [16], created a Reddit section name “r/deepfakes”. In this section, users started to share pornographic content with pornstars swapped with

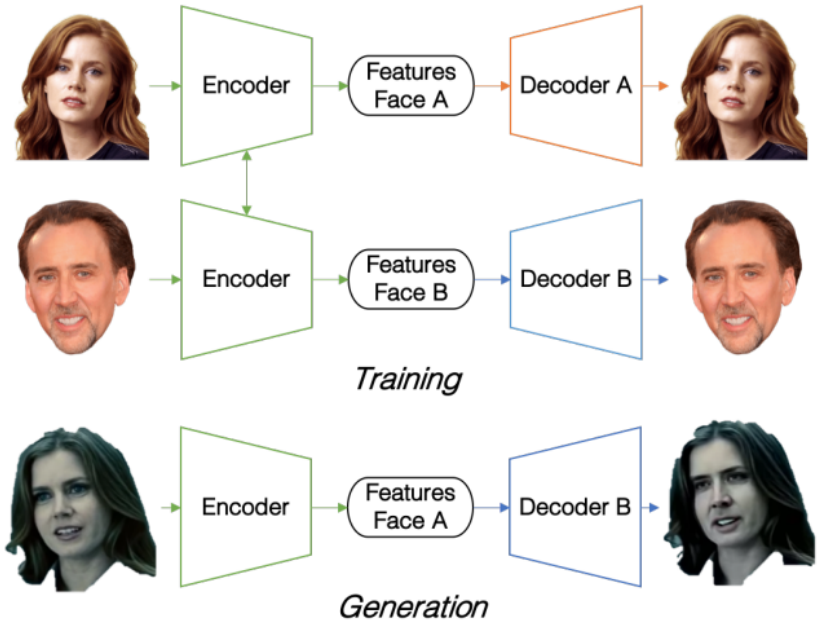


Figure 2.6: Autoencoder training and generation scheme. Image taken from [14].

female celebrities. The page became quite popular and by the time Reddit started taking action the page had more than 90000 active users. Nowadays, these non-consensual porn videos are still the vast majority of the deepfakes on the Web, but celebrities are not the only targets. Reports underlined an increase in videos targeting Youtube or Twitch personalities.

Alongside the spread of pornographic deepfakes, the diffusion of these videos in other fields has not gone unnoticed. Several cases of ID forgery deepfakes have been registered, where people are trying to impersonate someone in order to access a bank account or a Bitcoin wallet [3].

Another field where there has been a noticeable spread of deepfakes is politics. As an example, let us think about the Obama deepfake shown in Chapter 1. During US elections in 2020, the threat of political deepfakes was real and a lot of effort was put into verifying the authenticity of videos circulating on various media.

2.2.3 Applications and threats

Deepfake tools are simply doing what other techniques were already doing in the past, but better and faster. Once trained, these tools just need an image (or a video) in input and they are able to create convincingly forged video. The potential of this technology is quite high. Deepfakes can be used in the acting field, in order to reduce operational and production time. Disney has started doing that by training its own face swapping models based on deepfakes and iterating the operation in order to refine outputs. Soon movie and television production will adapt to this. With this techniques, there will also be the possibility to resurrect and revive characters for fans to enjoy [16].

The application of deepfake techniques is not restricted just to TV shows and movies. Let us just think the video editing potential that content creators on the Web could have available.

Social networks are impacted too; just think of all the possible filters that these tools can make available.

Another potential use of deepfakes is translation services. During 2020, an Indian team has developed a software able to translate videos by translating the audio and changing the lip movement accordingly. If this area is investigated further, we will be able to communicate with people around the world with ease.

Unfortunately, a lot of criminal groups have adapted and have started to use these new tools to nefarious ends: indeed, most of the videos posted on the Web had malicious ends. As described briefly before in the history section, the amount of deepfakes concerning pornography is huge. A lot of deepfakes available are videos where the face of a pornstar is swapped with the one of a famous celebrity. This phenomenon is mainly targeting female celebrities, moving from the entertainment world to the content creation world (like Twitch or Youtube). This is mainly because deep learning techniques need a large amount of images to be tuned and on the Web there are a lot of available images of known personalities. Hence, famous personalities are very easy targets.

More in general, the damage that can be dealt to individuals is huge. These videos can be used to defame people or blackmail them. They can also be used as ID forgery mechanism to be able to fool facial recognition

systems, to access a bank account or a bitcoin Wallet.

Unfortunately, targeting individuals is not the only problem these videos pose. Deepfakes can be used to spread fake news, create warmongering situations by showing fake videos of missiles launched to destroy the enemy state [6] or break diplomatic agreements by showing a major state president giving a speech.

2.3 Deep learning background

Deepfake tools make use of deep learning techniques. Therefore it is useful, in order to better understand our work, to have a basis on the techniques used. Here, we briefly introduce architectures we have used: Convolutional Neural Networks, Recurrent Neural Networks and Transformers. Again, this is just an introduction, to know deep learning techniques more in details see [17].

The basic building block of deep learning is the neural network. A neural network is an architecture constituted by nodes organized in layers. Layers are connected as a directed graph, where the output of one layer is the input of the next layer. The output of a neuron is the weighted sum of its inputs and via an activation function we can see if the neuron has been activated or not. Each link has a weight that needs to be learned with some training procedure. These weights are updated at every training step, trying to minimize a loss function that is defined depending on the specific problem.

There can be potentially infinite variations of a neural network, obtained, for example, by changing the number of layers in it or the number of neurons in each layer. By increasing the size of the neural network, its complexity and learning power increases, but overfitting probability increases too.

Nowadays, the most famous Neural Network architectures are very complex ones, since they are able to solve very complex tasks with enough training data available.

A first example of a network is the Feed Forward Neural Network, where layers are organized in chronological order, such that the output of a layer can only be the input of the following layer.

The Convolutional Neural Network (CNN) is an architecture mainly used to deal with images. The architecture is composed of two parts: the convo-

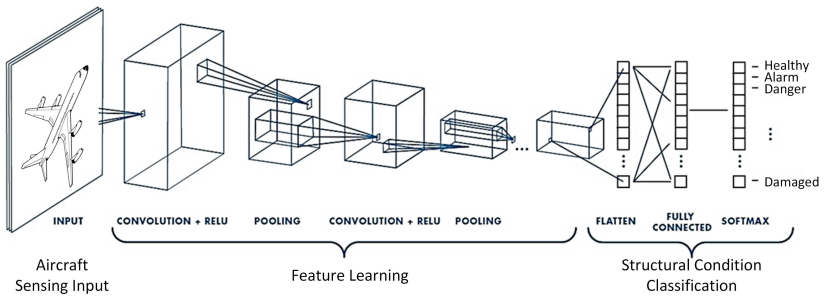


Figure 2.7: General CNN scheme. Here you can clearly see the network divided in two parts: the first, convolutional part for feature extraction; the second, fully connected for classification.

lutional and the feed-forward part. The convolutional part is based, as the name suggests, on the convolution operation. As we can see in Figure 2.7, the convolution is performed in multiple layers and, combined with pooling operations, it is able to reduce the size of the input and extract high-level semantic features of the image. This is crucial in order to correctly perform classification or regression task in images. The second part is a simple feed-forward neural network used to classify the input image. There can be a lot of variations of the CNN, adapting the network to solve other problems, like object localization, but we will not make use of these architectures hence they are not discussed here.

Recurrent Neural Networks (RNNs) are networks built to deal with time series or, more in general, with time-varying input. The term recurrent comes from the presence in the network of recurrent connections, which allows the system to remember old input by "storing" it as an hidden state. They are able to predict the next value of a series, given the information they store and the next input. In Figure 2.8 we can see how RNN deals with time series.

Classic RNNs suffer from the vanishing gradient problem, which means that they are not able to remember old data. New architectures, such as the Long Short-Term Memory (LSTM) [18], have been studied to tackle this problem. In this architecture, the recurrency weight is set to 1 enabling, to a certain extent, the training of RNNs with arbitrarily old data. LSTM architectures were initially thought for Natural Language Processing tasks, but they have been adapted to perform other tasks, like visual question answering or image captioning.

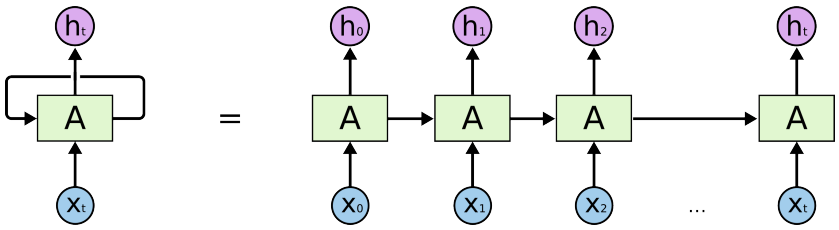


Figure 2.8: General RNN scheme. Here the architecture (block A) has been unrolled to exemplify how input (x_0,...,x_t) is fed one at a time and how hidden states (h_0,...,h_t) are generated.

In the last few years, Transformers [19] have become the de-facto standard for natural language processing tasks. These architecures are very different from classic Neural Networks since they do not make use of any convolution or recurrency. They make use of a mechanism called self-attention to compute the features for a specific input, that is how much a part of the input is important with respect to the other parts of the input.

2.4 Models for video classification

In this section we explain more in detail the architecture paradigms we have employed in our work.

2.4.1 Convolutional LSTMs

Convolutional LSTMs are architectures widely used in the video classification field, mainly for action recognition tasks. As we can see from Figure 2.10 they are composed of three parts: the first part is the feature extraction part, the second is the recurrency part and the third is the classification part. The first part is composed by any network that can extract high-level features: any hand-made CNN, well-known CNN architecture (like VGG, EfficientNet, Resnet) or any self-attention transformer model that computes attention over a single frame (like the Convolutional Vision Transformer [20], or CViT). Every frame is fed to this network and features are extracted. The high-level semantic features are then fed into the recurrency part, which is composed by any layer/cells able to store an hidden state, namely classic RNNs or LSTM/GRU cells. The hidden state of the recurrent part is then

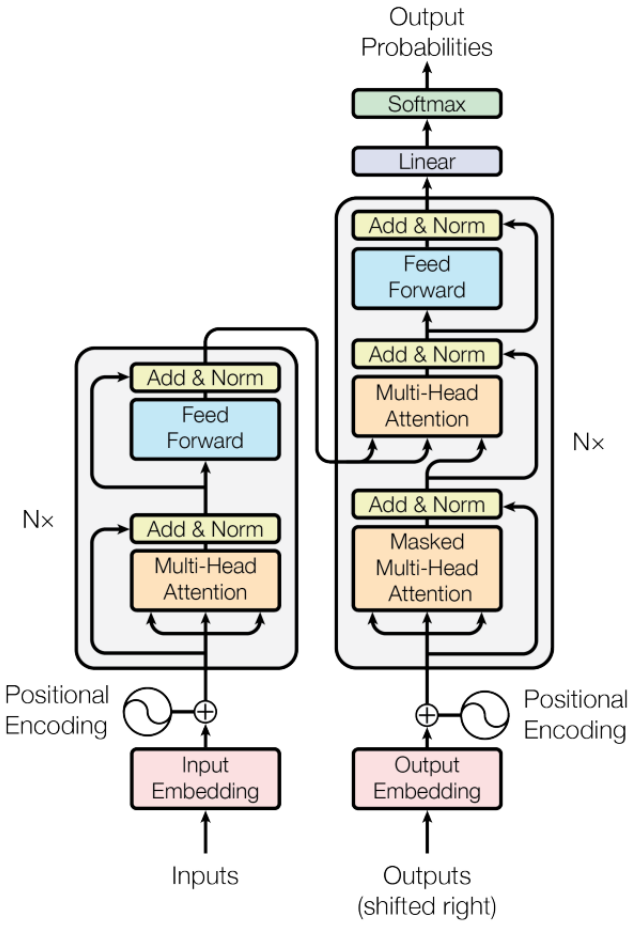


Figure 2.9: Scheme of the Transformer architecture. Image taken from [19].

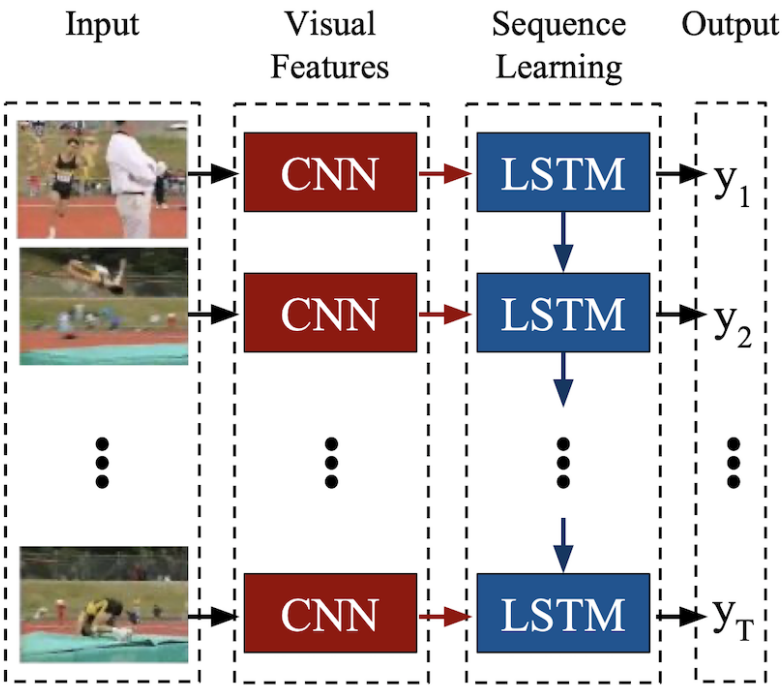


Figure 2.10: Scheme of a convolutional LSTM architecture. Here the feature extraction performed via a CNN. The CNN network is only one, every frame passes through the same network.

fed to the third section, composed of fully connected layers. The last layer has the softmax as activation function and the number of units equal to the number of classes of the task.

2.4.2 Transformers for video detection

In recent years, Transformers have quickly become the state of the art for most of the Natural language processing tasks and now researchers are starting to apply them also to other tasks, like for example Image or Video classification.

CViT is an example of Transformer used for Image Classification and, as we will mention in Chapter 3, people have also tried it in the deepfake detection field.

New Transformer architectures are coming up every day trying to tackle action recognition problems or other video classifications tasks. This is a very

hot topic nowadays since the performance on current most famous datasets (e.g. Kinetics) is not so high and people are still trying to figure out effective techniques. The main problem is mostly related to the fact that these Transformer architectures have a huge number of parameters and therefore training from scratch is not possible. The only feasible things are: smart initialization of the parameters or fine-tuning [21].

Transformers architecture

Transformers are architectures that, without any need of convolutional or recurrent neural networks, are able to solve a lot of different problems. They make use of the self-attention mechanism to perform the encoding (and eventually the decoding) of the input. In this work we only consider the building blocks used in image and video classification. In the classic architecture, there are two parts, the encoding part and the decoding part, but here we consider only the encoding.

Input embedding The input can be a single frame divided into multiple patches (image classification, see CViT), a sequence of frames divided into patches or simply a sequence of frames embedding (video classification, see Timesformer [5] and Video Transformer Network [4]).

Positional encoding The positional embedding layer is required in order to give information to the transformer about the order in the input. Differently from the LSTM, where inputs are fed sequentially (and one at a time) to it, in the Transformer the inputs are fed all at the same time, so it is required to have order information in the input embedding to preserve the semantics given by the order. Transformers do this by using the positional encoding layer, which is a layer that adds to the input embedding a vector, namely positional embedding, with order information. In the original Transformer paper, they used sin and cosine function to represent this embedding. The equation are the following:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{2.1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{2.2}$$

Where *pos* is the position in the sequence, i is the i-th feature in the input, and d is the total dimension of the embedding.

Multi-Head attention module The embeddings are then fed into the main module of the Transformer encoder, the multi-head attention module. Here the self-attention is computed (the main building block is shown in Figure 2.11). The block has linear layers to shrink the size of the input embedding to speed up computation. Then, 3 vectors are defined: Q, the query vector, K, the key vector and V the value vector. As an example to better understand what these vectors are, let us imagine the Q is a query, like a text we input to search a video on Youtube; K are the results of the search and V is the value of the similarity among Q and K. The self-attention matrix is then computed as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK}{\sqrt{d_k}}\right)V \tag{2.3}$$

As we can see, the cosine similarity is used to compute the self-attention, which is then shrunk between 0 and 1 using a softmax layer. In general, more of these blocks are used in parallel (that is why we the module is called "Multi-Head attention", Figure 2.12) to improve accuracy and confidence over the attention generated. Each input of this block is then concatenated and shrunk again to become the final encoding of our model. In video detection, we can think the self-attention between part of a single frame (in image classification) or attention between the same spatial patch across different frames (in video classification).

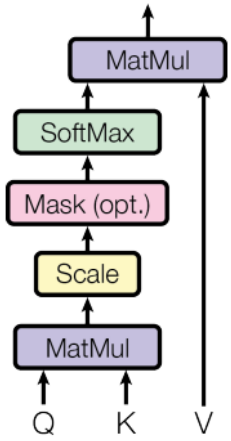


Figure 2.11: Scheme of the scaled dot product attention. Image taken from [19].

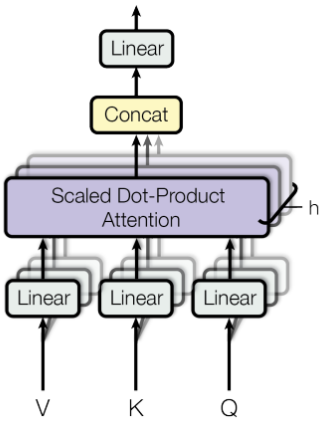


Figure 2.12: Scheme of the Multi Head attention layer; h is the number of heads. Image taken from [19].

Chapter 3

Related work

In this chapter we present the state of the art for deepfake detection. All the techniques can be summed up in three different categories: hand-crafted features, frame-based deep learning techniques and video-based deep learning techniques.

3.1 Hand-crafted Features

This first approach is characterized by the manual selection of the features to use for discriminating if a video is original or fake. The main advantage of using hand-crafted features is the interpretability of the results, as [22] said.

As a new hand-crafted feature approach Siegel et al. [22] proposed 3 different possibilities for hand-crafted features: eyes blinking, mouth region and image foreground region.

Yang et al. [23] presented a study over the inconsistent 3D head poses that are originated from the face swapping algorithms. They used Support Vector Machines to classify images as pristine or fake. Even if this technique is good for detecting face swapping, it performs pretty poorly on other deepfake generation techniques, like puppet mastery of lip syncing [24].

McCloskey et al. [25] showed their study over the colors of an image generated by GANs, explaining that the patterns present in these images (in terms of colors) are quite different from the ones generated by cameras.

Gu et al. [24] proposed a study on the detection of deepfakes generated by different types of techniques. They claimed that in fake videos there are

inconsistencies in the facial movements and expressions - whether it being a lip synced video (mouth inconsistent with the rest of the face) or a puppet-master video (the expression is inconsistent since an impersonator is choosing what to say).

Korshunov et al. [26] have presented a way to detect lip syncing videos, considering audio-visual features for every image and adopting an approach based on Principal Component Analysis and SVM as a classifier.

3.2 Deep learning approaches

In this section, the described methods are able to learn by themselves the features during training.

3.2.1 Frame-level deepfake detectors

Frame level detectors are based on classifying one single frame at a time. A lot of works are based on CNN architectures [27, 28, 29, 30, 31, 32, 33], and all the techniques make use of face detectors since it has been shown that cropping around the face helps improving the accuracy of the detectors [33].

Nguyen et al. [27] proposed a method based on capsule networks. At first, a CNN used to extract features is used. Features are then fed to the capsule network [34] in order to correctly perform the detection.

Afchar et al. [28] presented Mesonet architectures as detectors (namely Meso-4 and MesoInception-4) to deal with deepfake detection. They trained their architectures on the Face2Face generated videos, obtaining an accuracy of 0.96.

Rossler et al. [33] showed some CNN approaches to detect fakes using their own dataset [35]. The dataset is a collection of different deepfake generation techniques used (in 2019), like Face2Face and FaceSwap. They showed that XceptionNet outperforms other approaches such as Mesonet [28] and that finetuning is the way to go, considering the large amount of parameters these architectures have to learn, obtaining as accuracy 0.98.

Li et al. [30] used other popular CNN architectures, such as VGG and ResNet. They generated their own training data by taking images from the Internet and applying deepfake generation algorithm to some of these

samples. The accuracy obtained on UAFDV and Deepfake-Timit datasets [36] was respectively 0.97 and 0.99.

Amerini et al. [37] proposed a CNN architecture based on optical flow computation [38]. The idea behind it is that the optical flow of a fake video in different from a natural video, so a model trained on it would detect it pretty easily. The architecture they presented worked by computing the optical flow and by feeding this vector field to a CNN, called Flow-CNN, using as backbone VGG and Resnet.

Other authors [29, 32] proposed ensemble methods to try and improve accuracy of frame-based classifiers.

Bonettini et al. [29] presented an ensemble of CNNs (EfficientNets) and a way to produce attention over the parts of the image, showing which ones are the most important for classification.

Rana et al. [32] proposed an ensemble between several popular CNN architectures and proposed a method to also learn a way to combine different predictions, obtaining pretty good results on Face Forensics dataset, with over 0.99 overall accuracy.

Wodajo et al. [39] used the Convolutional Vision Transformer [20] to perform deepfake detection. The architecture is composed by, at first, a VGG-like convolutional network. Then, the various feature maps are flattened and fed into a transformer encoder network, which computes, the final encoding used for classification. They obtained discrete results: 91% in the DFDC dataset but other bad results on other datasets.

Jeon et al. [31] showed another self-attention model, concatenating FineTune Transformer(FTT) model with a convolutional pretrained network. They obtained 0.97 in Face2Face and Deepfake dataset. They also did an ablation study on the FineTune Transformer showing how using the FTT improves performance with respect to not using it.

The winner of the Deepfake Detection Challenge [1] [2] obtained 0.82 on the private test set, showing how it is still difficult to detect deepfakes. He used CNN-based architectures, obtaining the best results with EfficientNetB4, and performed strong augmentation in the training images (e.g. completely re-

[1] `https://www.kaggle.com/c/deepfake-detection-challenge/discussion/145721`

[2] `https://github.com/selimsef/dfdc_deepfake_challenge`

moving parts of the face).

3.2.2 Video-level deepfake detectors

Video level detectors are based on trying to classify the video as a whole, considering multiple frames. The basic idea is trying and spotting pixel artifacts across frames.

Other researchers [14, 40, 41, 42, 43] have tried to used RNNs in combination with CNN architectures. The CNN part is used for feature extraction, while the RNN part is used for combining the features of chronogically ordered, consecutive or not, frames.

Güera et al. [14] used their own deepfake dataset. They used Inception-V3 as backbone. The performance was very good on their own test set since they obtained more than 99%.

Chinthia et al. [42] proposed their own network using XceptionNet and using bidirectional LSTMs. They then diversified the model trainings by trying different loss functions, namely cross-entropy and KL divergence, also called relative entropy. They then proposed an ensemble of the previous two loss functions. The results they obtained were pretty good, obtaining 0.93 of pristine videos correctly classified and an overall 0.84 of fakes correctly classified.

Li et al. [43] proposed the use of conv-LSTM to detect eye blinking pattern inconsistencies. In their work they claimed that deepfake techniques used at that time did not make people blink regularly and they tried to estimate the pace of blinking in videos to detect fake ones. Unfortunately, this method soon became obsolete as new deepfake techniques added eye blinking to the forged videos [24].

Sabir et al. [41] showed that RNN architectures performed worse on Face Forensics dataset with respect to current state-of-the-art CNN-based methods.

Ganiyusufoglu et al. [44] presented 3D-CNN approaches to deepfake detection, using R3D or ID3 (respectively the 3D version of Resnet and Inception) architectures. This methods showed a pretty good generalization performance, computed by training an all except one deepfake generation method at a time and testing the model on the left-out method.

Chapter 4

Research problem

Our effort in this work is to answer research questions about deepfakes in the time domain. In particular, we follow 2 different approaches: investigation on video-level artifacts and investigation on video-level models. In the following we report the research questions goal of this work.

1. *What kind of video-level artifacts can we identify in deepfake datasets?*

   Deepfake videos are generated using frame-based techniques and aggregated into a single video without any preprocessing. This process may generate inconsistencies in the crafted deepfake. Furthermore, deepfake generation works best with images where there is a face in front of the camera; this means that when the face recognition fails, it generates inconsistencies across the video. Studying the DFDC dataset and using a Web Interface, we define several inconsistency labels: flickering, wrong proportions, irregular facial traits, obfuscated face, glasses inconsistency and color inconsistency. We use collected annotations from the Web Interface to build a new dataset.

   - *Which one can we detect automatically?*

     Detecting these inconsistencies can be helpful for deepfake detection. Unfortunately, in order to train models on these inconsistencies, new ground truth is needed. We collect some samples with the required labels, but the collected data is not enough to be used for training. Luckily, some inconsistencies can be synthetically replicated in order to try and spot them. This was the case

for flickering in our work, where we present a way to generate a dataset to train models to detect it.

- *What is human level performance on detecting coherently inconsistencies?*

  Alongside with the automatic mechanism to detect these inconsistencies, we test human level performance on labeling the dataset. We show statistics related to the agreement among the raters and compare them with the performances of our flickering detectors.

2. *Can video-based models outperform frame-based classifiers?*

   Video-level detectors have been recently studied and employed for deepfake detection. Studying the video as a whole may be useful to spot some pixel artifacts in the frames. As of now though, video-level models are outperformed by frame-based classifiers. In our work, we explore several video-level models and approaches to try and better characterize this field: we show different convolutional LSTM approaches and newly unseen video-level Transformers.

   - *Can Transformer models reach state of the art results on video-based deepfake detection?*

     Transformers are becoming the de-facto standard in NLP tasks and in several computer vision tasks. Recently, researchers are starting to apply them to video classification problems. In our work, we explore different Transformer approaches, showing that they can reach the performance of our state-of-the-art convolutional LSTM models.

Chapter 5

Proposed Methods

In this chapter we describe the proposed approaches for deepfake video detection and for artifacts detection.

5.1 DeepFake detection

In this section we define the employed approaches for deepfake video detection. We described the general pipeline, moving then to describe in detail each single part of it.

5.1.1 Approach overview

Deepfake detection is a binary classification problem where given an input video a label must be assigned to it. In our problem the labels used are REAL (or label 0) and FAKE (or label 1). The FAKE label is associated to deepfake videos, while the REAL label is associated to pristine videos.

As the works presented in Chapter 3 show, the best way to proceed is depicted in Figure 5.1.

We first preprocess the input video by extracting some of the frames, typically 30 or less is enough to get sufficient information. Afterwards, for each frame, only the area around the face is considered. The sequence of frames is then fed into the detector block, which can be any video-level model like convolutional LSTMs or Transformers, which outputs the predicted label for that specific input video.

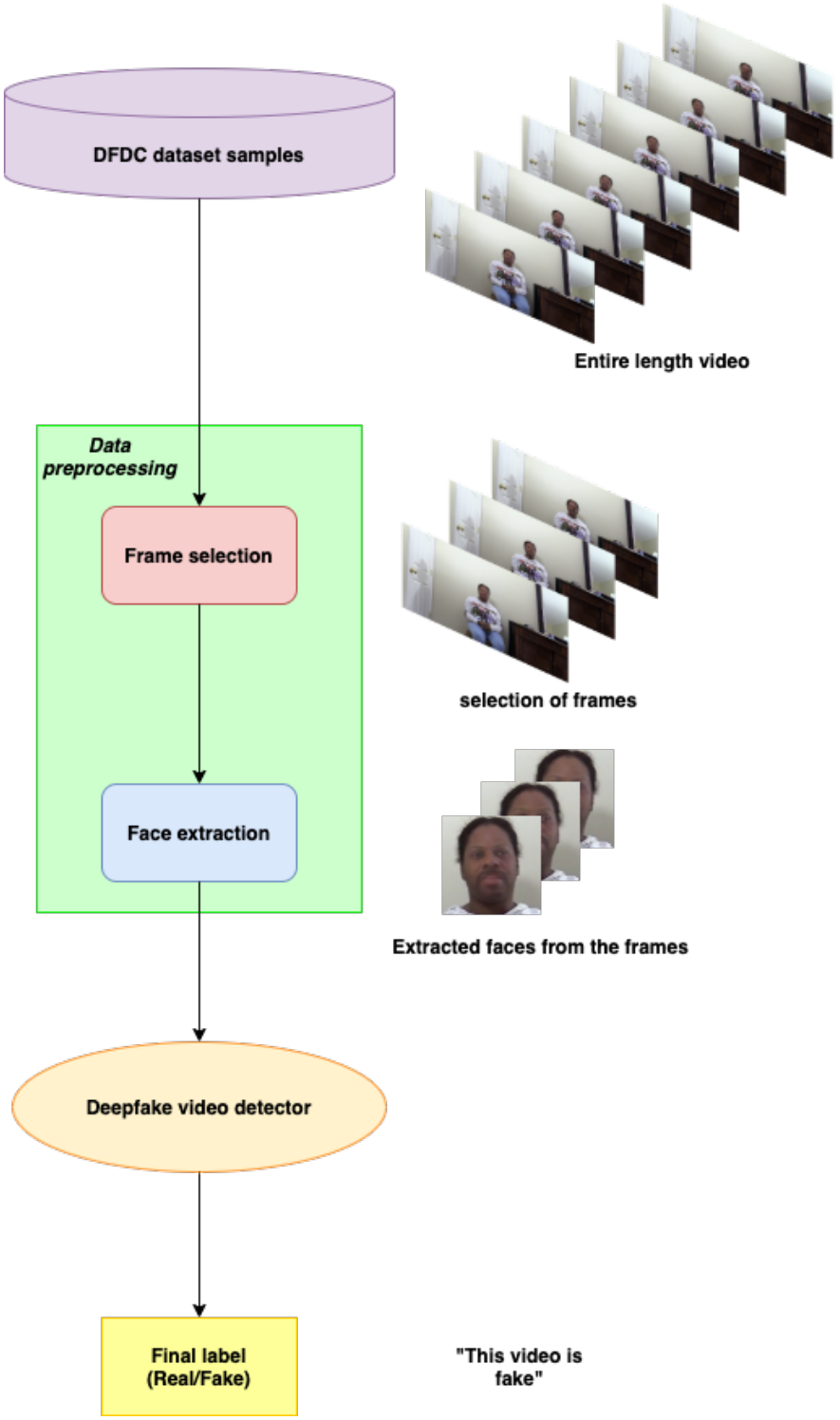


Figure 5.1: Scheme of the adopted pipeline.

5.1.2 Data preprocessing for classic deepfake detection

Before training the model, a preprocessing step is necessary. This way of proceeding has been the go-to approach by all works described in Chapter 3, correctly balancing computational cost and accuracy, having sufficiently informative data to learn how to correctly classify these videos.

Blazeface As [33] explains, focusing on other region of the image rather than the faces in the video is not going to help classification performance. This is because, in the current datasets and with the current open-source tools available (like Face2Face, FaceSwap, etc...), the modifications are performed only in the face area.

In our work we employ a face detector named Blazeface [45] with which we are able to crop the area around the face in every frame consistently for every video (frontal camera face detection accuracy is 0.986).

Selection of frames

We select 15 frames evenly spaced across every video in the dataset, in order to have samples on the entire length of the videos. With this approach we have the possibility to capture, potentially, much different head poses and face expressions and therefore it is easier for the architectures to spot some inconsistencies in the video.

Croppings

Here, the employed cropping are presented:

- **Detection adjusted to** 224×224 **crops**. For each frame, the most confident face detection is considered; each of them, that in principle may have arbitrary dimensions, is adapted to a region in the frame of size 224×224, always centered around the face.

  We choose 224×224 since it is a resolution that can be used in many different architectures without the need of rescaling and, at the same time, it is able to capture enough information while keeping computational time low.

Figure 5.2: Examples of 224×224 *croppings around the face.*

The adaptation is performed simmetrically, adding/removing the same number of pixels to both sides along the x (width) and y (height) dimensions. In the case this is not feasible (due to the fact that the region reached the frame borders), the adaption is made such that the cropped region is still 224×224, adding the quantity we could until the frame border is reached and the remaining quantity to the other side.

In Figure 5.2 we can see examples of this cropping while in Figure 5.3 we show the entire sequence (15 frames) of a sample.

- **Detection adjusted to** 380×380 **crops**. The crop generation is the same as the 224×224 crops, but here, instead, we adapt to a larger crop of size 380×380. We choose this size as a comparison to the 224×224 one since it is able to capture more information of the frames.

  Examples of this cropping are shown in Figure 5.4.

- **Detection with a small margin**. The last cropping we propose considers only the face detections given by Blazeface detector, adding a small margin around them. The margin is computed as: $margin = 0.2 \times detection\_width$, where detection_width is the width of the detection in terms of pixels. This quantity is simmetrically added to all 4 points of the detection. Then, for every frame, only the face detection with highest confidence score is considered.

Figure 5.3: An example of 15 evenly spaced frames across one sample video, 224×224 *croppings around the face.*

Figure 5.4: Examples of 380×380 *croppings around the face.*

Figure 5.5: Mosaic of detections with small margin. As we can see the detections are different in terms of size and therefore they need to be rescaled.

This approach generates differently sized and, in general, not squared detections. This a key point since these images need a rescale without keeping the aspect ratio.

Examples of this cropping are shown in Figure 5.5.

Final preprocessing

Before giving input samples to the architectures, some additional preprocessing is needed. A rescaling step(to the target dimension) is required for all the detections with small margin, while no manipulation is performed on 224×224 or 380×380 crops.

Furthermore, in our work, we compare and test several architecture, each one requiring different preprocessing. The performed additional preprocessings are the following:

- For the architectures with Inception-v3 as backbone, we rescale the RGB values ([0, 255]) to the range [-1, +1];
- For the models using EfficientNetB4, augmentation as used by [29] is needed, coming from the winner of the DFDC [1].

[1] `https://github.com/selimsef/dfdc_deepfake_challenge`

5.1.3 Detection models employed

In our work, we propose different video-level techniques to perform deepfake video detection, from the state-of-the-art methods to newly unseen ones. We explore the utilization of convolutional LSTM networks, Video Transformer Network and Timesformer. These last two approaches are state of the art for video classification and therefore we wanted to further explore deepfake detection and see what these models can bring up to the task.

Convolutional Long-short term memory

This architecture is the most commonly used to perform video-level deepfake detection. The first convolutional part is used to extract high level features from every single frame, while the LSTM layer is used to extract temporal features across different frames. Then, after the recurrent part, fully connected layers are used to perform classification.

Backbone employed For our work, we use Inception-V3 (as proposed by [14]) and EfficientNetB4 (since it is one of the best performing frame-based architectures on the DFDC dataset [29]). We decide to keep Inception-V3 for the LSTM models although performances with EfficientNetB0 and EfficientNetB1 are comparable.

Inception-V3

Inception-V3 is an architecture that has been presented by Szegedy et al. [46] in 2015. It has been proposed, alongside other bigger CNN architectures, mainly to increase efficiency during training and to keep parameter count relatively low, which are two big bottlenecks in big data scenarios. In our scenario, this comes particularly useful since we have to deal with up to 100,000 videos and more than 1 million images every training, so training time is definitely the bottleneck we need to take care of in our work.

Inception-V3 in newer version of the original Inception network [47]. The basic idea behind the original architecture is to try and optimize in terms of parameters (and therefore in terms of computational time) large $n \times n$ convolution and, at the same time, propose a way to apply different convolutional filters (1×1, 3×3, 5×5) to the same input volume. Large convolutional

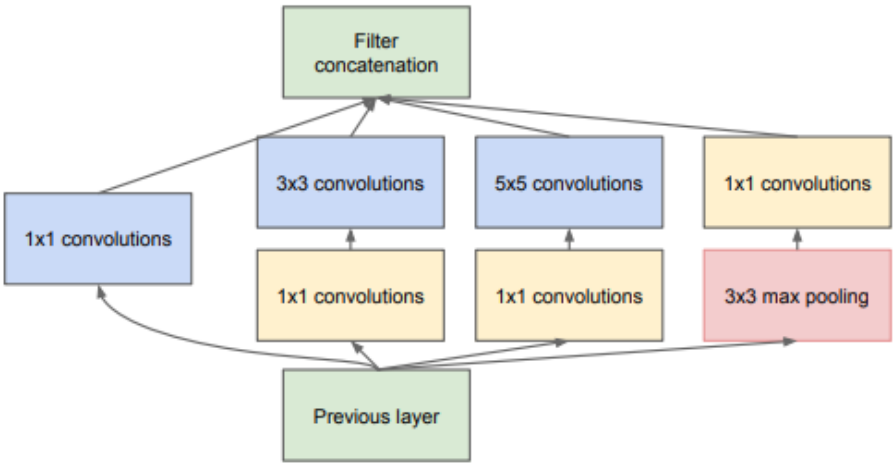


Figure 5.6: Base module of the Inception architecture. Multiple units of this module are used in the network. Image taken from [47].

filters are better since they are able to capture more dependencies, since they capture a larger area of the image, but at the same time they require a lot more parameters and the computation becomes a bottleneck.

The basic building block of the Inception network is shown in Figure 5.6.

In [46] a further optimization has been proposed, which consist in factorizing large convolution into smaller convolutions and thus obtaining another parameter reduction. For example, if we consider a 5×5 convolutional filter, that can be reduced to a sequence of 2 layers with 3×3 convolutional filters, which in the end obtain the same result as 5×5 convolutional filters but with 28% less parameters. More in general, we could factorize $n \times n$ convolutional filters with $n \times 1$ filters followed bu $1 \times n$ filters. An example scheme is shown in Figure 5.7. This approach, as [46] says, has been proven to be particularly successful with medium sized input volumes (width/height between 12 and 20, and using 7×7 convolutions).

EfficientNetB4

EfficientNetB4 is a CNN of the family of the EfficientNet networks [48]. This group of architectures is composed of several models, scaling in complexity and number of parameters, from EfficientNetB0 to EfficientNetB7 (recently an EfficientNetB7-v2 has been released). EfficientNet architectures have been

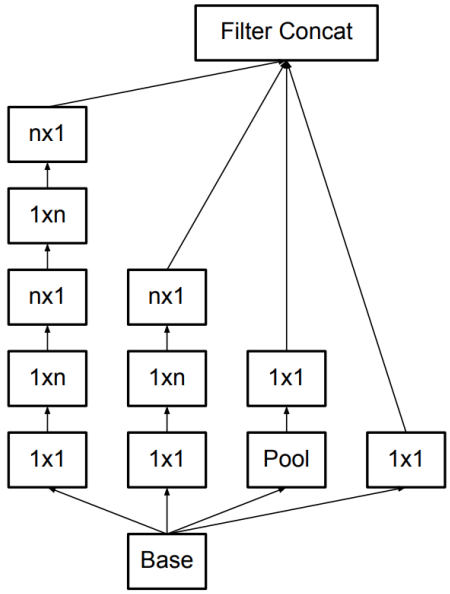


Figure 5.7: Improved base module to perform $n \times n$ convolutions of the Inception architecture, used in Inception-V2 and Inception-V3. Image taken from [46].

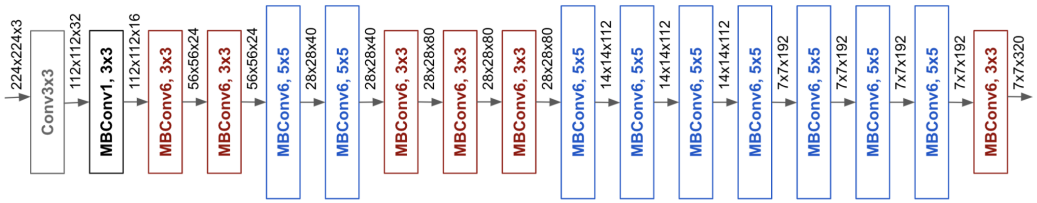


Figure 5.8: Architecture scheme of the EfficientNetB0 network.

studied in order to correctly balance number of parameters, computation and performance, demonstrating how they can simply build a network that is scalable without dropping the performance. In Figure 5.8 we can observe EfficientNetB0, the most basic EfficientNet architecture.

Proposed architectures

The proposed Convolutional LSTM architectures are 2:

- Inception-V3+LSTM, with Inception-V3 pretrained on Imagenet. This architecture has 35 millions parameters. It receives an input in $R^{T \times W \times H \times 3}$, where T are the number of frames, W is the image width and H is the image height. Every frame (dimension $W \times H \times 3$) is fed into a

shared-weights Inception-V3. Since Inception-V3 is fully convolutional, it can accept any kind of input-shape, but we mainly tried 224×224 and 380×380 resolutions. Inception-V3 generates a feature volume of $F \times F \times 2048$, where F is the dimension of the feature maps that depends from the input resolution used (5 for 224 and 7 for 380). The feature volume is too big to be directly fed to the LSTM. We initially tried giving the entire embedding to the LSTM but the model was not able to learn and we rapidly discarded this approach. Therefore, we opt for a Global Average Pooling layer (GAP) in between the CNN and the recurrent part. Like this, we are able to shrink the embedding vector dimension to "only" 2048, which is still big, but affordable. Models are able to learn with this approach and therefore no further dimensionality reduction is needed. The 2048-feature vector for each frame is then fed into the 2048-unit LSTM layer (with a 0.3 chance of dropout). The last hidden state on the LSTM, namely the one obtained after the last frame iteration, is fed into a fully connected layer of 1000 units with a tan-h activation function. The last layer is the classification layer and has 2 units and a softmax activation function. In figure Figure 5.9 an example scheme of the network is presented;

- EfficientNetB4 + LSTM, with EB4 weights taken from [29] (pretrained on DFDC), and EB4 backbone being frozen during training. The input of this network is still in $R^{T \times W \times H \times 3}$, where the single frames are fed into a shared-weights EfficientNetB4. EB4 accepts 224×224 input resolutions, and generates a feature vector (already passed through a GAP layer) of 1792 units for each frame. The input of the recurrent layer is $1792 \times T$, where T is the number of frames we used for the training (in this case 15). The remaining part of the network is the same as the one described for the Inception-V3 case.

Transformers

Transformer architectures have not been widely explored for deepfake detection task, and therefore, in our work, using the DFDC dataset (100k videos) [49], we test some of these networks. For our work, we propose two

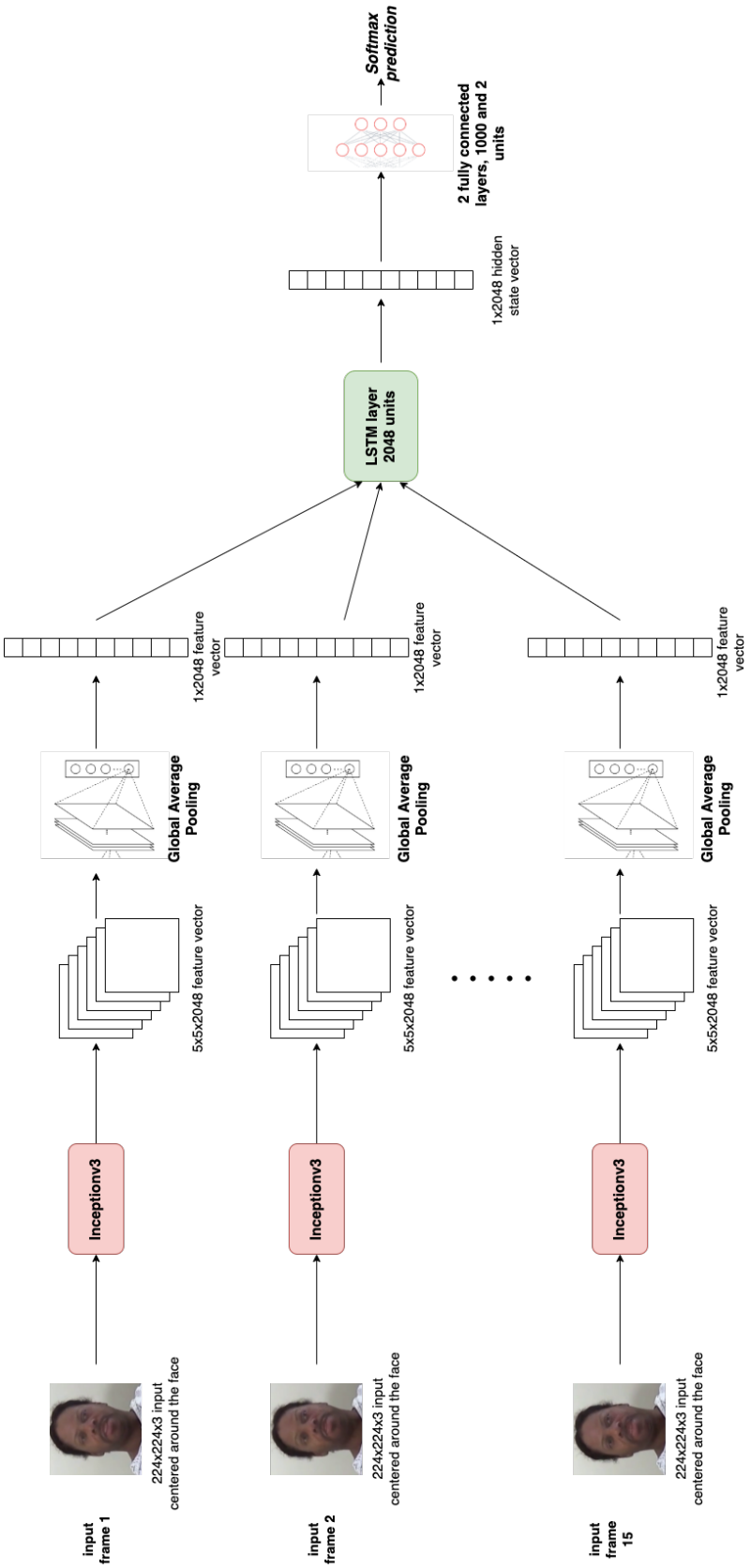


Figure 5.9: Scheme of the convolutional (inception-V3) LSTM architecture used for 224 × 224 crops.

video-level state-of-the-art architectures: Video Transformer Network [4] and Timesformer [5].

Video Transformer Network This architecture is taken from a github repository[2] and can be seen as a revisitation of the Convolutional Vision Transformer. Neimark et al. [4] used this type of architecture in their recent work. The github repository does not aim at reproducing the paper (also since the code was released far before [4] came out) but can be seen as an implementation of it.

Thea architecture takes as input N RGB frames with the dimension of 224×224. The network is composed of two parts: the feature extraction part and the self-attention module. In [4] they have opted for different feature extraction methods, such as ResNet or other self-attention models like the Vision Transformer. In our work we opt for Resnet50 (using the original code) and EfficientNetB0, pretrained on Imagenet. We choose EfficientNetB0 since it is a small, well-performing architecture.

After the feature extraction of every single frame, the positional embedding is computed and added to the embedding of every frame. Everything is then fed to the self-attention modules (Multi-Head attention). The MHA module has 16 parallel heads and 3 consecutive modules to compute the encoding. Every module encoding is used to compute the final encoding, using the encoding from the previous MHA layer as the new query vector for the next encoder, while V and K vectors remain the same across all the passages. The middle frame initial embedding is used as query value Q.

Every frame embedding is split into 16 parts (1 per each head) in order to speed up the training process. At the end of the encoders, all the computed attentions are concatenated once again and normalized. To finally perform classification, a fully connected layer (1024 units down to 1) with the sigmoid activation function is employed in order to classify the input as REAL or FAKE.

A scheme of the architecture is shown in Figure 5.10.

Timesformer The second architecture is the Timesformer [5]. The architecture presented in the paper is an attempt of completely getting rid of the

[2] `https://github.com/ppriyank/Video-Action-Transformer-Network-Pytorch-`

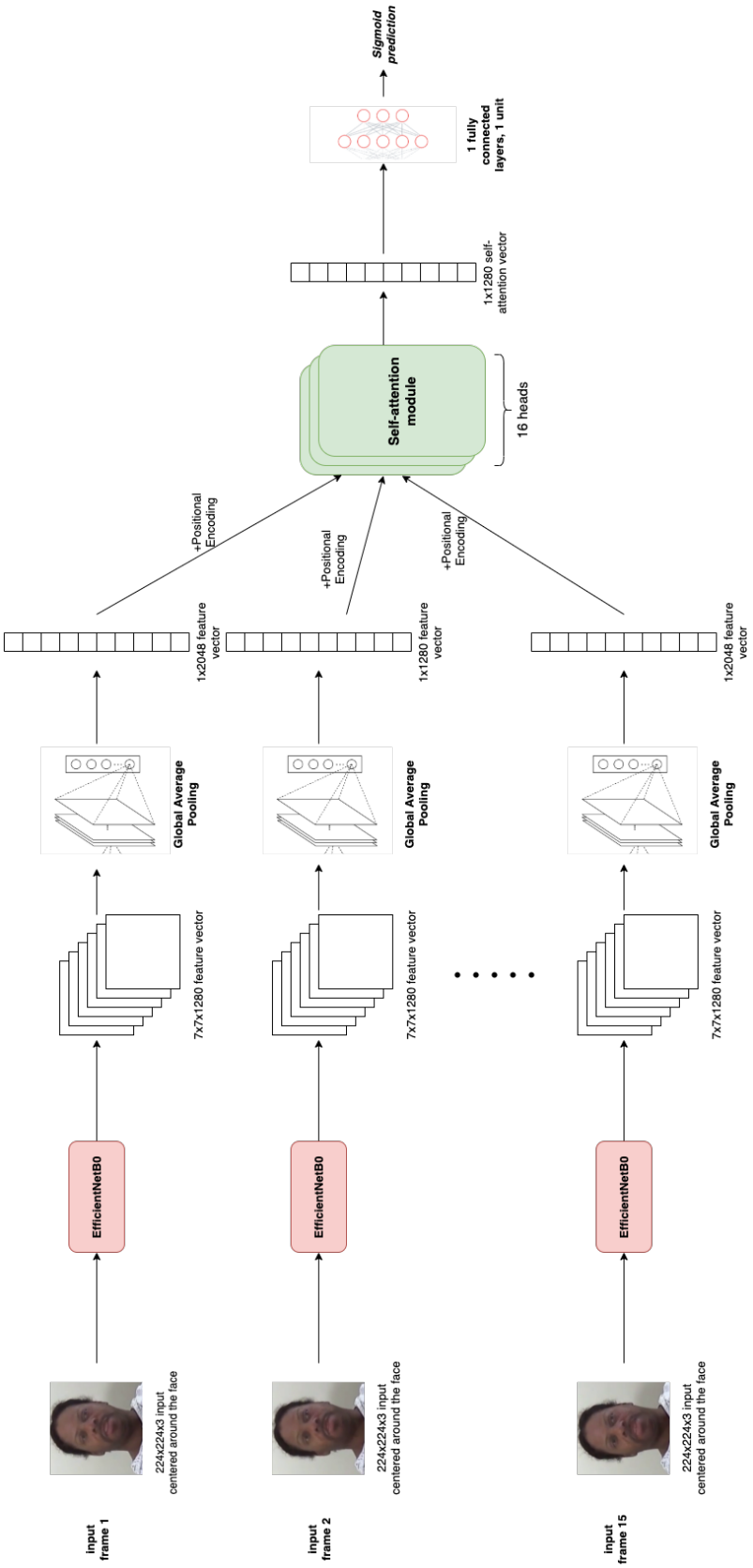


Figure 5.10: Scheme of the Video Transformer Network architecture used for 224×224 *crops.*

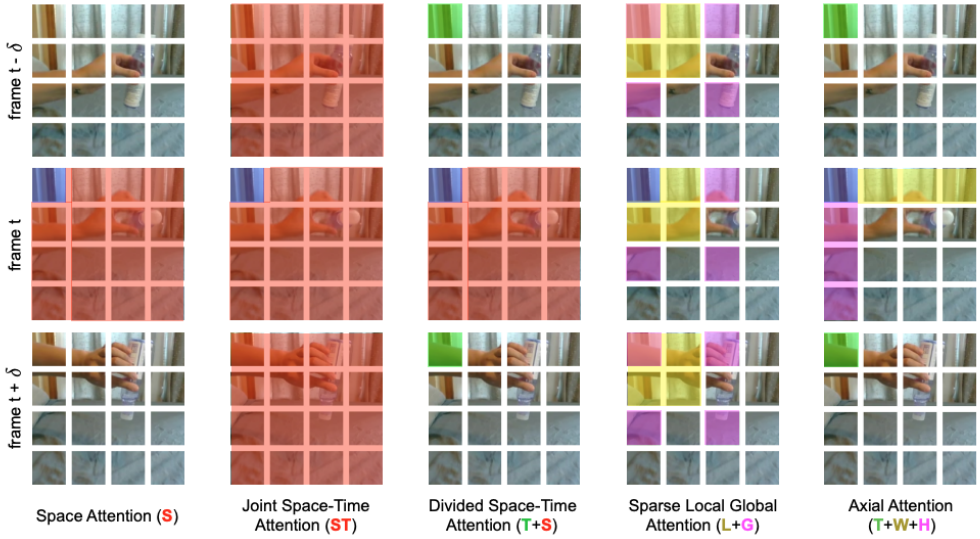


Figure 5.11: Different types of attention, as shown in [5]. In our work we use divided space-attention.

feature extraction part, replacing it with a spatial self-attention instead.

The architecture takes as input N RGB frames (in our case 8) with the dimension of 224 × 224. Every frames is then divided into patches of dimension $P \times P$ (where P is an hyperparameter, in our case it is equal to 8). Every patch is then associated with a learnable embedding composed of a learnable matrix and a positional embedding. These embeddings are then fed to the encoder, which is composed of N encoding blocks. Each one of these computes the Q, K, V values for the next encoding block.

Different attention types have been proposed in the paper: spatial only, divided space-time, joint space-time, sparse local-global attention and Axial. An example to better clarify which are the involved patches in every type of attention is shown in Figure 5.11. For our work, we propose divided space-time (Figure 5.12), as a good trade-off between computational time and accuracy.

For this architecture, or, more in general, for these very big Transformer architectures, fine-tuning is fundamental in order to achieve decent performances. Indeed, we are forced to choose 8 frames since it is the only feasible fine-tuning, in terms of space and time constraints. In our work we show the results of Timesformer on the DFDC dataset, pretrained on Kinetics-600

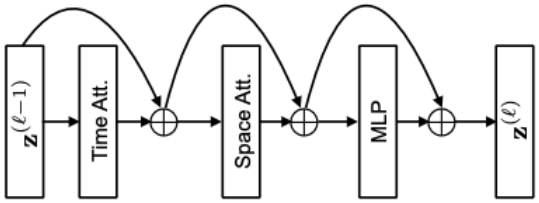


Figure 5.12: Scheme of the self-attention module employed. z is the embedding vector. The embedding vector at encoding layer l-1 is used to compute the input to the next encoding layer l. Image taken from [5].

[50] dataset, one of the best dataset for action recognition.

5.2 Artifacts detection

Artifacts detection is something that has been widely explored to spot deepfakes. On the other hand, video-level artifacts have not been studied in depth and no work has yet tried to explicitly spot these artifacts in videos. A big problem towards this direction is the lack of available datasets to train models to detect such inconsistencies.

Therefore, our aim is to collect new data on deepfakes regarding these inconsistencies and to propose a way to spot some of them.

5.2.1 Approach overview

Artifacts detection is a more general multi-class classification problem where given an input video one label must be assigned to it. In our problem the labels are defined by different classes of inconsistencies that may occur in videos. The assigned label is one of the following errors if detected, or NOTHING if no error is detected. The video can still be FAKE, nevertheless here we are aiming at spotting video-level artifacts. All the inconsistencies are concerning the face area, since the video manipulation occurs only in that area. These inconsistencies are due to the fact that the deepfake generation occurs frame-by-frame and no post-processing to correct it is performed.

Our approach is based on the fact that videos on the Internet show these kind of artifacts. This hypothesis is fair because, as stated in Chapter 1,

most of the uploaded videos are forged using the same cheap open source tools which generate not very convincing deepfakes.

5.2.2 Artifacts definition

Analysing videos from the DFDC dataset, we have come up with several artifacts labels. In this section, we introduce them.

Flickering The "flickering" phenomenon can be described as a sequence of consecutive frames across which some face traits abruptly change. This phenomenon happens for two causes:

- deepfake generation occurs frame-by-frame;
- the face detector fails at recognizing the face in the image. This may happen because the person is too far away from the camera, the video is too dark or the face is turned.

A peculiarity of flickering is that it does not have a particular pattern, therefore it means that there can be flickering across the whole video or in just a couple of frames. Examples of flickering are shown in Figure 5.13 and Figure 5.14.

Figure 5.13: Example of the flickering effect in three consecutive frames. Here the effect is pretty strong and noticeable, even frame-by-frame. In the middle frame we can see how the deepfake generator has failed in detecting the face and therefore in generating the modified face for that frame. Sample taken from the DFDC dataset.

Obfuscated Face There are several videos where the deepfake generation process has made some mistakes and therefore we may have sequences of

Figure 5.14: Another example of the flickering effect in 20 consecutive frames. We can notice how the beard and eyebrows change throughout the sequence. Sample taken from the DFDC dataset.

frames where the face is completely obfuscated. An example is shown in Figure 5.15.

Color Inconsistency The color inconsistency phenomenon occurs where the deepfake generation has created a facial area with inconsistent colors with respect to the rest of the face. An example is shown in Figure 5.16.

Face Proportion Inconsistency The face proportion inconsistency is due to not perfect deepfake generation, where the deepfake generation creates inconsistencies (also across multiple frames) in the proportions of the face traits. An example of this may be a nose too big or eyes too small. A sample frame is shown in Figure 5.17.

Irregular facial traits This category is constituted by all the videos where the deepfake generation procedure has crafted faces with non-natural facial

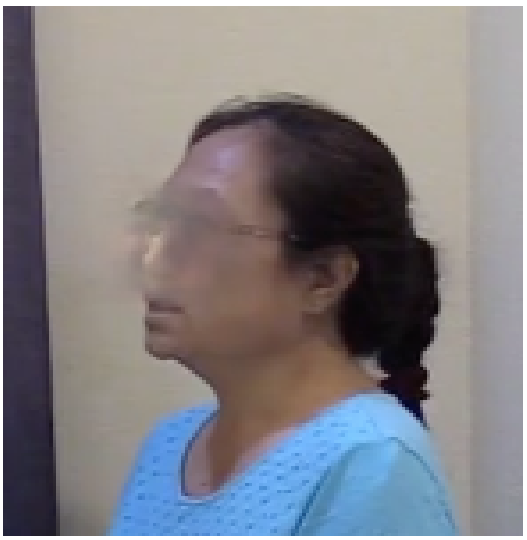


Figure 5.15: Example of an obufuscated face sample. Sample taken from the DFDC dataset.

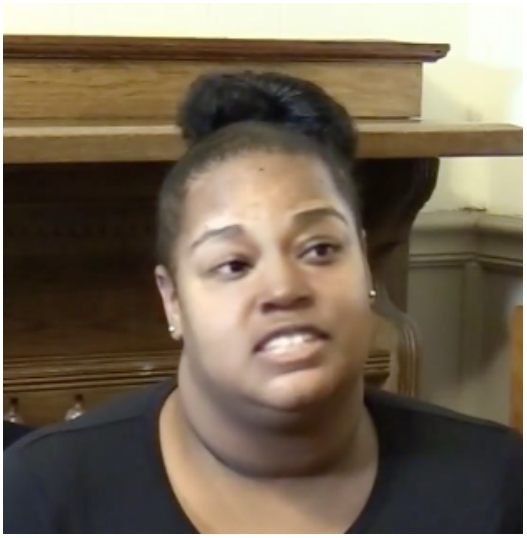


Figure 5.16: Example of a sample with color inconsistency. We can see how the color in the eye area is different from the rest of the face. Sample taken from the DFDC dataset.

traits. An example of this maybe a person with double-mouth, clearly artificial traits and so on. An example is shown in Figure 5.18.

Glasses Inconsistency This category is constituted by all the videos in which there are inconsistencies in the glasses. Since the face manipulation is performed in the face region, this generates inconsistencies in the glasses

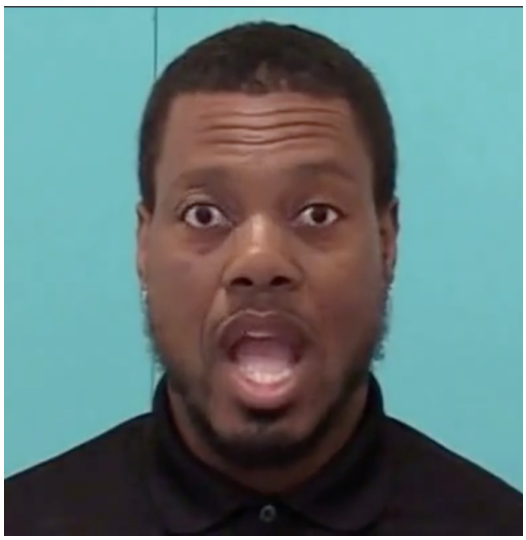


Figure 5.17: Example of a sample with wrong proportions. Here we can notice that the nose is too big with respect to the rest of the face. Sample taken from the DFDC dataset.

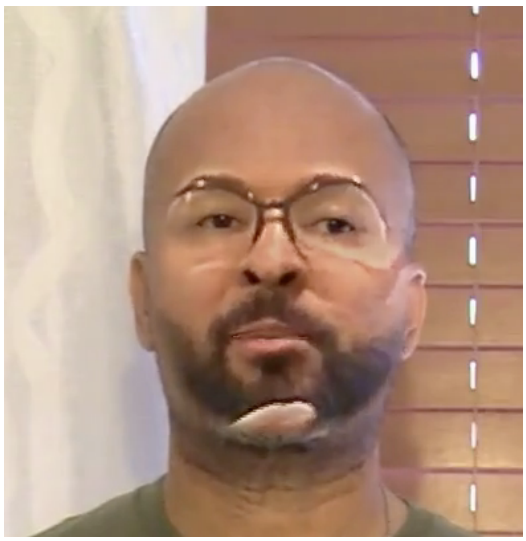


Figure 5.18: Example of a sample with irregular facial traits. We can see the chin area being clearly irregular in terms of color and shape (this sample could also be labeled as color inconsistency). Sample taken from the DFDC dataset.

arms which are missing in most of the videos where the face swapped has glasses and the original does not. An example is shown in Figure 5.19.

Figure 5.19: Example of a sample with glasses inconsistency. Here we can see how the glasses arms are clearly missing. Sample taken from the DFDC dataset.

5.2.3 Flickering detection

In our work we decide to detect flickering in videos. We make this choice since this inconsistency can be, to a certain extent, synthetically replicated. Using the synthetic dataset as ground truth, we train our video-level models.

The employed pipeline is the same as the deepfake detection one, the only differences are:

- training data is taken from our synthetic dataset (which will be further explained in Chapter 6);
- the final label is one between FLICKERING/NO_FLICKERING.

Frame selection We select 30 consecutive frames (randomly sampled from the entire sequence of frames) for each sample, considering that the flickering effect needs to be spotted across multiple consecutive frames. We use 30 and not 15 frames because we want to have a larger time span covered by a single sequence, in order to better identify the effect.

Detection model employed We opt for the same conv-LSTM architecture with Inception-V3 used for deepfake detection, since it is the most successful approach.

Final preprocessing As for deepfake detection, the only preprocessing needed is the rescale of RGB values in the [-1,+1] interval.

Chapter 6

Datasets and Web Interface

In this chapter we describe the off-the-shelf employed dataset, our flickering dataset generation and the Web Interface.

6.1 Off-the-shelf employed dataset

The dataset used for this work is the DeepFake Detection Challenge Dataset (DFDC) [49], a dataset that was used for the homonymous challenge on Kaggle. We choose it since for our work we need videos and the DFDC is the only one that had enough video samples to work with. Currently this is the biggest deepfake dataset, composed of more than 100,000 videos about 10 seconds long; every video has a frame-rate of 30 fps, for a total of more than 10 millions frames in the dataset. For our task we only use the training part of the dataset composed of 119,154 clips (RGB frames), most of them with a resolution of 1920x1080. The clips are divided into 50 folders which are organized in actors such that the same actor only ends up in one of the folders. In each of the 50 directories, there are multiple pristine videos with their respective deepfakes generated.

We split the dataset into training, validation and test set: the split is respectively 60%, 20% and 20%. We propose these percentages because the dataset is big enough to have sufficient data for training even with only 60%; at the same time, it allows us to have more test data, meaning that we can obtain a more accurate estimate of the generalization performance of our models. We take folder from 1 to 30 as training set, from 31 to 40 as

validation set and from 41 to 50 as test set. We choose this split in order to have videos with the same actor in the same set, thus avoiding overfitting. Indeed, at first, we tried splitting every single folder in train/validation/test and that resulted in a strong overfitting on the test data, since we were using very similar data for training, validation and testing.

The deepfakes in this dataset are generated using some of the state of the art methods: DFAE, MM/NN face swap [51], Neural Talking Head (NTH) [52], FaceSwap-GAN, StyleGAN.

6.2 Flickering dataset generation

For flickering detection, we need a ground truth in order to train our supervised models. Therefore, we employ the DFDC dataset (split as previously described) to build a training and validation set. The main idea of the synthetic dataset is to replicate flickering by forcing abrupt changes in consecutive frames of a video. The problem remains a binary classification, but there is no REAL/FAKE classification; instead we have a FLICKERING/NO_FLICKERING distinction to detect.

Before talking about the dataset generation we need to define what is an 'original' video. Given a video labeled as fake, we call 'original' the video from where the deepfake is generated. The dataset already provides this information, making things easier for us.

To generate samples, different approaches are considered whether we are dealing with a fake video or a pristine video. We process all the training and validation samples one at a time and we proceed like this:

- if it is a pristine video, we discard the current sample;
- if it is a fake video, then at first we check if the 'original' video has already been processed;
  - if it has been processed, then a choice has to be made and with a Bernoulli probability of 0.5 the sample is chosen to be FLICKERING or NO_FLICKERING. If it should be a no_flickering sample, then the video is discarded, since it would be equal to another no_flickering one with the same video as 'original';

– if it has not been processed, with a Bernoulli probability of 0.5 the sample is chosen to be FLICKERING or NO_FLICKERING. If it is a no_flickering sample, its 'original' is marked as processed.

Flickering samples A flickering sample is generated considering a fake video and its 'original' one. The same sequence of frames in both videos is processed and cropped. Then, the flickering series is generated. Sequence starts from t_0, the initial frame, to t_f, the final frame, where $t_f - t_0$ is equal to 30. For every i from t_0 to t_f, with a Bernoulli probability of 0.5, a sample is selected either from the fake or the 'original' sequence of frames. Following this selection process, abrupt changes in facial traits are forced between consecutive frames. We can see, in Figure 6.1, examples of synthetic flickering sequences.

No Flickering samples These samples are 'original' videos from the DFDC dataset.

With this approach the training dataset contains 25940 flickering samples and 4940 no_flickering samples. As we can see, the dataset is highly unbalanced towards the flickering class.

6.2.1 Stable videos

The reason why we define and use this category of videos is to avoid having flickering samples with uncontrolled abrupt changes, which could happen when the face detector recognizes something other than the face in the video or there are multiple people and therefore the face detector does not always detect the same person throughout a sequence. In order to define what a stable video is, we need to specify some useful values.

We denominate the average detection width of the video as the average of the width of the detections of the considered frames:

$$avg\_width = \frac{\sum_{i=1}^{T}(x_{max}[i] - x_{min}[i])}{T} \tag{6.1}$$

where T is the number of frames considered, $x_{min}[i]$ is the smallest horizontal position of all detection pixels of the i-th frame considered and $x_{max}[i]$ the

highest horizontal position.
In the same way, but for the height, we defined the average detection height:

$$avg\_height = \frac{\sum_{i=1}^{T}(y_{max}[i] - y_{min}[i])}{T} \tag{6.2}$$

where y_{min} and y_{max} refer to ordinates.

We define the average width point and the average height point of a video, with T frames considered, as follows:

$$avg\_width\_point = \frac{\sum_{i=1}^{T}\sum_{j=x_{min}[i]}^{x_{max}[i]} j}{\sum_{i=1}^{T}\sum_{j=x_{min}[i]}^{x_{max}[i]} 1} \tag{6.3}$$

$$avg\_height\_point = \frac{\sum_{i=1}^{T}\sum_{j=y_{min}[i]}^{y_{max}[i]} j}{\sum_{i=1}^{T}\sum_{j=y_{min}[i]}^{y_{max}[i]} 1} \tag{6.4}$$

here the denominator corresponds to the count of the number of total pixels considered across the entire video.

We define other two quantities: the width standard deviation and the height standard deviation. The width standard deviation is defined as the std_dev of of the x coordinates of the detection of all the frames, with respect to the average width point. The same holds for the height standard deviation, but considering y coordinates.

Finally, we define two ratios, one for the height and one for the width, as follows:

$$width\_ratio = \frac{std\_dev\_width}{avg\_width} \tag{6.5}$$

$$height\_ratio = \frac{std\_dev\_height}{avg\_height} \tag{6.6}$$

We define a video as **stable** if both ratios are smaller than 0.5. Using this definition of stable video, we discard videos that are not stable for the training, and subsequently, for the validation and test. In particular, for the synthetic flickering samples we discard the sample if either the fake video or its 'original' one are not stable. The threshold value 0.5 has been tested to be a good trade-off between keeping enough videos and their stability.

For stable samples instead, the region cropped is fixed for all its frames and it is a 224×224 crop adapted simmetrically from the avg_width_point

and the avg_height_point. In Figure 6.1 we can see two examples of stable synthetic flickering samples.

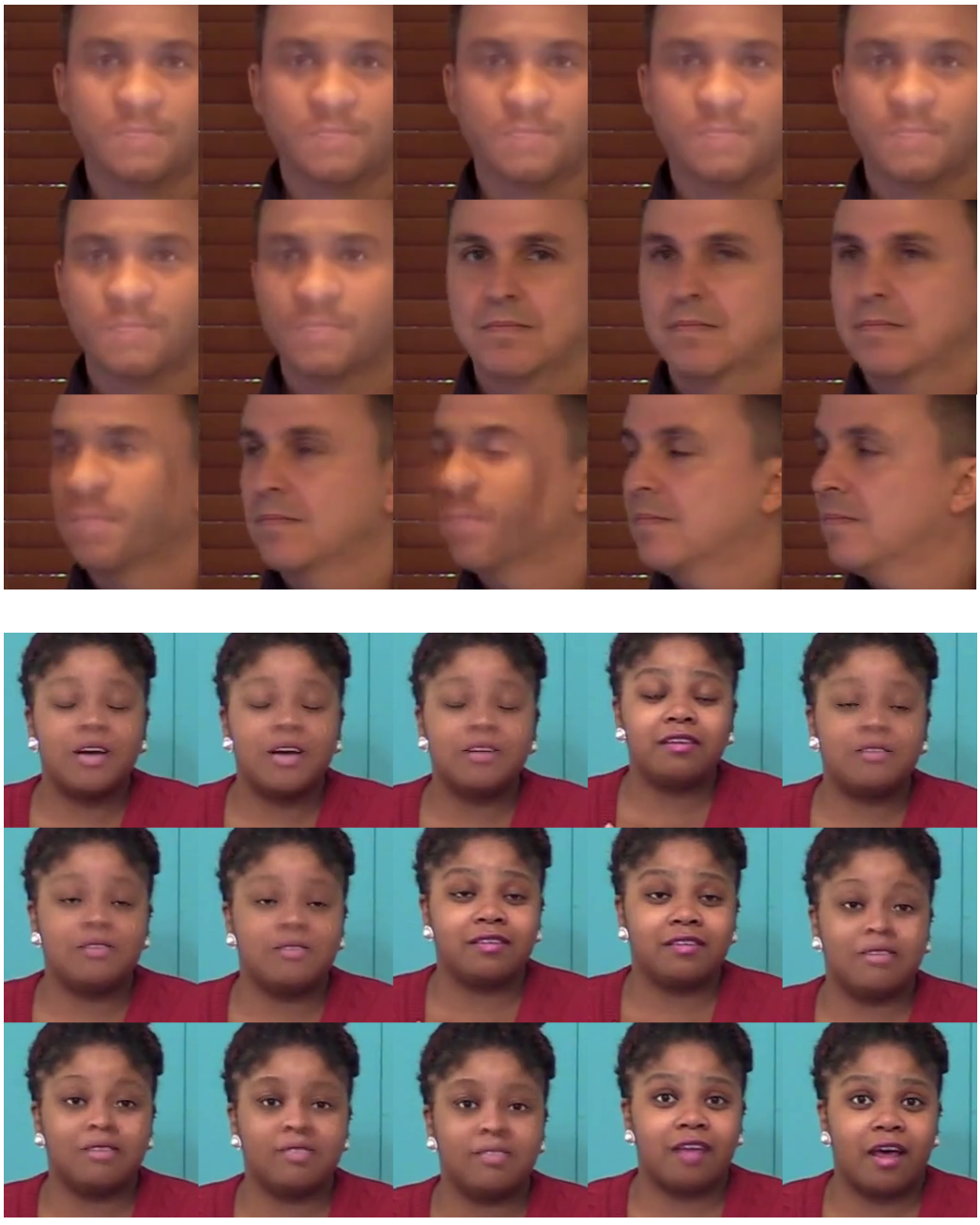


Figure 6.1: Example of synthetic flickering stable videos.

Flickering test set In order to build a small ground truth for testing, we randomly sample 200 stable videos from the test set directories of the DFDC dataset(split as described before). Ninety percent of videos we randomly select are fake and ten percent pristine. We use this split since the flickering effect we want to detect is only present in fake videos. We manually label these samples ourselves in order to test our models.

6.3 Deepfake Artifacts Dataset generation via Web Interface

A problem we face in our work is the lack of labeled datasets on inconsistencies.

A first attempt to address this problem is made with our synthetic dataset generation, where we artificially force flickering in videos. Labels regarding artifacts have been described in Chapter 5, where we have defined every single category providing examples of them.

A second attempt is made by showing a Web Interface we develop to users, where they are required to see clips and to try and label them the best they could. Finally, the annotation collected via the Web Interface are employed to assess performance of our flickering detection models, as we will explain in Chapter 7.

6.3.1 Web Interface

The Web Interface we use to interact with users is coded using HTML5, CSS, Javascript and JQuery. We extensively make use of the Bootstrap library (version 4.0), a very useful and high-level library that helps with the management of forms and elements positioning in websites.

The interface is composed of two pages, one for the introduction and the second one for video annotations. Furthermore, the interface has 2 versions, Italian and English, to allow usage as much as possible.

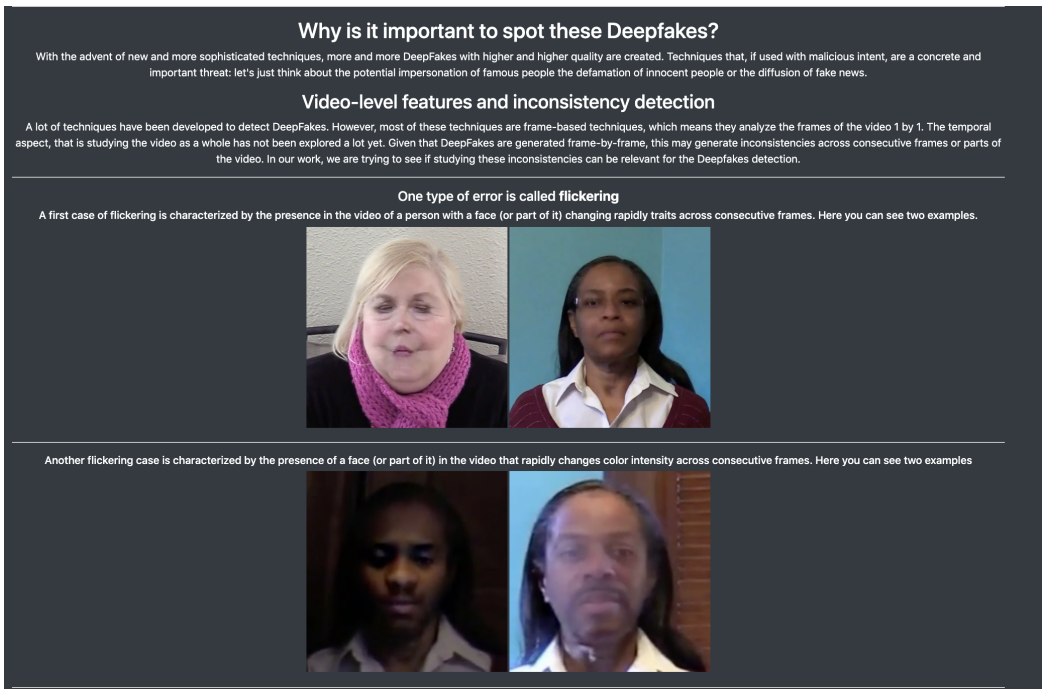


Figure 6.2: Screen from the introduction page. Here we can notice the brief text introduction for the user and some examples shown of the flickering effect. The pictures are autoplaying videos.

Introductory page

The first page is a welcome page. It briefly introduces the users to the deepfake concept and to the threats related to these videos. It then clarifies to the user what is the purpose of our work (artifacts detection) and proceeds in showing users different examples of the labels we want to collect. A screen of a portion of the page is shown in Figure 6.2.

Annotation page

As we can see from Figure 6.3, the page shows a short clip on the left, which is autoplaying and muted by default. Users can control the video with classic controls.

Since sometimes the faces are quite small and the inconsistencies may not be easily noticeable, we add a zoom functionality (Figure 6.4) that shows the user the video zoomed on the face region. This functionality, as several users reported, is very useful in labeling some videos.

On the right side of the page there is a single, multiple choice question.

The question asks the user to report what types of artifacts (if any) are present in the video. The answers are the types of artifacts presented in Chapter 5 with an additional "Other" answer. This answer is a custom one, where users can type whatever error/inconsistency they notice that is not captured by the previous categories. When the user finishes answering the question, he can confirm his selections and go to the next video.

Examples of possible answers are shown in Figure 6.5 and Figure 6.6.

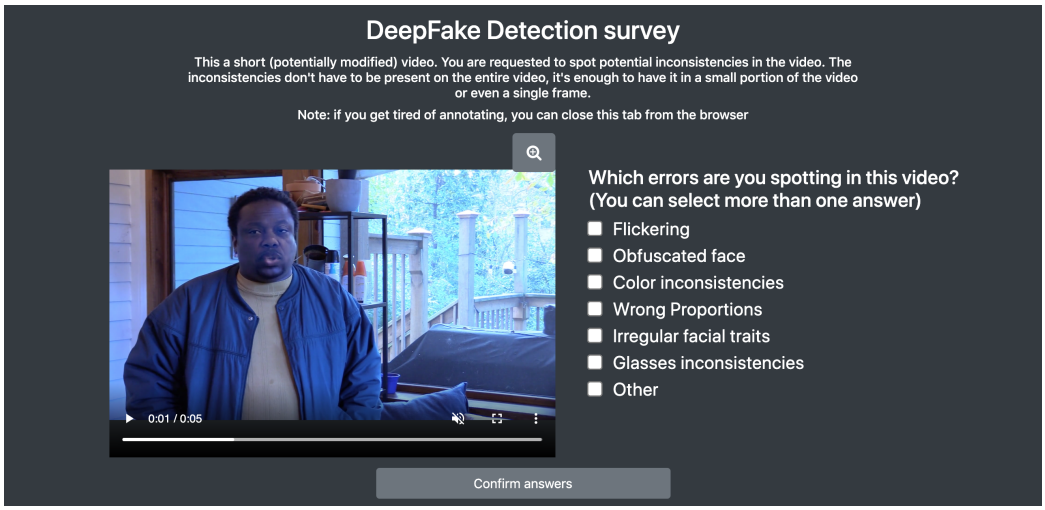


Figure 6.3: Screen of the annotation page.

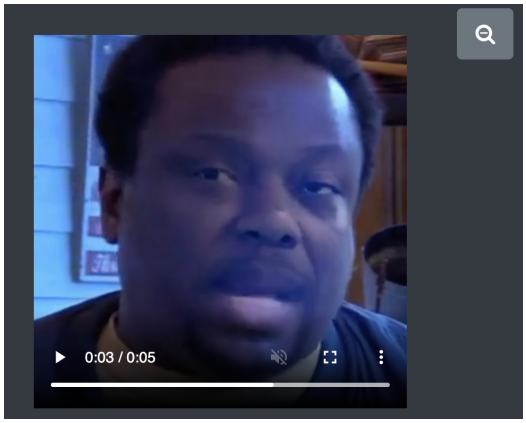


Figure 6.4: Example of the zooming functionality.

Dataset used

To collect annotations we use the DFDC test set, according to the directory split described before. From every folder, we take 20 fake videos and 2 pristine

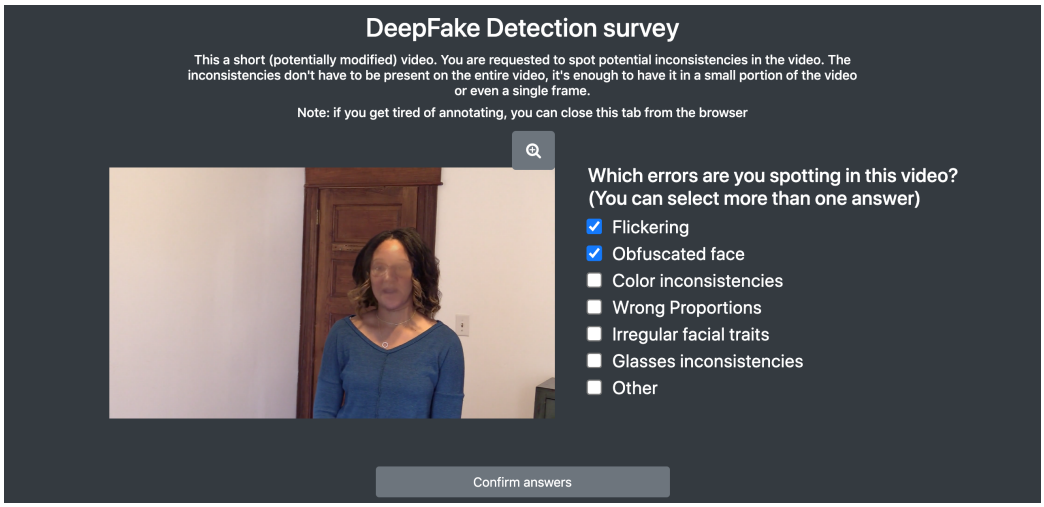


Figure 6.5: Example of a possible answer on a flickering video with obfuscated face. Answer taken from the collected annotations.

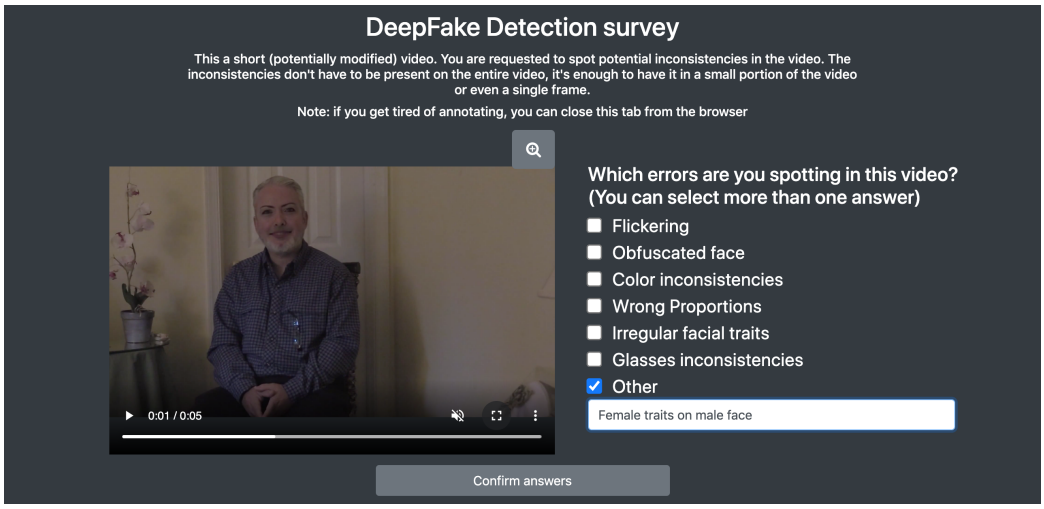


Figure 6.6: Example of a possible custom answer. Answer taken from the collected annotations.

videos. Every video is then split into two 5-seconds chunks, for a total of 400 fake chunks and 40 pristine chunks in the dataset. This split is done to show users shorter clips in order to make answering easier and less bothering. Every considered video is a stable video, following the definition explained above, given that we want to remove wrong face detections in zoomed videos.

Chapter 7

Experiments

In this chapter we explain the performed experiments on deepfake detection and on flickering detection. Finally, we report some statistics on the annotations collected via the Web Interface we have developed.

7.1 Deepfake detection

In this section we evaluate and compare our proposed approaches on deepfake detection.

7.1.1 Training setup

In this subsection we describe the framework employed to train the models shown in Chapter 5. Considering the vast amount of offered libraries for deep learning and its flexibility, we use Python as a programming language for our work (version 3.6.0).

LSTM training

For the convolutional LSTM models we opt for the use of the Tensorflow framework (version 2.3.0). In particular, we decide to use Keras to build and train our models, due to its understandability and easiness of use. To prepare the dataset as input to the models we use some Tensorflow libraries to adapt the models to our problem. In particular, we use them to create our own dataset generator. The most successful trainings (other hyperparameters

were tried but with less success) use a batch size of 4, a learning rate of 10^{-4} and Adam as the optimizer. Since it is a classification problem and the softmax is used to classify the data, we choose the sparse categorical cross-entropy as the loss function. Given the large quantity of data available and the fact that the dataset is highly unbalanced, 15 epochs are enough to make most of the models overfit. The validation step is performed at the end of each epoch. No cross-validation is performed, given the size of the dataset.

Transformers training

For Transformers architectures (Timesformer and Video Transformer Network) we use the Pytorch (version 1.9.0) framework. We decide to change the framework given the high number of parameters of these architectures and the huge training time these architectures require. Indeed, Pytorch offers a more optimized training when dealing with huge quantity of data and big models, saving a considerable amount of time.

The most successful training use a batch size of 2 (more was not possible due to memory constraints), a learning rate of 10^{-4} and Adam as the optimizer. Here we opt for a sigmoid activation function and, subsequently, for the binary cross-entropy with logits loss as a loss function.

Even with these models, 15 epochs are enough to make them overfit. The validation step is performed at the end of each epoch.

7.1.2 Evaluation metrics

The same preprocessing applied to the training set (as described in Chapter 5) is performed on the test samples before evaluation. The sequence of cropped frames is fed as input to a trained model (with the lowest validation loss checkpoint) which predicts class scores.

The metrics we select to evaluate performance are: accuracy, balanced accuracy and area under the curve. We compute these metrics on the test set, following the split defined in Chapter 6.

Accuracy

The accuracy of a model is the percentage of samples correctly classified with respect to the total number of samples:

$$Accuracy = \frac{TP + TN}{N} \tag{7.1}$$

where TP are the True positives (fake samples correctly classified), TN are the true negatives (pristine samples correctly classified) and N is the total number of test samples.

Balanced Accuracy

The balanced accuracy (or average recall) is a metric that is really important in our study since our training dataset is highly unbalanced. This quantity is defined as the average between sensitivity (or True Positive Rate or Recall) and specificity (or negative Recall). These values are defined below:

$$TruePositiveRate = TPR = \frac{TP}{TP + FN} \tag{7.2}$$

$$Specificity = Negative\_Recall = \frac{TN}{FP + TN} \tag{7.3}$$

$$Balanced\_Accuracy = \frac{TPR + Specificity}{2} \tag{7.4}$$

where FN are the False Negatives (fake samples classified as REAL) and FP are the False Positives (pristine samples classified as FAKE).

Area Under the Curve(AUC)

The area under the curve is the area below the ROC curve. The ROC curve is defined by True Positive Rate and False Positive Rate computed at different classification thresholds. Here we define False Positive Rate, while the True Positive Rate has been defined above.

$$FalsePositiveRate = FPR = \frac{FP}{FP + TN} \tag{7.5}$$

7.1.3 LSTM performance comparison

In the following section we present the results related to the Inception-V3+LSTM model (described in Chapter 5) trained with different approaches.

The effect of rescaling

In this section, two different preprocessings, differing in the rescaling part, are compared:

1. 224×224 crops, centered around the face, no rescale and augmentation performed;
2. face detection crops with a small margin, rescaled to 224×224.

In Table 7.1 we can notice the difference in the performance.

Preprocessing	Accuracy	Balanced accuracy	AUC
1	**0.911**	**0.864**	**0.952**
2	0.88	0.82	0.92

Table 7.1: Performance metrics of the two approaches using the same architecture (conv-LSTM-IV3) but with different rescaling.

As an additional experiment and following the ICPR work [29], we tried as cropping approach to only consider for each frame a squared region containing only the face detection, with no added margin, and rescaling it to 224×224, but the approach did not help improving classification accuracy with LSTMs.

Crop size choice

In this section, two different preprocessings, differing in the crop size, are compared:

1. 224×224 crops, centered around the face, no rescale and no augmentation performed;
2. 380×380 crops, centered around the face, no rescale and no augmentation performed.

In Table 7.2 we can notice the difference in the performance.

Frame selection approaches comparison

In this section we show the comparison between two approaches. They both use the most successful preprocessing: 224×224 crops centered around the face, no rescale and augmentation performed. The difference is the number of frames per video selected:

Cropping	Accuracy	Balanced accuracy	AUC
224×224	**0.911**	**0.864**	**0.952**
380×380	0.90	0.82	0.93

Table 7.2: Performance metrics of the two approache using the same architecture (conv-LSTM-IV3) but with different crop sizes.

1. 15 evenly spaced frames across the video;

2. 30 evenly spaced frames across the video.

In Table 7.3 we can notice the difference in the performance.

Frames used	Accuracy	Balanced accuracy	AUC
15	**0.911**	**0.864**	**0.952**
30	0.88	0.77	0.90

Table 7.3: Performance metrics of the two approaches using the same architecture (conv-LSTM-IV3) but with different selected number of frames.

As an additional experiment, we tried using 45 or more frames but the performances dropped considerably, therefore results are not shown.

Another frame selection approach we thought could help was trying and considering 30 consecutive frames per video. In theory, if there is an inconsistency, like flickering, it should contain it. The problems with this approach are mainly 2:

- the probability of catching an inconsistency while cropping in that region is quite low, since inconsistencies are random in the video and we do not know a priori if there is one in the video;

- in 30 consecutive frames (corresponding to more or less 1 second) the face does not move too much, therefore we are not catching enough information to maybe catch pixel differences in the frames.

Trainings indeed showed no improvement in classification accuracy and therefore we discarded this approach.

The effect of balancing the dataset

In this section we show the comparison between two approaches. They both use the most successful preprocessing: 224×224 crops centered around the

face, no rescale and augmentation performed. The difference is the training set used:

1. full training set as described above, so DFDC dataset folders from 1 to 30;

2. DFDC training set, with downsampling of the FAKE samples (taking more or less 30% of the FAKE samples), in order to have a more balanced dataset.

In Table 7.4 we can notice the difference in the performance.

Dataset	Accuracy	Balanced Accuracy	AUC
Full	**0.911**	0.864	0.952
Balanced	0.89	**0.884**	**0.956**

Table 7.4: Performance metrics of the two approaches using the same architecture (conv-LSTM-IV3) but with different training set.

As an additional experiment, we tried to oversample the REAL class by taking more sequences from the same video, but since the sequences are very similar in several cases, this resulted in overfitting in the training set and therefore results were not worth showing.

Furthermore, as we will show later in this chapter, we can notice how downsampling helps with the recognition of pristine samples by making the training dataset more balanced. At the same time though, having less training samples makes the classifier less accurate.

7.1.4 Transformer performance comparison

In the following subsection we discuss about the Transformer approaches we have tested.

Video Transformer Network

Since the 224×224 crop is the best performing approach on convolutional LSTMs, we decide to employ this preprocessing to Transformers architectures too.

We first propose the VTN with Resnet50 as feature extractor (as in the original code). Then, as a comparison, we change the backbone, selecting a

small but effective network (since we had memory limits with other architectures): EfficientNetB0. Metrics are shown in Table 7.5.

As an additional experiment, we tried changing VTN hyperparameters, like the number of heads, obtaining comparable performances with the one shown in Table 7.5.

Backbone	Accuracy	Balanced Accuracy	AUC
Resnet50	**0.905**	0.846	0.930
EfficientNetB0	0.901	**0.897**	**0.965**

Table 7.5: Performance metrics of the two best versions of the Video Transformer Network (VTN)(16 heads), with different backbones.

7.1.5 Deepfake detection architectures performance study

Here we present and compare the test performances of the architectures shown in Chapter 5. We decide to show these approaches since they are the best performing ones for all the proposed models. The approaches examined are:

1. convolutional LSTM (IV3 pretrained of Imagenet), 224×224 crops, no rescale no augmentation, end-to-end training, 15 frames;

2. convolutional LSTM (IV3 pretrained on Imagenet), 224×224 crops, no rescale no augmentation, smaller balanced dataset, end-to-end training, 15 frames;

3. convolutional LSTM (EB4 pretrained of DFDC dataset), 224×224 crops, no rescale, augmentation, EfficientNet frozen during training, 15 frames;

4. Video Transformer Network (EB0 pretrained on Imagenet), 224×224 crops, no rescale no augmentation, end-to-end training, 15 frames;

5. Timesformer (pretrained on Kinetics-600), 224×224 crops, 8 frames, no rescale no augmentation. end-to-end training.

The metrics computed for every approach are shown in Table 7.6.

Approach	Accuracy	Balanced accuracy	AUC
conv-LSTM-IV3	**0.911**	0.864	0.952
conv-LSTM-IV3-balanced	0.897	0.884	0.956
conv-LSTM-EB4	0.905	0.845	0.93
VTN-EB0	0.901	**0.897**	**0.965**
Timesformer	0.83	0.65	0.85

Table 7.6: Performance metrics of the proposed models.

Predictions study

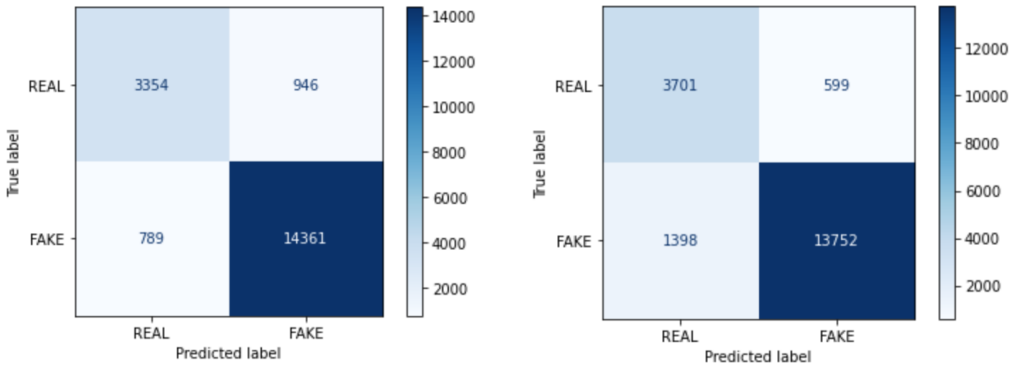


Figure 7.1: Confusion matrices of conv-LSTM-IV3 approaches. On the left we can see the approach with complete dataset training, while on the right the approach with balanced dataset training.

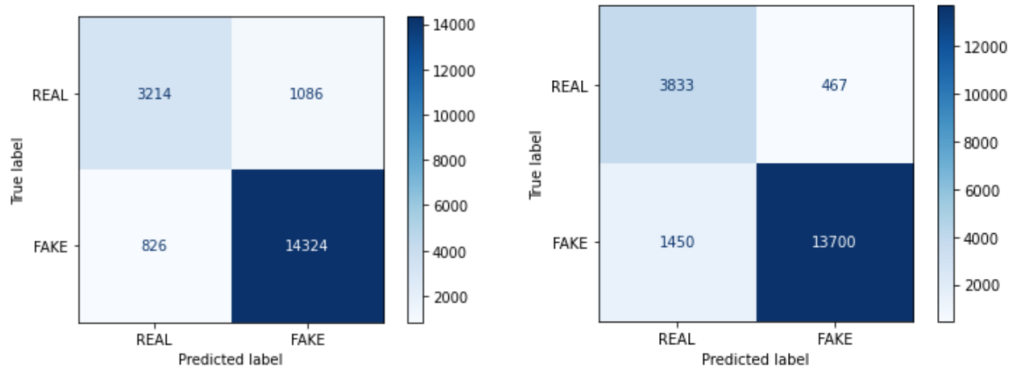


Figure 7.2: Confusion matrices of the VTN-EB0 approach, on the right, and of the conv-LSTM-EB4 approach, on the left.

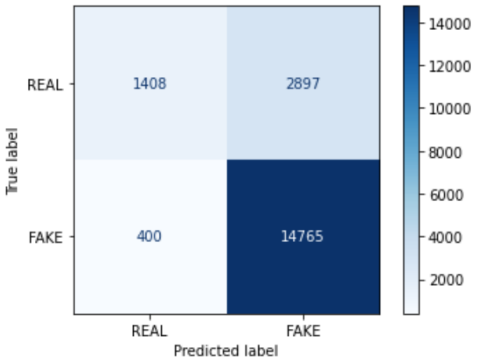


Figure 7.3: Confusion matrix of the Timesformer approach.

From Table 7.6 we can notice how convolutional LSTM (with Inception-V3 as backbone) and Video Transformer Network (with EfficientB0 as backbone) outperform any other proposed model/approach. In Figure 7.6 and Figure 7.5 we show the histograms of these models prediction for all test samples.

As we can see from the confusion matrices and the histograms of the predictions, every model performs very good on fake samples, classifying them with a pretty high score (most of the FAKE predictions are values close to 1). This has to be expected, given the imbalance of the training dataset. The VTN Transformer performs the best among our models on pristine samples, considering the balanced accuracy and the AUC, even if it is trained on the full dataset. We can appreciate this performance by having a look at the confusion matrix (Figure 7.2) and at the histogram (Figure 7.6). The convolutional LSTM models perform better on fake videos but worse on reak ones, even with the balanced dataset.

We can see in Figure 7.1 and in Figure 7.5, how balancing the dataset helps in better recognizing pristine samples for the conv-LSTM architecture.

Timesformer model performs very well on the fake samples while the accuracy on pristine samples is very low (Figure 7.3).

In Figure 7.4 we can observe the ROC curves of the models presented.

Calibration plots

It is useful, in order to evaluate our models, to evaluate how well models are calibrated.

Formally, a model is perfectly calibrated if, for any probability value p,

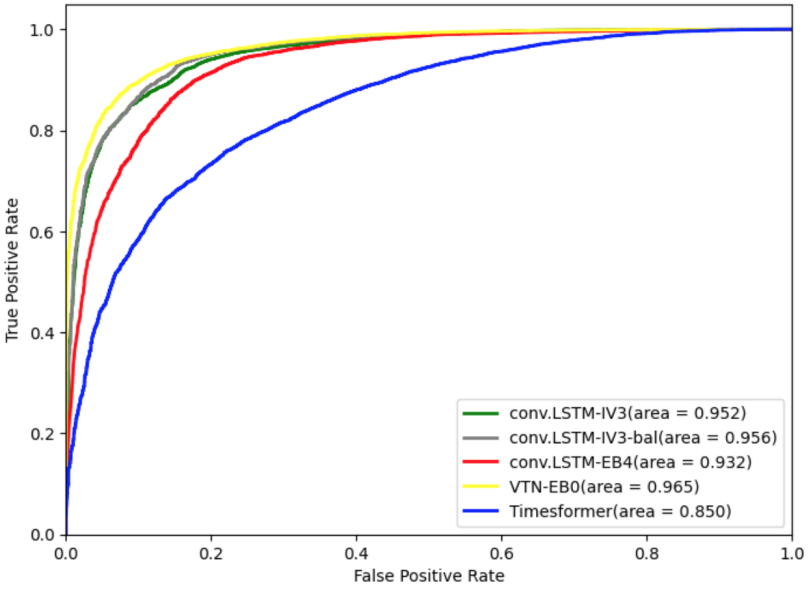
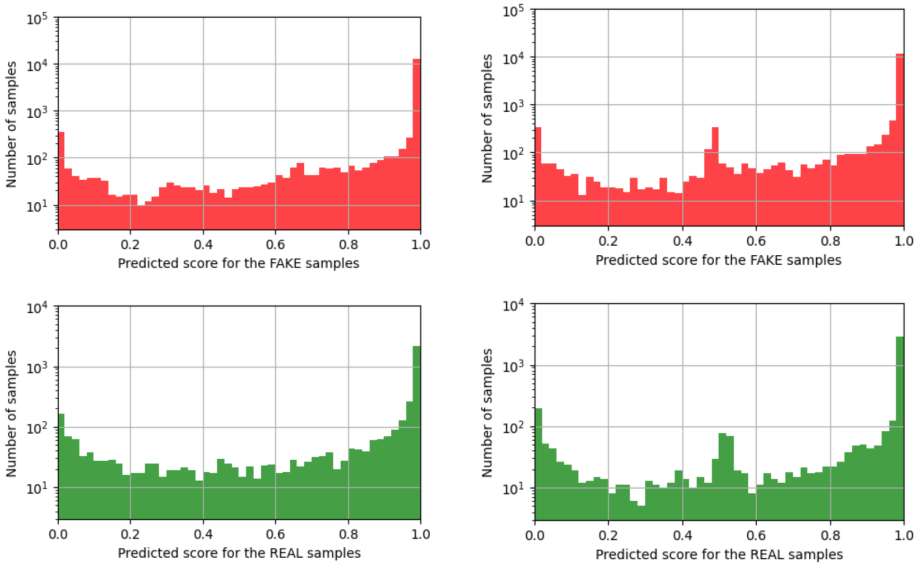


Figure 7.4: ROC curves of the 5 best deepfake detection approaches.

a prediction of a class with classification score p is correct $100 * p\%$ of the times [53]. As an example, if we consider the samples classified with a score of 0.30 by a perfectly calibrated classifier, then 30% of those samples belongs to class 1.

The calibration curve shows the calibration of a classifier. Given a series of bins (intervals), for every bin (prediction score) the percentage of samples of class 1 is computed. The curve obtained by connecting the dots is the calibration plot and can be compared to the perfect calibration plot which represents a perfect linear relationship. If a plot is below the perfect calibration curve it means that the model is under confident on the classification score, so there should be more samples classified as class 1; viceversa, when the plot is above the perfect calibration curve.

Here we show the calibration plots of the best 3 models: convolutional-LSTM-IV3, convolutional-LSTM-IV3-balanced, VTN-EB0. As we can see from Figure 7.7, the overall better calibrated model is the conv-LSTM without balanced dataset training. It is much better calibrated than the other models in the scores smaller than 0.5. This is maybe due to the fact that this model is very good in classifying fake samples, also with respect to the other models (as we can also see from the confusion matrices in Figure 7.1 and Figure 7.2). On the other hand, the conv-LSTM approach is worse on the



(a) Histogram of the predictions for convolutional LSTM with Inception-V3, full dataset, 224 × 224 crops.

(b) Histogram of the predictions for convolutional LSTM with Inception-V3, balanced dataset, 224 × 224 crops.

Figure 7.5: Histograms of conv-LTSM architectures.

higher probabilities. This is somewhat expected, since the model performs much worse on the pristine samples, classifying several samples as fake with high confidence. Given these considerations and the fact that the test set is imbalanced towards the fake class, it seems reasonable that the conv-LSTM curve is the best.

7.1.6 Frame-level models comparison

Unfortunately, video-level models are not yet competitive with frame-based detectors, at least in the DFDC dataset. This conclusion was also reached by Selim Sef, winner of the Deepfake Detection Challenge, obtaining good results on solely EfficientNetB4 and EfficientNetB7. Some metrics performances obtained by the ICPR work [29] are written in Table 7.7. The crops described

Frame-based classifier	Balanced accuracy	AUC
EB4, 224 × 224 crops	0.888	0.956
EB4, 380 × 380 crops	0.929	0.978
EB7, 224 × 224 crops	0.926	0.976

Table 7.7: Frame-based classifier results of the ICPR work [29].

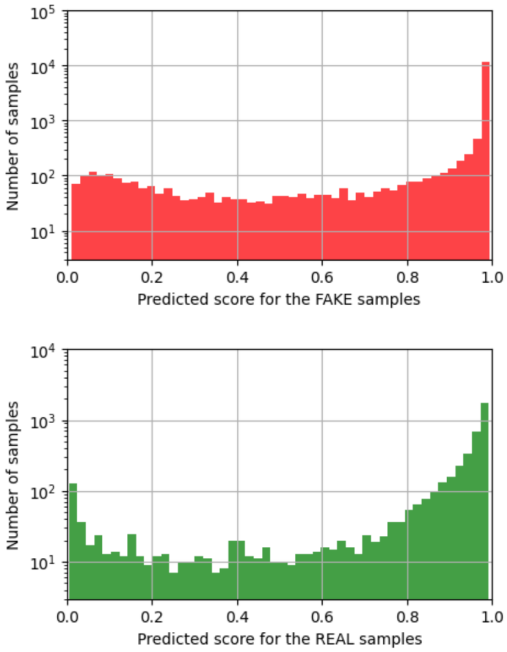


Figure 7.6: Histogram of the predictions for the Video Transformer Network with EB0, full dataset, 224 × 224 crops.

in the table are not the regions extracted from the frames but they are squares containing the face detection with no additional margin then rescaled to the desired dimension. The obtained results are at video level, where the video label is obtained by majority voting of its frame labels. This approach is

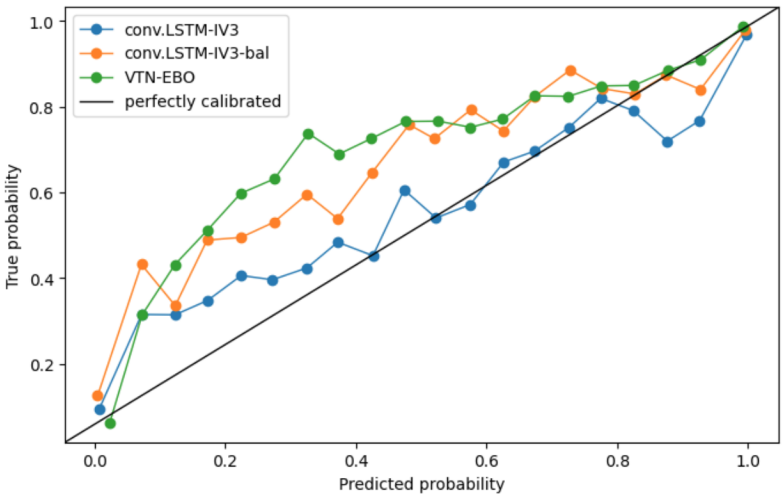


Figure 7.7: Calibration plots of the best 3 models. On the x axis, the prediction score and on the y axis the true probability. The number of values considered per each plot is 20.

overall better with respect to aggregating frame features via LSTM or any other video-level model we propose, but this discrepancy in performance should not mislead.

In the frame-based approaches only a single frame is considered at a time, allowing use of traditional CNN architectures to extract features and classify. Furthermore, frame-based models are trained on potentially millions of different samples, since multiple frames from the same videos can be used as different samples. Let us suppose we take 10 frames from each video in the DFDC, we have almost 1 million samples for an EfficientNetB4 architecture.

Video-based approaches are more challenging. The de-facto standards (also what works best for us) are architectures based on feature extraction and recurrency. The aim of these architectures is to try and spot very little pixel discrepancies across frames. This task is not trivial, since what is fed to the LSTM layer are not the frames themselves but their high-level features. To tackle this problem, as an additional experiment, we tried to directly fed raw images to the LSTM layer but we obtained poor results.

Another potential problem for these architectures is the considerably lower size of employed datasets. For video-level classification, only a single sample for each video must be considered in order to avoid overfitting. If we consider the DFDC dataset (which is the largest available deepfake video dataset) we only have almost 120,000 samples, which is not nearly enough to what is needed to solve the complex problem we are facing and to make models learn properly.

7.2 Flickering detection

In this section we report the results of our best models on the synthetic flickering detection.

7.2.1 Test samples labeling

In order to label a test video sample, we process it entirely. Since the models are trained on 30-frames-long sequences, the entire video, which on average has 300 frames, is processed in 30-frames chunks with a sliding window between two chunks of 25 frames. We opt for overlapping chunks since we want

to be able to capture flickering in every possible sequence.

A sample is labeled as FLICKERING if at least one chunk is labels as FLICKERING. The rationale behind this choice is that flickering may not be constant across the video and, at the same time, it may just be present in a small portion of the entire sequence.

7.2.2 Evaluation metrics

Accuracy In the flickering detection task we consider accuracy and not the balanced accuracy because the test set is balanced and the two metrics do not differ by a significant margin. In addition, we compute two other metrics (precision and negative predictive value) in order to check the performance of our models in the positive and negative class predictions.

Precision Precision is the percentage of True Positives among all samples labeled as positives:

$$Precision = \frac{TP}{TP + FP} \tag{7.6}$$

Negative Predictive Value Negative Predictive Value (NPV) is the percentage of True Negatives among all samples labeled as negatives:

$$NPV = \frac{TN}{TN + FN} \tag{7.7}$$

7.2.3 Performance evaluation for synthetic training sets

The model tested is the convolutional LSTM with Inception-V3 as a backbone, which is described in Chapter 5. We train the model on the synthetic dataset described in Chapter 6. The obtained performance is 76% accuracy over the flickering test set described in Chapter 6. The model shows good performance on the videos with consistent flickering but presents a large number of false positives, mainly due to the face moving too much in the video or the video being too dark. To address this fact, we tried decreasing the flickering frequency (putting less fake frames in the flickering samples) in the training but it did not help with the performance. To compensate for the darkness of some videos we tried adding (using sk-image python library) more contrast in the images, but that overall decreased the performance on

the test set, obtaining 74%. To try and reduce the number of false positives we tried downsampling the flickering samples and oversampling the no_flickering samples but the performance did not improve. Several other approaches were tried but did not achieve any good results. Below, we name some of them:

- considering as input to the conv-LSTM-IV3 architecture a 30-frames-long sequence where every frame is the pixel-wise difference between a frame and its successor;
- considering as input the mean pixel-wise difference among consecutive frames, getting rid of the feature extractor, feeding a 30-floats-long vector, which is none other than a simple time series, directly to the LSTM layer;
- considering other machine learning models (Random Forest, XGBoost, SVM).

Results are shown in Table 7.8.

Dataset	Accuracy	Precision	NPV
stable	0.76	0.68	0.84
stable, contrast enhancement	0.74	0.68	0.80

Table 7.8: Flickering detection results.

7.3 Performance study using interface data

In this section we show results and statistics of the collected annotation using the Web Interface.

7.3.1 Labels collected

We have collected 925 annotations from users, for a total of 347 different samples labeled. There is an average of 3.5 annotations per annotated sample.

As we can see in Table 7.9, all the artifacts are detected in a not negligible part of the dataset, while the "Other" category is present in only a few

annotations (51), meaning that in the vast majority of the videos at least one of the other inconsistencies is detected.

	Present	Not Present
Flickering	360	565
Obfuscated Face	201	724
Irregular Traits	247	678
Colors inconsistency	244	681
Irregular Proportions	145	780
Glasses Inconsistency	170	755
Other	51	874

Table 7.9: Collected annotations.

7.3.2 Extracted data

As a first study, we show the number of labeled samples per category. Results are shown in Table 7.10. As we can see, the sum of samples in every row is greater than 347. This is due to the fact that some samples have received both the positive and the negative answer by different users in some categories.

Another useful index is the number of the samples with coherent/incoherent labels per category, shown in Table 7.11. This table shows how it is very easy for the users to agree on the negative answer while it is more tricky and challenging in the positive answers.

Finally we show the percentages per category of coherent labels, in Table 7.12.

	Present	Not Present
Flickering	196	274
Obfuscated Face	114	307
Irregular Traits	159	310
Colors inconsistency	166	309
Irregular Proportions	117	330
Glasses Inconsistency	91	314
Other	44	345

Table 7.10: Number of samples labeled as one of the classes. Data taken from the collected annotations.

	Coherent Positive	Coherent Negative	Incoherent
Flickering	73	151	123
Obfuscated Face	40	233	74
Irregular Traits	37	188	122
Colors inconsistency	38	181	128
Irregular Proportions	17	230	100
Glasses Inconsistency	33	258	58
Other	2	303	42

Table 7.11: Number of samples with coherent/incoherent labels per each class. Data taken from the collected annotations.

	Coherent Positive(%)	Coherent Negative(%)	Incoherent(%)
Flickering	21	43.5	35.5
Obfuscated Face	11.5	67	21.5
Irregular Traits	10.6	54.2	35.2
Colors inconsistency	11	52.2	36.8
Irregular Proportions	5	66.2	28.8
Glasses Inconsistency	9.5	74.4	16.1
Other	0.6	87.3	12.1

Table 7.12: Number of samples with coherent/incoherent labels (in percentage) per each class. Data taken from the collected annotations.

7.3.3 Statistics

In this section, we describe some statistics related to the data collected via the Web Interface. Finally, we compare these values with the accuracy of our artifacts detection models.

Inter-rater agreement

The inter-rater agreement is a statistics used to check how much several raters agree on the answers to the same questions/samples. It is the percentage of the pairs of responses to the same items that are coherent. In our case we

consider a general setting with a (potentially different) number of answers per sample. In order to compute the inter-rater agreement, we consider these following values per sample:

- $n_{pos-pos}$, the number of pair of answers with coherent positive answers:

$$n_{pos-pos} = \begin{cases} \binom{pos}{2} & pos >= 2 \\ 0 & otherwise \end{cases} \tag{7.8}$$

  where pos is the number of labels with positive answers for a sample;

- $n_{neg-neg}$, the number of pair of answers with coherent negative answers:

$$n_{neg-neg} = \begin{cases} \binom{neg}{2} & neg >= 2 \\ 0 & otherwise \end{cases} \tag{7.9}$$

  where neg is the number of labels with negative answers for a sample;

- n_{TOT}, the total number of pair of answers.

For the entire dataset of responses, we consider:

- $N_{pos-pos}$, the total number of pair of labels with coherent positive answers:

$$N_{pos-pos} = \sum_{i=1}^{K} n_{pos-pos}[i] \tag{7.10}$$

  where $n_{pos-pos}[i]$ is the number of pair of responses with coherent positive answer for sample i, K is the total number of samples;

- $N_{neg-neg}$, the total number of pair of labels with coherent negative answers:

$$N_{neg-neg} = \sum_{i=1}^{K} n_{neg-neg}[i] \tag{7.11}$$

  where $n_{neg-neg}[i]$ is the number of pair of responses with coherent negative answer for sample i, K is the total number of samples;

- N_{TOT}, the total number of pair of labels in the entire dataset:

$$N_{TOT} = \sum_{i=1}^{K} n_{TOT}[i] \tag{7.12}$$

  where $n_{TOT}[i]$ is the total number of pair of responses for sample i, K is the total number of samples.

The inter-rater agreement can be computed as follows:

$$IRA = \frac{N_{pos-pos} + N_{neg-neg}}{N_{TOT}} \tag{7.13}$$

Normalized inter-rater agreement

Normalized inter-rater agreement is a more precise and useful statistics since it takes into consideration the probability for the raters to give the same answer by chance. In order to compute this value we need to define 3 quantities:

- f_{pos}, the frequency of the positive answer across all answers. Data taken from Table 7.9;
- f_{neg}, the frequency of the negative answer across all answers. Data taken from Table 7.9;
- p_e, which is the probability of two raters giving the same answer by chance:

$$p_e = \sum_{l \in labels} f_l^2 \tag{7.14}$$

  where labels = {pos, neg}.

The formula for the normalized inter-rater agreement is:

$$norm\_IRA = \frac{IRA - p_e}{1 - p_e} \tag{7.15}$$

We can evaluate the statistics meaning given the interval where the value falls, as showed in Table 7.13.

norm_IRA	**Agreement**
< 0	less than chance agreement
0.01 - 0.20	slight agreement
0.21 - 0.40	fair agreement
0.41 - 0.60	moderate agreement
0.61 - 0.80	substantial agreement
0.81 - 0.99	almost perfect agreement

Table 7.13: Meaning of norm_IRA. Table taken from [54].

Statistics of our setting

As previously state in this section, we have collected 925 annotations over 347 samples of our dataset. We consider each video as a sample and every category question as a separate independent question. A user gives positive answer to the question if it detects the inconsistency related to it, negative otherwise. In Table 7.15 we can see the statistics computed on our responses dataset.

Looking at the norm_IRA, the categories with *moderate agreement* are **Obfuscated Face** and **Glasses Inconsistency**. This result is coherent with expectations since these categories should be pretty easily spotted.

Flickering is the only category with *fair agreement*. It is overall a pretty good agreement, considering that some videos are pretty difficult to label and less focused users (or simply less expert users) might find hard to label them. Other videos are instead very evident and therefore a medium value for norm_IRA has to be expected.

Irregular Traits, **Colors Inconsistency** and **Irregular Proportions** are the categories with *slight agreement*. Indeed, the agreement on these categories is little as users tend to give different interpretations to these classes, mostly to the **Irregular Proportions** one. At the same time, these ones are the most difficult categories to be spotted, so a decrease in the agreement is foregone.

Other is the only category with negative norm_IRA. There is no particular agreement on this class, since it is quite arbitrary. Furthermore, the number of annotations were this category is selected is quite small, meaning that this class has not been of particular use for our study.

	$N_{pos-pos}$	$N_{neg-neg}$
Flickering	270	480
Obfuscated Face	163	768
Irregular Traits	118	660
Colors inconsistency	111	654
Irregular Proportions	32	835
Glasses Inconsistency	134	831
Other	8	1012

Table 7.14: $N_{pos-pos}$ and $N_{neg-neg}$ computed for each category.

	IRA(%)	p_e(%)	Norm_IRA
Flickering	65	52.4	0.265
Obfuscated Face	80.6	66	0.43
Irregular Traits	67.4	60.8	0.168
Colors inconsistency	66.3	61	0.135
Irregular Proportions	75.1	73.5	0.06
Glasses Inconsistency	83.6	70	0.45
Other	88.4	89.5	-0.1

Table 7.15: p_e, inter-rater and normalized inter-rater agreement of the annotations collected via Web Interface. To compute this table we use data from Table 7.14.

Flickering detection performance comparison with norm_IRA

Our best flickering detection model obtains 0.76% as overall accuracy. The IRA of the flickering class is 65% so our model outperforms human level agreement in the flickering labeling.

Chapter 8

Conclusion

In this work we have explored the problem of deepfake detection exploiting the time domain focusing on the following research questions:

- *What kind of video-level artifacts can we identify in deepfake videos?*
  - *Which one can we detect automatically?*
  - *What is human level performance on detecting coherently inconsistencies?*
- *Can video-based models outperform frame-based classifiers?*
  - *Can Transformer models reach state of the art results on video-based deepfake detection?*

To answer these questions we have investigated two different approaches to deepfake detection: one based on video-level artifacts/inconsistencies in videos; the other one based on video-level classifiers. Moreover, we have performed a data collection campaign asking users to label deepfake videos according to the detected artifacts.

Studying the DFDC dataset, which is currently the largest deepfake dataset available on the Web, we have found some recurring video-level artifacts and we have decided to name these inconsistencies. There are mainly two reasons for the presence of these artifacts: deepfake generation happening frame-by-frame and failing if the face is not correctly recognized by the face detector.

The categories we have defined are: flickering, obfuscated face, proportions inconsistency, color inconsistency, irregular facial traits and glasses inconsistency.

As a first approach, we have focused on detecting some of these inconsistencies via some deep learning approach, namely convolutional LSTMs. Since there is no available ground truth for the problem we have formulated, we have proposed a way to reproduce flickering by alternating 'original' and fake video frames. Our models have obtained 0.76 accuracy on our manually labeled test set.

As a second approach, we have used a Web Interface to make people annotate some videos. In particular, users were requested to identify (if any) inconsistencies in videos. With the collected data, we first have created a new dataset, composed of users' answers, and then we have used the annotations to study the human performance on detecting artifacts. People moderately agree on detecting obfuscated face and glasses inconsistency, fairly agree on detecting flickering and slightly agree on the other categories. Focusing on flickering and comparing our models results with the human level performance, we can see how our model is overall better in recognizing flickering with respect to humans.

Continuing the study on the time domain, we have proposed and compared several video-level classifiers: we have started with a detailed study on state-of-the-art models, namely convolutional LSTMs, comparing different preprocessings and different hyperparameters configurations. Afterwards, we have shown 2 unseen architectures on deepfake detection: Video Transformer Network and Timesformer. These models are the state of the art in video-level classification tasks but no one has ever tried self-attention video-level models for deepfake recognition. We have examined the performance of these architectures, comparing different preprocessings and hyperarameters configurations.

As we have shown in Chapter 7, so far no video-level model is able to achieve results of state-of-the-art frame-based classifiers on deepfake detection. The result does not come unexpected and confirms the results obtained by the winners of the DFDC, showing how EfficientB4 (or B7) architectures are better. As said in Chapter 7, this result should not mislead since video-level detection is a much more challenging task than the frame-based.

Considering the Transformers architectures, the VTN architecture has achieved state-of-the-art LSTM performance while the Timesformer model have not really achieve good results on the DFDC. As a general consideration, finetuning a convolutional feature extractor seems the way to go to achieve good performances on deepfakes. From the results shown in Chapter 7 we can see how the VTN architecture is much more balanced and has a considerably better performance in classifying pristine samples with respect to the LSTM architectures, while the LSTM architectures are much better at recognizing fake samples.

8.1 Future work

Time domain artifacts and architectures are yet to be explored deeply. In our work, we have given possible starting points for the study of inconsistencies and of self-attention architectures. Some possible research directions could be the following:

- **Collect a larger dataset via Web Interface in order to directly use that ground truth as training set.**

  Via the Web Interface, we have collected a restricted number of annotations. Unfortunately, data was not enough to employ it as a training set, therefore we have used it to test some performances. With more time and data available, it would be interesting to directly train on users' labels in order to try and detect also other inconsistencies that are not synthetically reproducible. As a further study, a comparison between synthetic dataset and user label dataset could be performed on flickering detection.

- **Improve flickering detection.**

  In our work, we have tried reproducing flickering by alternating fake frames with the frames from the 'original' video, stabilizing the detection in order to avoid introducing wrong abrupt changes. It would be interesting seeing other ways to reproduce this effect, trying to distinguish cases where there is flickering and where the face simply turns in the video.

- **Explore other Transformer architectures.**

  In our work, we have presented 2 Transformers architectures to deal with deepfake detection but the number of new architectures coming up is huge. With more available time, it would be interesting to try new architectures (or a combination of them) to see whether or not they can outperform state-of-the-art models.

Bibliography

[1] Somers. Deepfakes, explained. 2020. URL `https://mitsloan.mit.edu/ideas-made-to-matter/deepfakes-explained`.

[2] Supasorn Suwajanakorn, Steven M Seitz, and Ira Kemelmacher-Shlizerman. Synthesizing obama: learning lip sync from audio. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.

[3] Jennifer Walter. Deepfakes: The dark origins of fake videos and their potential to wreak havoc online. 2020. URL `https://www.discovermagazine.com/technology/deepfakes-the-dark-origins-of-fake-videos-and-their-potential-to-wreak-havoc`.

[4] Daniel Neimark, Omri Bar, Maya Zohar, and Dotan Asselmann. Video transformer network. *arXiv preprint arXiv:2102.00719*, 2021.

[5] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? *arXiv preprint arXiv:2102.05095*, 2021.

[6] Momina Masood, Marriam Nawaz, Khalid Mahmood Malik, Ali Javed, and Aun Irtaza. Deepfakes generation and detection: State-of-the-art, open challenges, countermeasures, and way forward. *arXiv preprint arXiv:2103.00484*, 2021.

[7] Yuval Nirkin, Iacopo Masi, Anh Tran Tuan, Tal Hassner, and Gerard Medioni. On face segmentation, face swapping, and face perception. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pages 98–105. IEEE, 2018.

[8] KR Prajwal, Rudrabha Mukhopadhyay, Vinay P Namboodiri, and CV Jawahar. A lip sync expert is all you need for speech to lip generation in the wild. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 484–492, 2020.

[9] Justus Thies, Michael Zollhöfer, Christian Theobalt, Marc Stamminger, and Matthias Nießner. Headon: Real-time reenactment of human portrait videos. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.

[10] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2387–2395, 2016.

[11] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[12] Thanh Thi Nguyen, Cuong M Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, and Saeid Nahavandi. Deep learning for deepfakes creation and detection: A survey. *arXiv preprint arXiv:1909.11573*, 2019.

[13] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings, 2012.

[14] David Güera and Edward J Delp. Deepfake video detection using recurrent neural networks. In *2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS)*, pages 1–6. IEEE, 2018.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[16] Deepfake. 2020. URL `https://en.wikipedia.org/wiki/Deepfake`.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[21] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. *arXiv preprint arXiv:2103.15691*, 2021.

[22] Dennis Siegel, Christian Kraetzer, Stefan Seidlitz, and Jana Dittmann. Media forensics considerations on deepfake detection with hand-crafted features. *Journal of Imaging*, 7(7):108, 2021.

[23] Xin Yang, Yuezun Li, and Siwei Lyu. Exposing deep fakes using inconsistent head poses. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8261–8265. IEEE, 2019.

[24] Shruti Agarwal, Hany Farid, Yuming Gu, Mingming He, Koki Nagano, and Hao Li. Protecting world leaders against deep fakes. In *CVPR workshops*, volume 1, 2019.

[25] Scott McCloskey and Michael Albright. Detecting gan-generated imagery using color cues. *arXiv preprint arXiv:1812.08247*, 2018.

[26] Pavel Korshunov and Sébastien Marcel. Deepfakes: a new threat to face recognition? assessment and detection. *arXiv preprint arXiv:1812.08685*, 2018.

[27] Huy H Nguyen, Junichi Yamagishi, and Isao Echizen. Capsule-forensics: Using capsule networks to detect forged images and videos. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2307–2311. IEEE, 2019.

[28] Darius Afchar, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Mesonet: a compact facial video forgery detection network. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7. IEEE, 2018.

[29] Nicolò Bonettini, Edoardo Daniele Cannas, Sara Mandelli, Luca Bondi, Paolo Bestagini, and Stefano Tubaro. Video face manipulation detection through ensemble of cnns. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 5012–5019. IEEE, 2021.

[30] Yuezun Li and Siwei Lyu. Exposing deepfake videos by detecting face warping artifacts. *arXiv preprint arXiv:1811.00656*, 2018.

[31] Hyeonseong Jeon, Youngoh Bang, and Simon S Woo. Fdftnet: Facing off fake images using fake detection fine-tuning network. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 416–430. Springer, 2020.

[32] Md Shohel Rana and Andrew H Sung. Deepfakestack: A deep ensemble-based learning technique for deepfake detection. In *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 70–75. IEEE, 2020.

[33] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics: A large-scale video dataset for forgery detection in human faces. *arXiv preprint arXiv:1803.09179*, 2018.

[34] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017.

[35] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics: A large-scale

video dataset for forgery detection in human faces. *arXiv preprint arXiv:1803.09179*, 2018.

[36] Pavel Korshunov and Sébastien Marcel. Deepfakes: a new threat to face recognition? assessment and detection. *arXiv preprint arXiv:1812.08685*, 2018.

[37] Irene Amerini, Leonardo Galteri, Roberto Caldelli, and Alberto Del Bimbo. Deepfake video detection through optical flow based cnn. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[38] Steven S. Beauchemin and John L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466, 1995.

[39] Deressa Wodajo and Solomon Atnafu. Deepfake video detection using convolutional vision transformer. *arXiv preprint arXiv:2102.11126*, 2021.

[40] Amritpal Singh, Amanpreet Singh Saimbhi, Navjot Singh, and Mamta Mittal. Deepfake video detection: A time-distributed approach. *SN Computer Science*, 1(4):1–8, 2020.

[41] Ekraam Sabir, Jiaxin Cheng, Ayush Jaiswal, Wael AbdAlmageed, Iacopo Masi, and Prem Natarajan. Recurrent convolutional strategies for face manipulation detection in videos. *Interfaces (GUI)*, 3(1):80–87, 2019.

[42] Akash Chintha, Bao Thai, Saniat Javid Sohrawardi, Kartavya Bhatt, Andrea Hickerson, Matthew Wright, and Raymond Ptucha. Recurrent convolutional structures for audio spoof and video deepfake detection. *IEEE Journal of Selected Topics in Signal Processing*, 14(5):1024–1037, 2020.

[43] Yuezun Li, Ming-Ching Chang, and Siwei Lyu. In ictu oculi: Exposing ai created fake videos by detecting eye blinking. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7. IEEE, 2018.

[44] Ipek Ganiyusufoglu, L Minh Ngô, Nedko Savov, Sezer Karaoglu, and Theo Gevers. Spatio-temporal features for generalized detection of deepfake videos. *arXiv preprint arXiv:2010.11844*, 2020.

[45] Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, and Matthias Grundmann. Blazeface: Sub-millisecond neural face detection on mobile gpus. *arXiv preprint arXiv:1907.05047*, 2019.

[46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[47] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[48] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[49] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) dataset. *arXiv preprint arXiv:2006.07397*, 2020.

[50] Joao Carreira, Eric Noland, Andras Banki-Horvath, Chloe Hillier, and Andrew Zisserman. A short note about kinetics-600. *arXiv preprint arXiv:1808.01340*, 2018.

[51] Dong Huang and Fernando De La Torre. Facial action transfer with personalized bilinear regression. In *European Conference on Computer Vision*, pages 144–158. Springer, 2012.

[52] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9459–9468, 2019.

[53] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

[54] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.