Department of Digital Systems
Piraeus, Greece

Department of Informatics
Milan, Italy

MSc Thesis
in
Big Data and Analytics

Deepfake Detection with Deep Learning: Leveraging Remote Heart Rate Estimation to discriminate real from fake videos of faces

Spanidis Loukas

Supervisors: Professor Raffaella Lanzarotti
Department of Informatics
University of Milan

Professor Ilias Magklogiannis
Department of Digital Systems
University of Piraeus

2023

Spanidis Loukas

Deepfake Detection with Deep Learning: Leveraging Remote Heart Rate Estimation to discriminate real from fake videos of faces

2023

Supervisor: Professor Raffaella Lanzarotti
Co-Supervisor : Professor Ilias Magklogiannis

University of Milan
Department of Informatics
Milan, Italy

University of Piraeus
Department of Digital Systems
Piraeus, Greece

Abstract

Deepfakes are swiftly becoming a reality. These media are generated by very powerful tools which allow fast and cheap media manipulation. These videos can be used for unethical and malicious applications, such as spread of fake news or fake porn. Given the upcoming problems these media pose, researchers have started working on different ways to try and detect these types of media. This master's thesis investigates the temporal aspect of deepfake detection, with a specific emphasis on exploring the physiological artifacts associated with the corruption of physiological signals, such as heart rate variability, obtained through the pyVHR framework. The study employs several state-of-the-art Convolutional Neural Network (CNN) models, including AlexNet, ResNet, and SincNet, along with various modifications and different data preprocessing techniques. These models are trained and evaluated on the FaceForensics+ dataset, providing a comprehensive evaluation of their effectiveness in deepfake detection. The results of this investigation demonstrate that the SincNet model outperforms other models across different datasets within the FaceForensics++ dataset. Notably, it achieves an impressive accuracy of 95.8% and an F1-score of 95.9% when trained on the FaceShifter dataset. However, the performance of the model varies depending on the specific dataset used for training, indicating the importance of dataset selection in deepfake detection. The results highlight the potential of using physiological signals, extracted from the pyVHR framework, in detecting deepfake videos. Remarkably, the SincNet model demonstrated the ability to effectively distinguish between real and fake videos using only the BVP signals extracted from the pyVHR framework.

Περίληψη

Τα Deepfakes γίνονται γρήγορα πραγματικότητα. Αυτά τα μέσα δημιουργούνται από πολύ ισχυρά εργαλεία που επιτρέπουν γρήγορο και φθηνό χειρισμό μέσων. Τα βίντεο αυτά μπορούν να χρησιμοποιηθούν για ανήθικες και κακόβουλες εφαρμογές, όπως η διάδοση ψεύτικων ειδήσεων ή ψεύτικο πορνό. Δεδομένων των επερχόμενων προβλημάτων που θέτουν αυτά τα μέσα, ερευνητές έχουν αρχίσει να εργάζονται με διαφορετικούς τρόπους για να προσπαθήσουν να εντοπίσουν αυτούς τους τύπους μέσων. Αυτή η διπλωματική εργασία διερευνά τη χρονική προσέγγιση της ανίχνευσης deepfake, με ιδιαίτερη έμφαση στην εξερεύνηση των φυσιολογικών τεχνουργημάτων που σχετίζονται με την καταστροφή των φυσιολογικών σημάτων, όπως η μεταβλητότητα του καρδιακού ρυθμού, που λαμβάνονται μέσω του πλαισίου pyVHR. Η μελέτη χρησιμοποιεί πολλά μοντέλα τελευταίας τεχνολογίας Συνελικτικού Νευρωνικού Δικτύου (CNN), συμπεριλαμβανομένων των AlexNet, ResNet και SincNet, μαζί με διάφορες τροποποιήσεις και διαφορετικές τεχνικές προεπεξεργασίας δεδομένων. Αυτά τα μοντέλα εκπαιδεύονται και αξιολογούνται στο σύνολο δεδομένων FaceForensics+, παρέχοντας μια ολοκληρωμένη αξιολόγηση της αποτελεσματικότητάς τους στον εντοπισμό του deepfake βίντεο. Τα αποτελέσματα αυτής της έρευνας δείχνουν ότι το μοντέλο SincNet έχει καλύτερη απόδοση από άλλα μοντέλα σε διαφορετικά σύνολα δεδομένων στο σύνολο δεδομένων FaceForensics++. Συγκεκριμένα, επιτυγχάνει εντυπωσιακή ακρίβεια 95,8% και F1-score 95,9% όταν εκπαιδεύεται στο σύνολο δεδομένων FaceShifter. Ωστόσο, η απόδοση του μοντέλου ποικίλλει ανάλογα με το συγκεκριμένο σύνολο δεδομένων που χρησιμοποιείται για την εκπαίδευση, υποδεικνύοντας τη σημασία της επιλογής δεδομένων στην ανίχνευση deepfake βίντεο. Τα αποτελέσματα υπογραμμίζουν τη δυνατότητα χρήσης φυσιολογικών σημάτων, που εξάγονται από το εργαλείο pyVHR, για την ανίχνευση deepfake βίντεο. Είναι αξιοσημείωτο ότι το μοντέλο SincNet έδειξε την ικανότητα αποτελεσματικής διάκρισης μεταξύ πραγματικών και ψεύτικων βίντεο χρησιμοποιώντας μόνο τα σήματα BVP που εξάγονται από το εργαλείο pyVHR.

Acknowledgements

I am deeply grateful to my supervisors, Professor Raffaella Lanzarotti, for her exceptional guidance and mentorship throughout this master's thesis. Moreover, I want to express my heartfelt appreciation to her for suggesting the thesis topic. I would like to extend a special appreciation to Assistant Professor Alessandro D'Amelio, whose expertise and encouragement have been invaluable in the development of this work. Your thoughtful inputs and suggestions have played a crucial role in refining the research methodology and analysis. I also want to thank my co-supervisor, Professor Ilia Maglogianni, for their contributions and valuable insights to this research. I would also like to thank the University of Milan and the University of Piraeus for their support and access to resources, which have been essential for conducting this thesis. I would also like to express my gratitude to all my fellow students and friends for their mutual help and support throughout this journey. Last but not least, I want to wholeheartedly thank my family for their endless support and encouragement during all these years of studying. Your belief in me has been my driving force, and I am forever grateful for your love and support.

Contents

1 Introduction

Video manipulation refers to the act of altering or changing the content of a video for the purpose of misleading or deceiving the audience. This can be achieved through various techniques such as splicing, editing and adding or removing objects. Nowadays, media manipulation is becoming cheaper and cheaper. With the advancements in digital video editing software, it has become easier for individuals to create and distribute manipulated videos. This has raised serious concerns about the spread of false information and the impact it can have on public opinion and decision-making. 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 derived from combining the words "deep learning" and "fake," and it is an artificial intelligence-powered video manipulation technique that use deep learning algorithms to generate realistic-looking photos and videos of individuals doing or saying things they never actually did. This technique may be used to generate videos that represent actual individuals in imaginary circumstances, making it difficult to tell the difference between real and fake material. Deepfakes have been a source of worry due to their ability to spread misleading information and affect public opinion. Deepfakes have the potential to be exploited for evil reasons such as extortion, misinformation, and propaganda, which has alarmed both the public and corporate sectors.

Deepfakes have a negative impact on a variety of domains, including politics, organizations, and the development of non-consensual explicit content. Deepfakes can be used in politics to create fabricated video of politicians, which can spread false information, manipulate public opinion, and disrupt the democratic process. Such malicious use of deepfakes can undermine election integrity, with serious social and political implications. Furthermore, deepfakes can have negative effects on organizations. For instance, they can be used as a tool for blackmailing managers or executives by creating fake videos that make them appear involved in unethical situations. This can lead to reputational damage, financial losses, and weakened organizational decision-making. Additionally, the creation of non-consensual explicit content through deepfakes, commonly known as "deepfake porn," can cause significant emotional pain, humiliation, and reputational damage on the targeted individuals, including employees of organizations. Deepfake porn can violate personal privacy, consent, and dignity, harassment and abuse, which can have consequences for the victims' well-being and professional lives[16].

As deepfake technology continues to advance and become more accessible, it is crucial for individuals to be able to recognize manipulated videos and understand the dangers they pose to society. To limit the spread of video manipulation, effective tools and technologies for detecting and preventing the spread of false information must be developed.

1.1 Motivation and Prob Statement

According to [16], deepfakes often produce arifacts that may be difficult for humans to notice but can be easily detected through machine learning and forensic analysis. Several studies have focused on identifying deepfakes by searching for specific artifacts. These artifacts can be categorized into seven types: spatial artifacts in blending, environments, and forensics; temporal artifacts in behavior, physiology, synchronization, and coherence.

- **Blending (spatial)**. Spatial artifacts that occur when the generated content is blended back into the frame;
- **Environment (spatial)**. The facial features of a fake face can appear inconsistent or abnormal compared to the surrounding elements within the frame.
- **Forensic (spatial)**. Spatial artifacts that have abnormalities or distortions in a video when subtle features or patterns are added by the model (e.g. inconsistent head poses).
- **Behavior (temporal)**. Temporal artifacts that result from unnatural mannerisms or other human behaviors introduced by the model;
- **Physiological (temporal)**. Computer-generated content have disruptions in the physiological signals that are present in the human faces(e.g. heart beat, blood flow, breathing).
- **Synchronization (temporal)**. Inconsistencies concerning speech to landmarks on the mouth area, including differences between visemes and phonemes in fake videos;
- **Coherence (temporal)**. Disrupted coherence between consecutive frames (e.g. flickers)

The primary objective of this research is to address research questions concerning deepfake detection. Specifically, the following questions were explored:

1. Can the distinction between real and fake videos be achieved by utilizing physiological signals extracted from the videos as input to the detection models?
2. What is the performance difference between using raw BVP signals and spectrogram representations as input for 2D CNN models in the deepfake detection?
3. Can the proposed model predict more accurately when dealing with more realistic videos manipulated algorithms?

The primary focus of this thesis is to investigate the temporal aspect of deepfake detection, with a specific emphasis on exploring the physiological artifacts associated with the corruption of physiological signals, such as heart rate variability. The proposed method concerns around the implementation of several signal-level models. For the purpose of training and testing, the FaceForensics+ dataset is used. At first, state-of-the-art CNN models, including AlexNet, ResNet, and SincNet, are introduced along with various modifications and different data preprocessing techniques. These variations involve both raw signal classification and the creation of spectrograms derived from the signals. The training and testing processes of these models are described, with a careful comparison of their performance across all the proposed approaches.

1.2 Thesis Structure

The rest of the thesis is composed of five chapters.

Chapter 2 presents the background needed for the understanding of this thesis. It covers key aspects such as the dataset used, the feature extraction techniques employed, the CNN models utilized, and the evaluation measures employed.

Chapter 3 displays various related works on deepfake detection approaches. This chapter analyze different methodologies and techniques employed by researchers in tackling the deepfake detection challenge.

Chapter 4 provides the methods and pipeline employed in this study. It offers detailed specifications of the models utilized, along with the data prepossessing techniques.

Chapter 5 discusses the training process and the specific training parameters employed in this study. It provides details of how the models were trained, optimizing their performance to achieve accurate deepfake detection. Additionally, this chapter presents the corresponding results obtained from the training process.

Chapter 6 serves as the conclusion of this thesis, summarizing the key findings, highlights, limitations, and suggestions for future work.

2 Background

This section provides important background concepts for the understanding of this thesis. Initially, a brief explanation about deepfakes generation tools (deepfake algorithms) will be introduced, followed by an overview of the techniques employed for features extraction. Then, an introduction on some useful deep learning techniques and architectures will be provided, along with the evaluation measures that will be used during the thesis.

2.1 Deepfake Generation

The FaceForensics++ dataset [19] is a collection of 1000 original video sequences designed for forensic analysis. The dataset consists of videos sourced from 977 YouTube videos with a resolution greater than 480p. These videos feature mostly frontal faces without occlusions, making them suitable for automated tampering methods to generate realistic forgeries. To create a large scale manipulation database, cutting-edge video editing techniques were adapted for fully automated operation. The dataset includes three graphics-based approaches (Face2Face, FaceShifter, and FaceSwap) and two learning-based approaches (DeepFakes and NeuralTextures). All five methods require pairs of source and target actor videos as input. The output of each method is a video composed of generated images. In the following paragraphs, these methods will be briefly described.

1. **Deepfakes:** Despite the commonly use of the word "Deepfakes" as the general technique of face replacement, it is also a specific manipulation method. Within the FaceForensics++ dataset, the implementation of that method is from the faceswap GitHub repository, accessible here. This implementations involve replacing a face in a target video with a face observed in a source video or image collection. The method relies on two autoencoders with a shared encoder that are trained to reconstruct images of the source and target faces.
2. **Face2Face:** Face2Face [20] is a real-time facial approach that transfers the expressions of a source video to a target video while keeping the identity of the target person. This method aims to recover facial identity from the video and tracks the facial expressions of both the source and target videos using a dense photometric consistency measure, which allows for accurate tracking of facial movements. Once the facial expressions are tracked, the technique transfers the expressions from the source actor to the target

video in a photo-realistic fashion, ensuring that the manipulated output video appears natural and seamless.

3. **FaceSwap:** FaceSwap accordig to Rössler et al. [19] is a graphics-based approach used to transfer the face region from a source video to a target video. The process begins by detecting sparse facial landmarks in both videos. These landmarks are then used to extract the face region. This approach fits a 3D template model with blendshapes using the observed landmarks. This model is then projected onto the target image by minimizing the difference between the projected shape and the localized landmarks. The textures of the input image are used throughout this phase to improve the visual coherence. Finally, the rendered model is blended with the target image, and color correction is performed. These steps are repeated for all pairs of source and target frames until the video sequence is complete.
4. **NeuralText:** NeuralTextures, as demonstrated by Thies et al. [21], uses facial reenactment as an example of the rendering approach. The method leverages the original video data to train a neural texture model specific to the target person, which includes a rendering network. This approach is implemented by a patch-based GAN-loss approach. NeuralTextures are composed of a set of optimal feature maps, rather than simple RGB values, that are learned during the scene capture process. The NeuralTextures technique relies on tracked geometry, which is used during both training and testing. It is important to note that the facial expressions in the mouth region was only modified, while the eye region was kept unchanged.
5. **FaceShifter:** FaceShifter [11] is a two-stage framework designed for high-quality and occlusion-aware face swapping. In the initial stage, the framework generates the swapped face by effectively utilizing and combining target attributes. In the second stage, the framework address the problem of face occlusions using a specialized network called Heuristic Error Acknowledging Refinement Network (HEAR-Net). The results of this approach have visually appealing outcomes and notable advancements in the field of face swapping.

   The proposed framework exhibits exceptional performance in generating highly realistic face images without the need for subject-specific training. Through extensive experimentation, the FaceShifter algorithm significantly outperforms earlier face swapping techniques. Figure 2.1 shows some examples obtained using the framework, highlighting the superior capabilities in generating convincing and authentic face swaps across various face pairs. These evidence depicted in the Figure 2.1 further supports the advancements achieved by the FaceShifter algorithm in the field of face swapping.

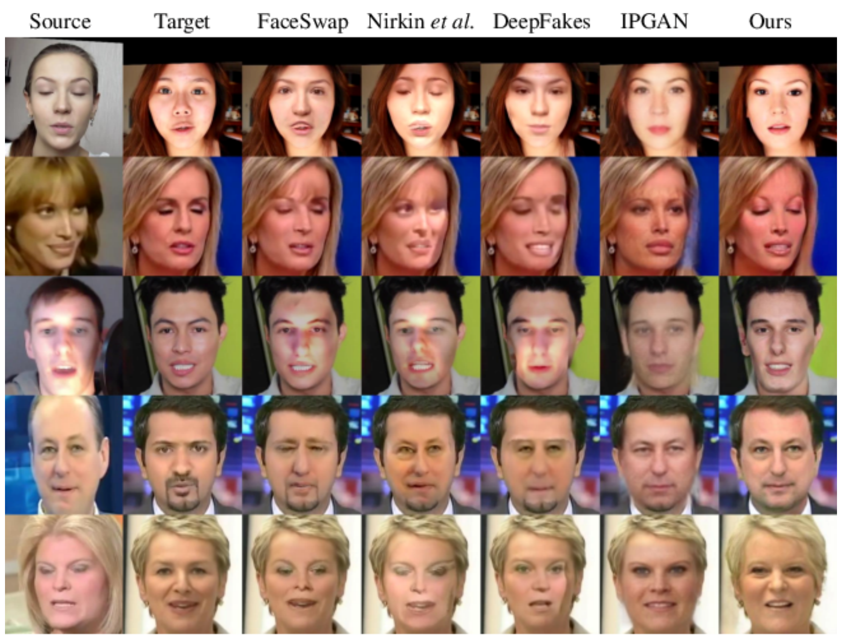


Fig. 2.1: Comparison of the FaceShifter framework to some of the other face swapping algorithms. Clearly, FaceShifter framework shows greater realism to the target characteristics, such as lighting and image resolution. Source: Li et al. [11]

2.2 Feature Extractions

2.2.1 PPG

Photoplethysmography (PPG) as it described in [1] is a non-invasive optical technique used to detect blood volume changes in peripheral blood vesselss. The technique involves a light source and a photodetector on the skin, usually a LED, and measuring the light that is transmitted or reflected from the skin. An example is depicted in Figure 2.2 As the volume of blood in the peripheral blood vessels changes with each heartbeat, the amount of light transmitted or reflected also changes, resulting in a waveform known as a photoplethysmogram. This waveform has gained significant attention from researchers worldwide, as it holds the potential to provide valuable information in addition to heart rate estimation and pulse oxymetry readings.

However, these signals are mostly measured from skin-contact ECG/BVP sensors, which may cause discomfort and are inconvenient for long-term monitoring. To solve this problem, remote photoplethysmography (rPPG) , which targets to measure heart activity remotely and without any contact, has been developing rapidly in recent years.

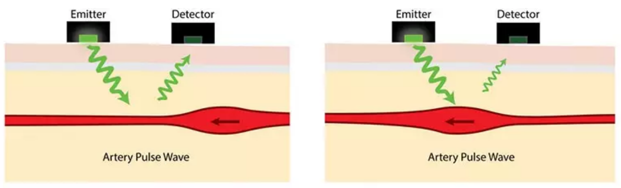


Fig. 2.2: Optical heart rate sensing uses light to analyze pulse waves. Narrower arteries on the left side reflect more green light (higher reflectivity), indicating lower pressure before the pulse wave. Wider arteries on the right side absorb more light (lower reflectivity), indicating a higher blood pressure pulse. Source: `https://www.noldus.com/blog/what-is-rppg`

2.2.2 rPPG

Remote-photoplethysmography (rPPG) is an extension of PPG that allows for the measurement of blood volume changes in peripheral blood vessels using a camera and light source at a distance (contactless) from the subject. As an extension of the PPG, rPPG can be used to estimate parameters such as heart rate, arterial pressure, blood glucose level or oxygen saturation level. In more details, according to Wang et al. [23], remote photoplethysmography (rPPG) is a non-contact technique for monitoring a person's pulse rate with a multi-wavelength RGB camera. The technology detects the small color differences caused by pulses on the surface of human skin. An example of these reflections is depicted in Figure 2.3. The video of the person's face is captured by the camera, and the pulse signal is extracted from the video by the rPPG algorithm. For extracting the pulse signal from a video, there are numerous main rPPG approaches. Blind Source Separation (BSS), Chrominance-based rPPG (CHROM), Projection onto Blood Volume Pulse (BVP), and Spatial Subspace Rotation (2SR) are a few examples. The primary distinction between these rPPG approaches is how RGB-signals are combined into a pulse signal. The main purpose of the rPPG in this thesis is to analyze and extract feature (estimation of the heart rate variability) concerning the Blood Volume Pulse signals.

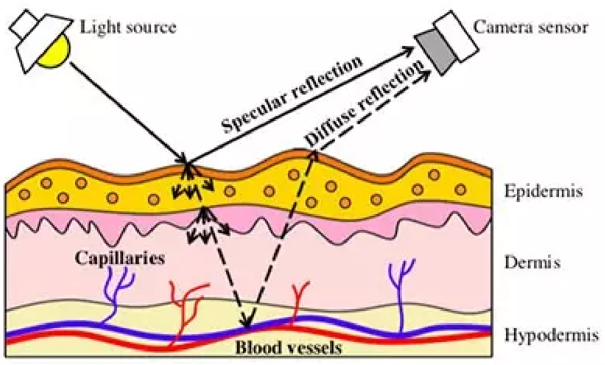


Fig. 2.3: In the skin reflection model, there are two types of reflections: specular and diffuse. Among these, only the diffuse reflection carries pulsatile information. Source: Wang et al. [23]

2.2.3 pyVHR

PyVHR is an open-source Python framework by Boccignone et al. [2] for analyzing Heart Rate Variability (HRV) by extracting remote photoplethysmography (rPPG) signals from videos, particularly videos of subjects' facial skin. pyVHR framework is a multi-stage pipeline that includes the entire process of extracting and analyzing HR fluctuations. It is specifically developed to allow HRV analysis in theoretical research as well as practical applications where the use of wearable sensors would be inconvenient or impracticable. The pyVHR framework differentiates between two commonly used extraction approaches:

- **Holistic:** This method in pyVHR involves extracting the complete skin region from the subject's face in each video frame. Because of its simplicity, this method is widely used in controlled conditions. However, it can result in significant variations in skin tone and shading effects when illumination conditions are unstable.
- **Patches:** This method in pyVHR involves selecting a region of interest (ROI) consisting of multiple patches placed on specific landmarks of the subject's face in each video frame. This approach is more comlex than the holistic method but offers advantages such as the ability to exclude regions affected by shadows or poor lighting conditions. Furthermore, the patches method have more reliable estimations due to the increased number of observations that can be obtained.

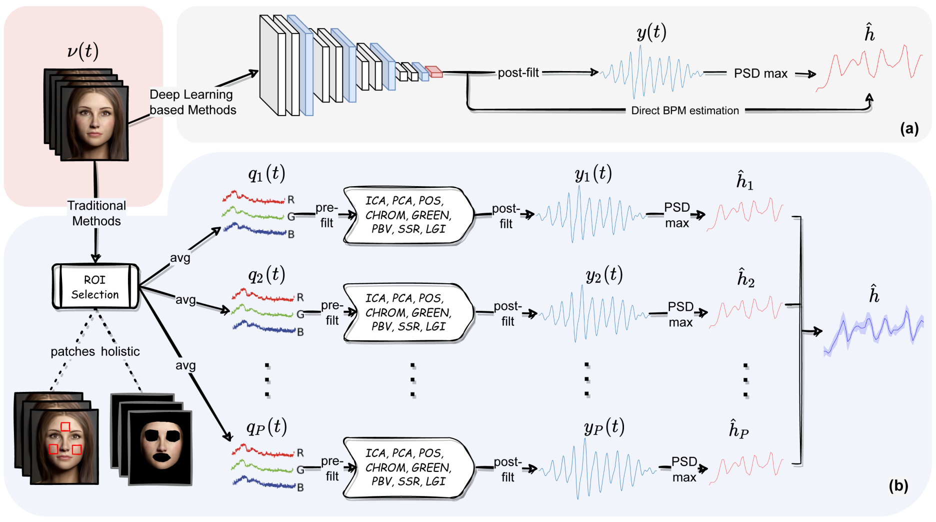


Fig. 2.4: The pyVHR pipeline. (A) The pyVHR framework implements a multi-stage pipeline that utilizes end-to-end deep learning (DL) methods for estimating Beats Per Minute (BPM) through Power Spectral Density (PSD) analysis. (B) For traditional approaches, the pyVHR framework follows a multi-stage pipeline that involves windowing and patch collection, computation of RGB traces, pre-filtering, application of an rPPG algorithm to estimate a Blood Volume Pulse (BVP) signal, post-filtering, and finally, BPM estimation through PSD analysis. Source: [2]

The following section provide a detailed description of all the different modules that constitute the pyVHR pipeline. These modules are shown in Figure 2.4 and can be summarized as follows:

1. **Skin extraction:** In the first step, the goal is to segment the face skin to extract areas related to PPG. This can be done either by collecting these areas in a single patch (holistic approach) or by using multiple sparse patches that cover the whole face (patch-wise approach). In general, the analysis in pyVHR excludes the regions corresponding to the eyes and mouth. This can be achieved by using two different methods in pyVHR shown in Figure 2.5 and summarized as follows:
   - **Convex-hull:** This extractor uses the entire face and subtracts the ones calculated based on the landmarks related to the eyes and mouth. This method creates a mask that isolates the pixels connected with the skin.
   - **Face parsing:** This extractor performs a semantic segmentation of the subject's face. It generates pixel-wise label maps for various semantic components such as hair, mouth, eyes and nose. This allows to keep only the label mappings connected with the skin regions.

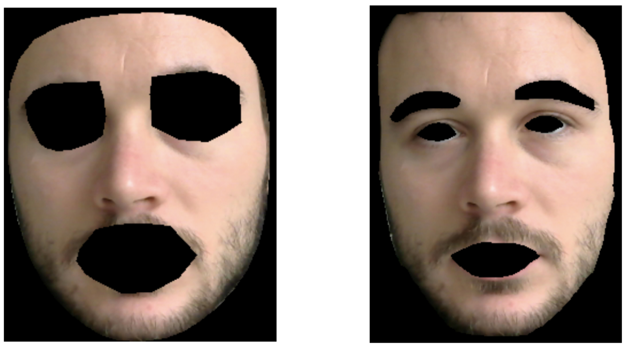


Fig. 2.5: Comparison of the two skin extraction methods implemented in pyVHR: (A) Convex-hull extraction approach and (B) Face parsing extraction approach. Source: [2]

2. **RGB signal processing:** Once the patches (skin regions of the face) have been selected, they are tracked and used to calculate the average color intensities along overlapping windows. This generates multiple time-varying RGB signals for each window. The Figure 2.5 illustrate the Patch tracking on this process. In more details, the RGB signal $q_i(t)$ is computed averaging N_i pixels $p_{ij}(t)_{y=1}^{N_i}$ of the pixel intensity $p_{ij}\epsilon[0, 255]^3$ each belonging to then $i-th$ patch at time t:

$$q_i(t) = \frac{1}{N_i}\sum_{j=1}^{N_i} p_{ij}(t) \tag{2.1}$$

Where $i = 1...P$ (P is the number of the overlapping patches, P=1 for the 'holistic' approach). Then the RGB signal of the $i - th$ patch $q_i(t)$ is sliced into k overlapping time windows $M = W_s * F_s$, where W_s is the window length in seconds and F_s is the sampling rate. Then the final equation is:

$$q_i^k(t) = q_i(t)w(t - k\tau F_s) \tag{2.2}$$

Where $\tau < W_s$ is the overlapping time and w is the rectangular window:
$$w(t) = \begin{cases} 1, 0 \leq t \leq M \\ 0, otherwise \end{cases}$$

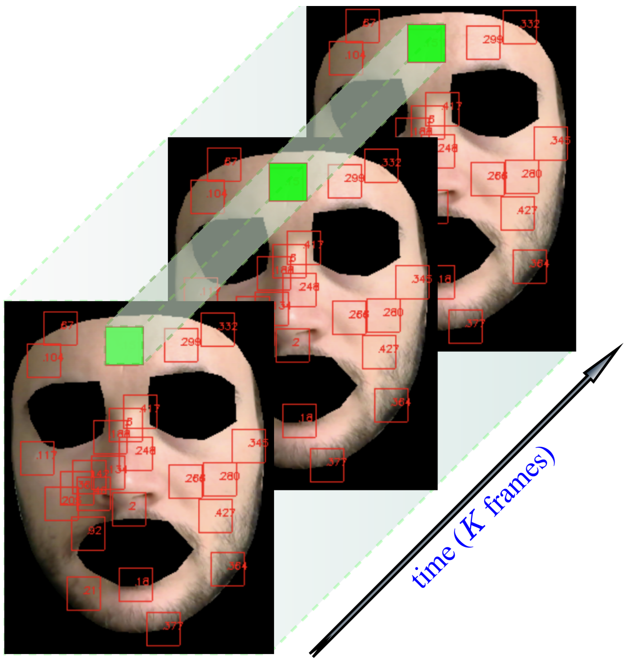


Fig. 2.6: Patch tracking of a temporal window. Source: [2]

3. **Pre-filtering:** The raw RGB traces $q_i(t)$ can be optionally preprocessed by applying intensity thresholds such as canonical filtering, normalization, or de-trending to limit values outside the RGB colors interval. The resulting signals serve as inputs for any subsequent rPPG method.
4. **BVP extraction:**
   The rPPG method is applied to the time-windowed signals to produce a collection of heart rate pulse signals (BVP estimates) for each patch. Specifically, the rPPG method receives windowed RGB traces $q_i(t)$ as input. These traces have a length of K and are processed to produce the estimated BVP signals $y_i(t)$ corresponding to the $i - th$ patch in the $k - th$ time window. There are various methods implemented by pyVHR in order to convert the RGB signals into a pulse signal (BVP), which are listed below:
   - **POS**
   - **GREEN**

- **CHROM**
- **PCA**

5. **Post-filtering:** Optionally, the computed BVP signals from the above mentioned methods can be pass to a Band-Pass filter to exclude frequencies that fall outside the normal range of heart rates.
6. **BPM estimation:** Finally, a beats per minutes (BPM) estimate is derived using basic statistical analysis based on the peak points of the BVP Power Spectral Densities (PSD) using Discrete time Fourier transform (DFT).

This project used the traditional approach of the pyVHR framework. It involved using the Face Parsing extractor for skin extraction and employing the Patches approach with 100 patches on the face. During RGB signal processing, a window length of 6 seconds with an overlapping parameter of 1 second was used. The Plane Orthogonal Skin (POS) method was employed to convert the RGB signals into BVP signals, and an example of a predicted BVP signal using the POS method is shown in Figure 2.7. A Band-Pass filter also used, and only the selected BVP signals were retained for the following analysis and predictions. It is important to note that the final step of the pyVHR framework, which involves BPM estimation, was not implemented in this project.

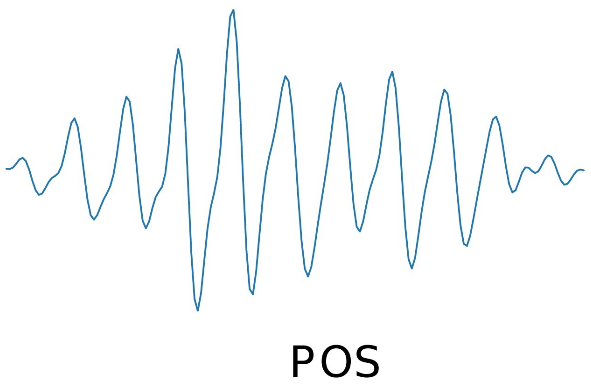


Fig. 2.7: Example of a predicted BVP signals using POS method. Source: [2]

2.2.4 Spectogram

A spectrogram, according to Manhertz and Bereczky [15] is a visual representation of the distribution of signals across frequencies and over time. The vertical axis of a spectrogram typically represents time, the horizontal axis represents the discrete frequency steps, and the amount of power detected is represented as the intensity at each time-frequency point. The method to create spectograms is called short-time Fourier transform (STFT). The STFT is built upon the Discrete Fourier Transform (DFT), which captures the frequency and phase elements of a specific portion of a time-varying signal. The STFT focuses on analyzing a small portion of a longer signal by calculating its Fourier transform. This is done by multiplying the longer time function $x[n]$ by a window function $w[n]$. There are two commonly used window functions: the rectangular window, which extracts the desired segment as it is, and the Hamming window, which applies a smoothing effect at the edges to improve the frequency representation in the transformed domain. The mathimatical representation of the discrete STFT can be descrived as follows

$$STFTx[n](m,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega m} \tag{2.3}$$

where $x[n]$ is the signal, $w[n-m]$ is the window function, j is the complex unit, m is the time (discrete), ω is the frequency (continuous) and n is the chunk. The magnitude squared of the $STFTx[n](m,\omega)$ produce the spectrogram representation of the power spectral density of the function.

An example of the STFT is depicted in the Figure 2.8, illustrating a speech signal along with its corresponding spectrogram.

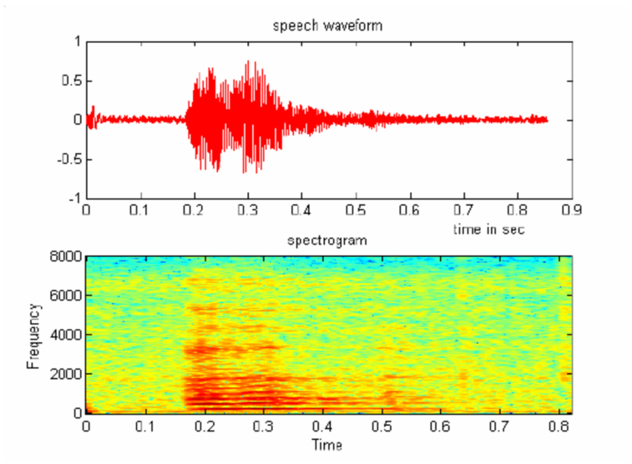


Fig. 2.8: The speech signal (top figure) and its spectrogram (bottom figure) of the vowel “A”: female sound. Source: [3]

2.3 Deep Learning Techniques

2.3.1 Artificial Neural Network

Artificial Neural Networks (ANNs) are computational models inspired by the structure and functioning of the human brain. They are widely used in machine learning and deep learning to solve complex problems and make predictions. The network’s architecture typically includes an input layer, one or more hidden layers, and an output layer. An example of an ANN is shown in Figure 2.9. Each layer has interconnected nodes called artificial neurons. Each neuron receives input signals, applies weights to them, and passes the result through an activation function, producing an output.

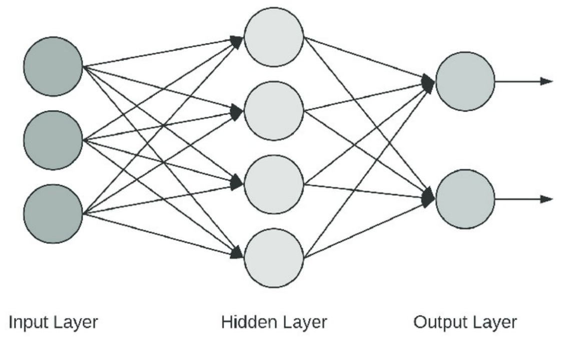


Fig. 2.9: Example of an Artificial Neural Network Model with 3 input neurons, one hidden layer with 4 neurons and an output layer with two neurons. Source: [22]

2.3.2 Convolutional Neural Network

The Convolutional Neural Network (CNN) [17] is a type of deep neural network designed mainly to deal with images. Although, this architecture is also used in different forms of arrays: 1D for signals and sequences, 2D for images and audio spectograms and 3D for videos. The architecture is composed of three parts: the convolutional layer, pooling layer and the fully-connected layer (feed-forward part). An example of a typical 1D CNN architecture is shown in the Figure 2.10. In this architecture, the convolutional layer receives input data in the form of a vector and applies convolution using a 1 × N type kernel. The feature map is created as one-dimensional data. A Fully-connected layer that is created similarly to a traditional Artificial Neural Network (ANN) is used in the learning process. Backpropagation is used for learning, where optimization techniques such as stochastic gradient descent (SGD) based on the gradient descent method or the Adam optimizer are employed. These techniques make it easier to adjust the network's weights and biases in order to reduce error and improve model performance.

- **Convolutional Layer:** The convolutional layer is a fundamental component of CNNs that plays a crucial role in processing and extracting features from input data. It consists of a set of learnable filters or kernels that slide across the input data, performing convolutions to generate feature maps. During the convolution operation, the filters is applied to local patches of the input and extract different feature and create multiple feature maps. These feature maps capture different aspects of the input data. To improve the network's modeling capabilities, activation functions are applied after the convolutional layer, introducing non-linearity. The most commonly used activation functions in CNNs are the Rectified Linear Unit (ReLU) and Leaky ReLU. ReLU sets

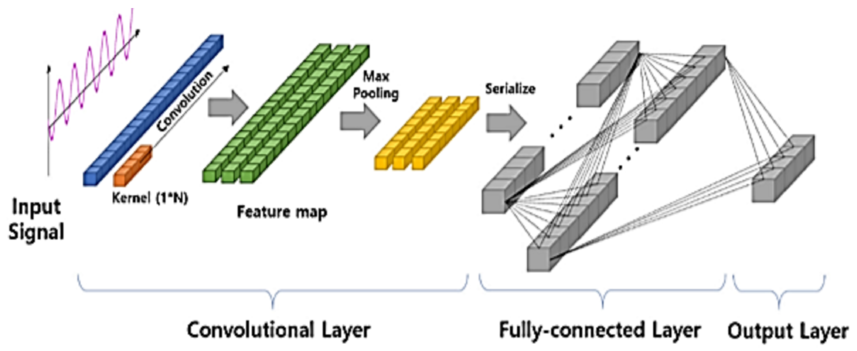


Fig. 2.10: General structure of 1D CNN for signal pattern recognition. It contains the Convolutional Layer, the Fully-connected Layer and the Output Layer. Source: Kim et al. [9]

negative values to zero and promotes sparse activation, while Leaky ReLU addresses the "dying ReLU" problem by introducing a small slope for negative inputs. Graphically, ReLU and Leaky ReLU can be represented as shown in the Figure 2.11. These activation functions enable CNNs to capture complex relationships and improve their performance in learning and extracting meaningful features from the input data.

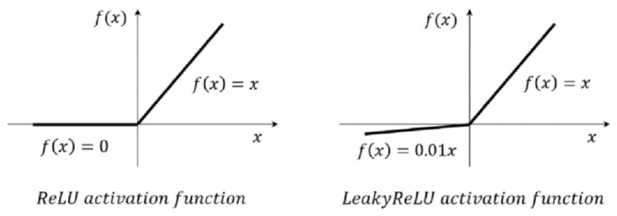


Fig. 2.11: ReLU activation function vs. LeakyReLU activation function (slope a = 0.01). Source: Li et al. [14]

- **Pooling layer:** As it is described by O'Shea and Nash [17], the pooling layer is positioned after the convolutional layer in a CNN architecture to reduce the spatial dimensions of the feature maps generated by the convolutional layer. It achieves this through two commonly used operations: Max Pooling and Average Pooling. Both operations divides the feature maps into non-overlapping rectangular or square regions, referred to as pooling windows. For each window, the maximum or the average value within that region is selected and passed on to the next layer, discarding the other values.This downsampling process provides several benefits, including spatial dimension reduction, improved efficiency, and increased robustness to local variations. However, the pooling layer also has limitations, such as information loss and reduction of the spatial resolution.
- **Fully Connected Layer**:
  The fully-connected layer, also known as the dense layer, is typically positioned towards the end of the neural network architecture, following the convolutional and pooling layers. In this layer, every neuron in the previous layer is connected to every neuron in the current layer. The connections between neurons possess weights that determine their significance. These weights are learned during the training process through

backpropagation and gradient descent. Fully-connected layer is the final layer aiming to detect patterns throughout the input from the convolutional layers and to generate the final classification or regression output.

2.3.3 Hyperparameters in CNN

Hyperparameters in Convolutional Neural Networks (CNNs) are pre-defined parameters that determine the architecture, behavior, and learning process of the network. They are not learned from the data but rather selected by the user through trial and error. Grid Search is a common approach used to find the optimal values for these hyperparameters. In order to use grid search, a range of values must be specified for each hyperparameter, and the network's performance must be carefully assessed for each and every possible combination. Analyzing and fine-tuning these hyperparameters is essential for optimizing the CNN's performance. The main hyperparameters used in a CNN are shown in the Figure ??.

Hyperparameters	**Description**
Kernel Size	Kernel size of the convolutional layer.
Kernel Count	Kernel count of the convolutional layer.
Activation Function	Neuron's activation function(ReLU, sigmoid).
Max Pooling	Down-sampling operation that selects the maximum value from a group of neighboring neurons.
Pooling Size	Refers to the dimension of the pooling window.
Layer Depth	Number of layers constituting entire network.
Padding	Process of adding extra pixels (usually filled with zeros) around the input image.
Zero-padding	Involves adding zeros to the border of an input image or feature map.
Batch Normalization	Normalizes the output of a layer by adjusting and scaling the activations within each mini-batch during training.
Loss Function	Function to calculate error.
Learning Rate	Amount of change in weight that is updated during learning.
Epoch	Number of learning iterations.
Batch Size	Group size to divide training data into several groups.
Dropout	Regularization technique that randomly deactivates a certain percentage of neurons during training.

Tab. 2.1: Hyperparameters of a general 1D CNN.

2.3.4 CNN Architectures

1. **AlexNet model**:
   AlexNet is a convolutional neural network (CNN) architecture that made significant contributions to the field of computer vision when it was introduced by Krizhevsky

et al. [10]. It was designed specifically for image classification tasks (2D Data) and achieved remarkable performance on the ImageNet dataset, significantly outperforming previous models. The architecture of AleNet is composed of eight layers, including five convolutional layers, followed by three fully connected layers, one of those is the output layer. Krizhevsky et al. [10] found out that using the relu as an activation function accelerated the speed of the training process by almost six times. Two Dropout layers were also used in order to provent the model from overfitting. The AlexNet architecture is depicted in the Figure 2.12.

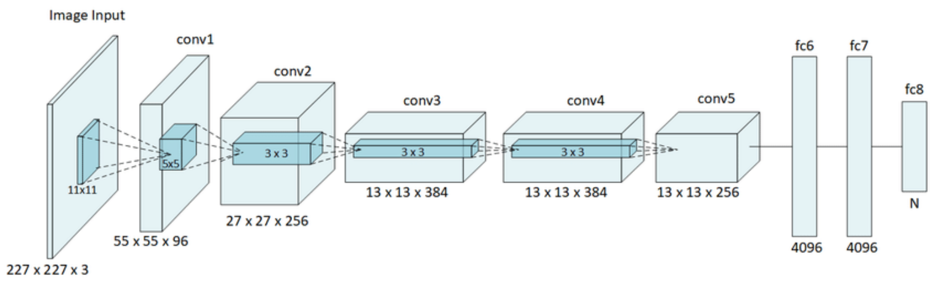


Fig. 2.12: Simplified illustration of the AlexNet architecture, created for the ImageNet challenge. Source: Hemmer et al. [8]

2. **ResNet model**:
   One of the key challenges in training deep CNNs is the problem of vanishing gradients, which can cause the network to have difficulty learning from the input data. This issue arises when the gradients computed during backpropagation become extremely small. The vanishing gradients problem limits the The network's ability to recognize complex patterns and relationships, which slows learning process. Moreover, when gradients vanish, the network is impossible to update the weights of the earlier layers, which can lead to slow convergence or even prevent the network from learning anything useful. This problem limits the depth of neural networks and slows down learning.

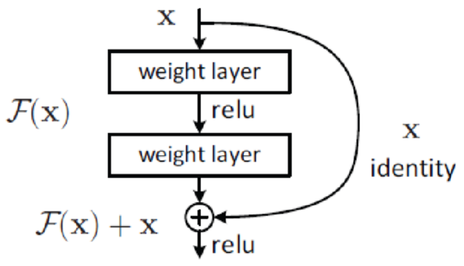
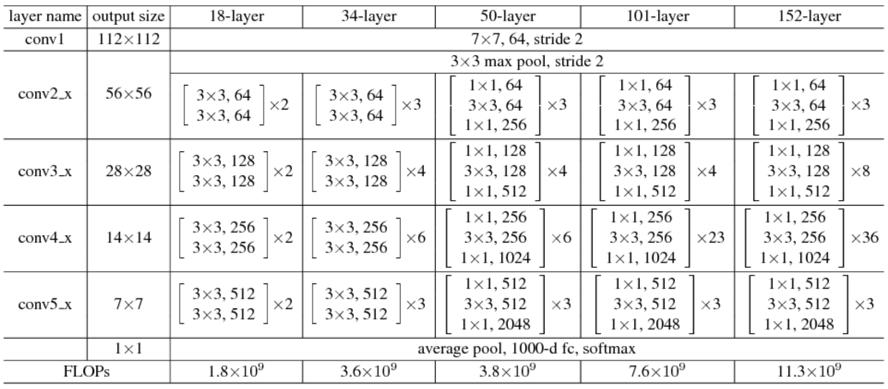


Fig. 2.13: Residual learning: a building block. Source: He et al. [7]

To address this problem, He et al. [7] proposed a deep neural network architecture called ResNet, which stands for "Residual Network". The ResNet model introduces a new building block known as "residual block", shown in Figure 2.13, that help the gradients to take a shortcut path and directly flow to earlier layers, allowing the gradient to flow more easily through the network and preventing them from vanishing. The

ResNet architecture consists of several layers of convolutional and pooling operations, followed by a global average pooling layer and a fully connected layer for classification. As it depicted in the Figure 2.14, there are five different architectures of the ResNet model depending on the number of the Convolutional layers. This number could be 18,34,50,101 and 152, according to the complexity of the use case. These architectures allows ResNet to effectively train deep networks by overcoming the problem caused by vanishing gradients.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig. 2.14: Different architectures (sizes) of the ResNet model. All of them work the same way as explained above. Source: He et al. [7]

3. **SincNet model**:
   The SincNet model is a deep learning architecture specifically designed for processing raw audio waveforms. It was introduced by Ravanelli and Bengio [18] and the key innovation of this model lies in its ability to learn filters directly from the audio signal, using the concept of band-pass filters. Traditional convolutional neural networks (CNNs) use predefined filters to extract features from the input data that learn all elements of each filter. In contrast, SincNet learns the parameters of band-pass filters that are applied to the raw audio waveform. This allows the model to capture frequency-specific information directly from the input signal.
   The SincNet architecture consists, as it is depicted in the Figure 2.15 , of three main components: the Sinc convolutional layer, the sub-sampling layer, and the fully connected layers. The Sinc convolutional layer is a key element of the SincNet architecture and distinguishes it from traditional CNN models. Its ability to learn band-pass filters directly from the input waveform allows SincNet to effectively capture frequency-specific information and model raw audio signals without the need for handcrafted features. More specifically, SincNet layers train "wavelets" for feature extraction by applying convolution operations on the input signal:

$$y[n] = x[n] * g[n, \theta] \tag{2.4}$$

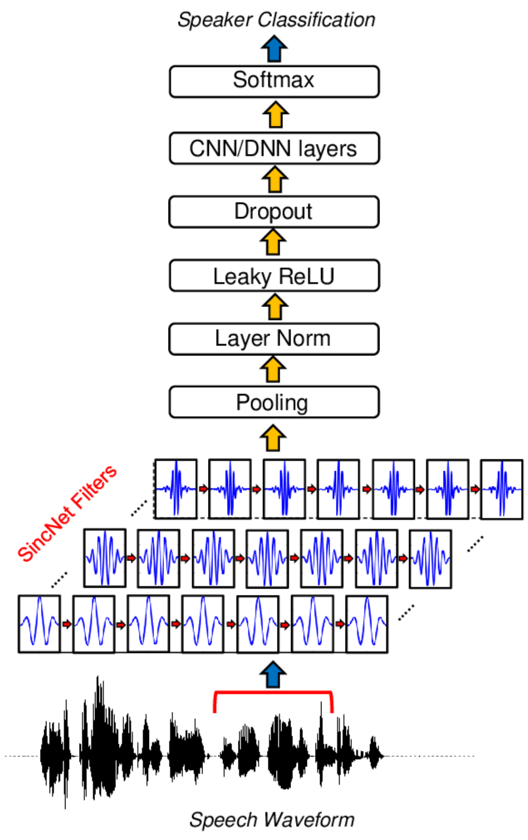


Fig. 2.15: The architecture of SincNet model. Source: Ravanelli and Bengio [18]

where n is represents the index of the probe and θ represents the parameters of the wavelets that are determined during the training process. The equation of the wavelet function g is:

$$g[n, f_1, f_2] = 2f_2 sinc(2\pi f_2 n) - 2f_1 sinc(2\pi f_1 n) \tag{2.5}$$

where sinc function

$$sinc(x) = \frac{sin(x)}{x} \tag{2.6}$$

f1 and *f2* are the cutoff frequencies determined by the SincNet layer during the training phase. The initialization of the pair of filters (*f1*,*f2*) in the SincNet architecture is based on the frequencies typically employed for computing Mel-frequency cepstral coefficients (MFCCs).
The sub-sampling layer in the SincNet architecture reduces the dimensionality of the feature maps, enabling the model to capture more robust and efficient representations. This reduction in dimensionality helps improve the efficiency and effectiveness of the network. The fully connected layers in SincNet then combine the extracted features and are responsible for performing classification or regression tasks. Additionally,

one advantage of SincNet is its ability to reduce the number of parameters in the first convolutional layer.

2.4 Evaluation Measures

The last subsection of this chapter presents the evaluation measures that will be used in the rest of this thesis for the evaluation of the models across all different datasets.

In the final subsection of this chapter, the evaluation measures to be used throughout the thesis for assessing the performance of the models across various datasets are presented. These evaluation measures will be used a framework for assessing and comparing the effectiveness of the models in different scenarios.

Cross-entropy loss is a loss function commonly used in machine learning and optimization. It measures the number of bits required to transform the output probability distribution of a model for a specific input into the actual distribution of that input. The cross-entropy loss serves as a measure of a model's performance. It penalizes the probability of each predicted class based on the distance from the actual expected value. A higher cross-entropy loss indicates greater divergence between the predictions and the ground truth. Ideally, the loss should be minimized and close to zero. This loss function is also referred to as log loss or logarithmic loss. Mathematically, the cross-entropy loss can be formally defined as follows:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log_2 \hat{y}_i + (1-y_i)\log_2(1-\hat{y}_i)\right] \tag{2.7}$$

where N is the total number of samples, y_i is the ground truth and $\hat{y}_i$ is the sample prediction.

Accuracy is a commonly used evaluation measure for classification tasks. It measures the proportion of correctly classified instances out of the total number of instances in a dataset. In other words, accuracy calculates the percentage of predictions that match the true labels. Accuracy is a straightforward measure that provides a general overview of the model's performance. It is easy to interpret and understand, making it a popular choice for evaluating classification models. It is defined as:

$$\text{Acc.} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{2.8}$$

However, it is important to note that accuracy alone may not provide a complete picture of a model's performance, especially in scenarios when the classes are imbalanced. In such cases, additional evaluation measures, such as precision, recall, or F1 score, may be necessary to assess the model's performance.

Recall is an evaluation metric used in binary classification tasks. It quantifies the ability of a classification model to correctly identify positive instances from the total number of actual positive instances in a dataset. It is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.9}$$

Precision is an evaluation metric used also in binary classification tasks. It assesses the ability of a classification model to accurately identify positive predictions from the total number of predicted positive instances. It is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.10}$$

where TP is the number of true positives, FP the number of false positives and FN the number of false negatives.

F1 score is an evaluation metric that combines precision and recall into a single score for binary classification tasks. It provides a balanced assessment of a model's performance by considering both the ability to correctly identify positive instances (recall) and the accuracy in classifying positive predictions (precision).

The F1 measure is calculated using the harmonic mean of precision and recall and it is defined as:

$$\text{F1} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})} \tag{2.11}$$

F1 score provides information about the performance of one of the categories. During the multi-label classification, metrics that calculate the performance of all labels were used like:

m-$\mathbf{F_1}$ (macro-F1) is the unweighted average of all F1 scores per label. It is defined as:

$$\mathrm{m{-}F_1} = \frac{\mathrm{F1_1} + \mathrm{F1_2} + \ldots + \mathrm{F1_C}}{C} \tag{2.12}$$

Where $\mathrm{F1_1}$ is the F1 score of the first label and C is the number of labels.

μ-$\mathbf{F_1}$ (micro-F1) is the global average F1 score, also known as the accuracy score, is calculated by summing the true positives (TP), false positives (FP), and false negatives (FN) for each label and then calculating the proportion of correctly classified samples out of all observations. Hence, it represents the accuracy of the classification model. It is defined as:

$$\mu{-}\mathrm{F_1} = \frac{\sum_{i=1}^{C} \mathrm{TP_i}}{\sum_{i=1}^{C} \mathrm{TP_i} + \frac{1}{2}\left(\sum_{i=1}^{C} \mathrm{FP_i} + \sum_{i=1}^{C} \mathrm{FN_i}\right)} \tag{2.13}$$

When the dataset is imbalanced, we need both of these metrics to understand how our models perform. μ-$\mathrm{F_1}$ assigns weight to the labels with more samples, while m-$\mathrm{F_1}$ treats all labels equally.

In cases where the dataset is imbalanced, it is important to consider both μ-$\mathrm{F_1}$ and m-$\mathrm{F_1}$ metrics to gain a better understanding of the model's performance.

Related Work 3

Zhao et al. [25] proposes the pair-wise self-consistency learning (PCL) method for deep-fake image detection. The proposed approach leverages the concept of source feature inconsistency within forged images to train Convolutional Neural Networks (ConvNets) for representation learning. Experimental results on seven datasets, including the Face-Forensic++ dataset, demonstrate that PCL outperforms the state of the art, improving average AUC from 96.45% to 98.05% in in-dataset evaluation and from 86.03% to 92.18% in cross-dataset evaluation.

[12] This PDF file introduces a new method to detect AI-generated fake face videos by analyzing eye blinking. The method involves several steps, including pre-processing video frames, locating face areas, extracting facial landmarks, aligning face regions, and extracting eye contours. Eye blinking detection is performed using a combination of Convolutional Neural Network (CNN) and Long-Short Term Memory (LSTM) network. Experimental results demonstrate that the Long-term Recurrent Convolutional Networks LRCN method achieves the highest performance (0.99) compared to CNN (0.98) and EAR (0.79) in accurately detecting eye blinking in videos. This approach provides a robust solution for detecting AI-generated fake face videos by focusing on the eye blinking behavior.

Research has demonstrated that biological signals, such as heartbeat, can be a reliable indicator of authentic videos. For instance, Ciftci et al. [5] proposed a Generative Adversarial Network (GAN) based model that leverages the "heartbeat" of deep fakes to detect their source. The proposed model comprises multiple detector networks that receive real videos as input. Following this, a registration layer is used to pair real and fake videos and extract facial regions of interest (ROI) and biological signals to create spatiotemporal windows called PPG cells. These cells contain multiple faces extracted using a face detector. Finally, a classification layer is employed to determine whether the video is fake or real. The authors evaluated their model using publicly available datasets and achieved an accuracy of 97.3% in detecting deep fakes, and the source model with 93.4% accuracy.

The method proposed by Yang et al. [24] is based on identifying errors introduced during the splicing of synthesized face regions into original images by estimating 3D head poses from the face images and developing a classification method. The authors observe a significant difference in the estimated head pose between Deep Fakes and authentic images. They use these differences to train a simple SVM-based classifier to differentiate original and

Deep Fakes. The proposed method achieves an Area Under ROC (AUROC) of 0.843 on the DARPA GAN Challenge dataset and 0.738 AUROC on the Deep Fake Dataset using head orientations and translations as features.

Li and Lyu [13] proposes a deep learning approach to detect DeepFake videos by targeting face warping artifacts. The authors observe that current DeepFake algorithms generate images with limited resolutions, which are then transformed to match the faces to be replaced in the source video. To capture these artifacts, they employ a dedicated convolutional neural network (CNN). The CNN is trained on a dataset comprising real and DeepFake videos, with variations in the DeepFake generation algorithms and techniques. The CNN is trained to classify videos as real or DeepFake based on the presence of face warping artifacts. This approach leverages the power of deep learning to effectively detect DeepFake videos by focusing on the characteristic artifacts introduced during the face transformation process. This method also achieves high accuracy on the DeepFake TIMIT dataset, with the best-performing CNN model achieving an accuracy of 98.5%.

Ciftci et al. [4] introduces a novel approach for detecting deep fakes by utilizing hidden biological signals in portrait videos as an implicit descriptor of authenticity. The proposed approach achieves a high accuracy of 99.39% in detecting fake content. The study demonstrates that involving biological signals dramatically improves deep fake detection compared to simple machine learning-based approaches. The spatial and temporal properties of biological signals are important, and these enable the network to perform significantly better than complex and deeper networks. The paper documents all experiments with the signal transformation, feature set, and SVM classification accuracy, trained on FF and DF datasets. The experiments show that biological signals are not consistently preserved in different synthetic facial parts. The results show a slight general tendency of creating false positives, but the probabilistic video classification, incorporating frequency domain, and appropriate segment durations eliminate these side effects. The paper concludes that the proposed system, named FakeCatcher, is a promising candidate for automatic monitoring systems, as it prioritizes minimizing false negatives while managing false positives within an acceptable range for secondary manual verification processes.

Demir and Ciftci [6] presents a novel approach to detect deep fakes using gaze tracking. The authors focus on identifying distinct eye and gaze features that differentiate deep fakes from real videos. They analyze and compare these signatures across different variations of real and fake videos, and use a deep neural network to classify any video as real or fake with high accuracy. The results of their experiments show that their approach achieves high accuracy in detecting deep fakes, with a mean accuracy of 90.6% and 90.3% in two 5-fold validations using different dataset splits, and 80.72% and 73.68% accuracies in cross-dataset evaluations between CDF and DFor datasets.

4 Proposed Methods

In this chapter, the proposed approaches for deepfake video detection are presented. The chapter begins with an overview of the adopted methodology, providing a clear understanding of the key components of the research. Following this, an in-depth analysis of the preprocessing techniques employed is presented. Finally, the chapter showcases the various architectures of the Convolutional Neural Network (CNN) models utilized.

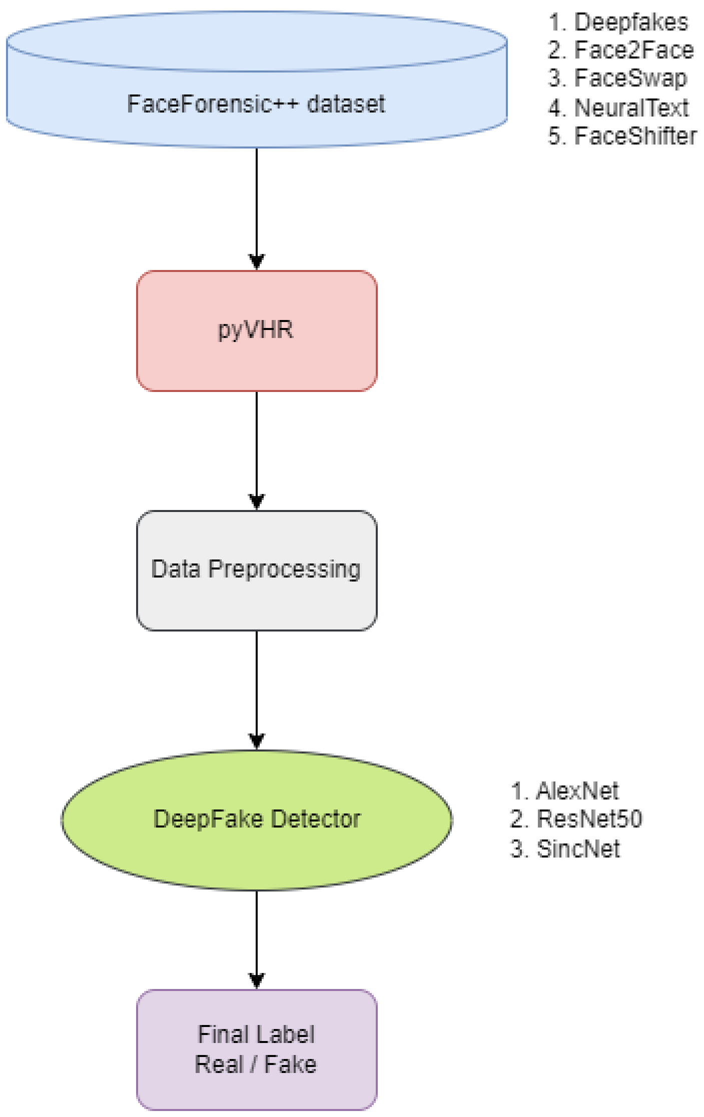


Fig. 4.1: Schema of proposed pipeline.

4.1 Approach Overview

Deepfake detection is a binary classification problem, distinguishing between REAL (label 0) and FAKE (label 1) labels. This type of classification is common in the context of deepfake detection, as the goal is to determine whether a given video is real or manipulated. Figure 4.1 illustrates the main parts of the recommended approach. The FaceForensics++ dataset is partitioned into five smaller datasets with 1000 real videos and their corresponding fakes, each altered by a deepfake algorithm. As a result, each of the five datasets contains 2000 videos. Each dataset is named after the specific altering technique employed. The core of the proposed method is in the extraction process of the BVP (blood volume pulse) signals from the input videos using the pyVHR framework. These produced signals undergo several preprocessing techniques before being fed into various state-of-the-art CNN models, which provide the predicted label for each specific input signal. All the models were implemented from scratch without the used of the transfer learning technique. As a result, all the weights in the CNN models were initialized randomly and updated specifically for the deepfake problem, ensuring a custom-tailored approach for better accuracy and reliable predictions.

The fundamental aspect of this thesis approach concerns individual signal-based classification. In other words, the focus lies on classifying each signal independently as opposed to aggregating signals from multiple frames. This unique approach allows the models to make decisions based on individual signal characteristics, improving the accuracy of deepfake video detection.

In this thesis, the approach to address the deepfake detection challenge is split into two distinct categories of CNN models. The first category is focused on 1D classification, where the individual BVP signals extracted from the videos are used as inputs to the CNN models. The second category involves CNN models for 2D classification, achieved by transforming the 1D BVP signals into 2D data through spectrograms.

Lastly, all models, including 1D and 2D CNNs, were trained using only the first deepfake dataset, known as the Deepfakes dataset. Due to time constraints and limitations in available resources, this initial training served as a foundational step to choose the optimal model. Moreover, the best-performing model was further trained on the other four datasets, generating additional results. Although the ideal scenario would have involved training all models on the entire dataset collection, practical restrictions required this step-by-step approach.

4.2 Data Pre-processing

Before training the models, several important preprocessing steps were applied to improve the quality of the data. These steps ensured that the models could better recognize significant patterns and features, leading to more accurate and effective deepfake detection.

Signals Creation

In the FaceForensics++ dataset, videos exhibit varying durations, resulting in a different number of extracted BVP signals for each video. To this point, the face of the subject is detected and a set of 100 patches is automatically tracked on it. This tracking involves a temporal window of 6 seconds moving through the entire video duration with a 1-second overlap. Within each temporal window, 100 patches are extracted, each containing 150 data points. For example, if a video have 8 frames within the temporal window, a total of 800 individual signals (8 * 100) are generated. These BVP signals from the videos are stored in nested lists, with each list varying in size based on the video duration.

Resampling

It is worth noting that the frame rates of the videos in FaceForensics++ dataset, where the BVP signals are estimated, vary. Although the majority of the videos are recorded at 25fps, some have different frame rates. Specifically, the FaceForensics++ dataset contains videos with the following frame rates: 15, 18, 24, 25, 29, 30, 50, and 60. Additionally, each BVP signal is 6 seconds, which leads to different lengths of the final BVP signal depending on the video's frame rate. The corresponding lengths of the BVP signals are as follows: 90, 108, 144, 150, 174, 180, 300, and 360. Since the FaceForensics++ dataset contains videos with varying frame rates, all the BVP signals were re-sampled to a common sampling rate of 25Hz. Example of this transformation is shown on Figure 4.2. This step enabled the signals to be standardized and made them suitable for the model training process. As a result, the length of each BVP signal became 150, regardless of the video's original frame rate.

Unfolding Process

Given the variation in video durations within the FaceForensics++ dataset, a challenge arises in handling all videos in the same way during the deepfake detection process. To address this issue, a solution was created by unfolding the signals specific to each individual video from the nested list structure and organize them into a new list, without considering their original sequence within the video. This method treated the BVP signals of each video independently, regardless of the video duration. This method effectively solved the

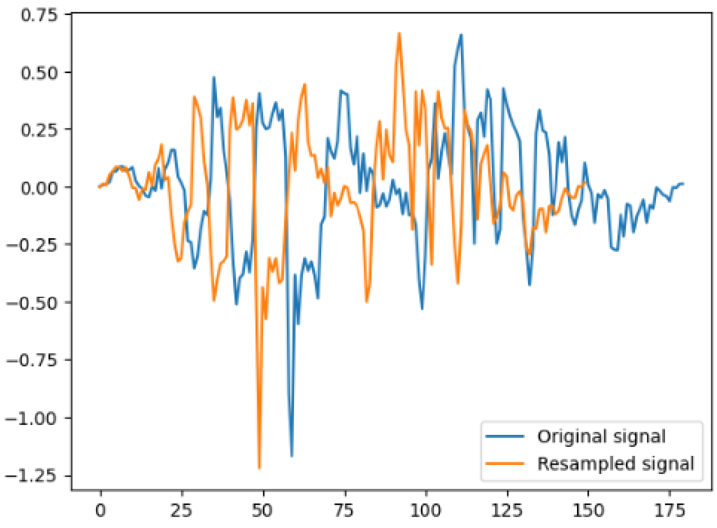


Fig. 4.2: Specific signal before (29Hz) and after (25Hz) the resample process.

problem of varying video lengths, making sure that all signals were ready to be fed into the various DeepFake Detectors(CNN models).

Missing Data Points

In certain instances, due to the movement of the area of interest (face) and the positioning of the patches during BVP signal extraction, some signals may contain missing data points, as depicted in Figure 4.3. To ensure the integrity of the data fed into the CNN models, any signals with missing data points were removed from the dataset. This step was taken to maintain the consistency of the BVP signals, as missing data points can interfere with the proper feeding of the CNN models.

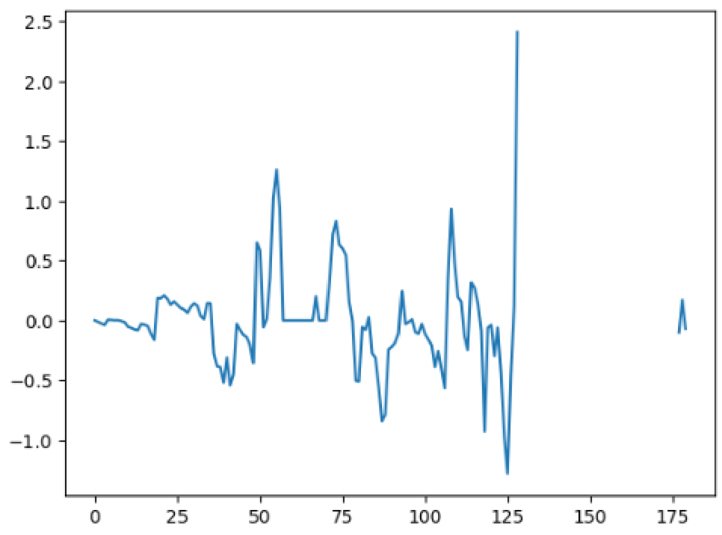
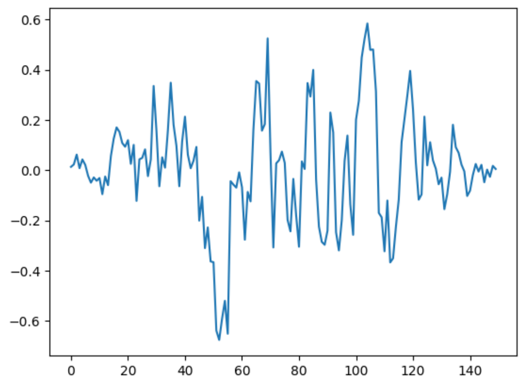
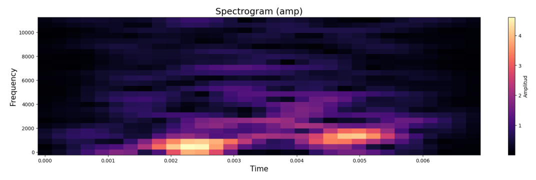


Fig. 4.3: BVP signal with missing data points.



(a) BVP Signal



(b) Spectogram

Fig. 4.4: Original BVP signal and the corresponding Spectogram.

Spectogram

In the second approach an important step is the conversion of one-dimensional BVP signals into spectograms (two-dimensional data). To find the optimal parameters for this conversion, various combinations of window size and hop length were experimented. Different combinations, such as 50ms, 10ms, 50ms, 5ms, 100ms, 10ms, and 100ms, 5ms, were tried to evaluate their impact on the resulting spectrograms. The optimal parameters that have the most informative spectograms are presented in Table 4.1. Figure **??** shows an example of a spectrogram derived from a BVP signal, along with the corresponding original BVP signal.

Model	**Domain**
Sampling rate	25Hz
Window length	50
Hop length	5

Tab. 4.1: Optimal parameters of spectrograms.

4.3 Models Architectures

The architecture overview of the CNN models used in this thesis falls into two distinct categories: one for processing 1D BVP signals and the other for handling 2D spectrograms. For the 1D BVP signals, three CNN models were employed: AlexNet, ResNet50, and SincNet.

For the 2D spectrogram inputs, two CNN models, AlexNet, and ResNet50, were adapted to handle the transformed data.

4.4 1D Approach

In the 1D model explanation section, an important point is the uniform input shape shared across all 1D CNN models, which is (150, 1). Additionally, the detailed configurations of each 1D CNN model, including AlexNet, ResNet50, and SincNet, will be explored.

AlexNet

In the initial analysis, the AlexNet CNN model served as the foundation for the first deepfake detection architecture. However, through a careful process of Hyperparameters tuning utilizing the KerasTuner optimization framework, significant changes were adopted in the model's architecture. By building upon the standard AlexNet, a more suitable and effective architecture for the deepfake detection challenge was created. The updated architecture maintained the core structure with five convolutional layers, but introduced changes in the number of filters employed. Additionally, an extra fully connected layer was added, changing the model's depth and allowing it to capture more robust patterns from the input BVP signals. To improve generalization and reduce over-fitting, a Batch Normalization layer was added after every Convolutional Layer, and a Dropout Layer with a rate of 0.5 was introduced after every Fully Connected Layer. This deeper CNN model demonstrated a notable improvement in performance, resulting in more accurate and reliable results for deepfake detection. The updated architecture can be shown in Table 4.2, providing a more clear view of the new model. Furthermore, Figure 7.1 depicts the whole architecture of the new model with the respective parameters.

ResNet50

The second model utilized in the thesis was the ResNet50, a CNN model containing the impressive number of 50 layers. Unlike the first model (AlexNet), ResNet50 was employed without using Hyperparameters tuning for its architecture. However, a crucial modification was made to the first layer of this CNN model to make it compatible with the 1D data input. Although the complete ResNet50 architecture is quite big to be presented in the thesis, Figure 4.5 provides an overview of its components. For the specific 1D deepfake detection problem, a small representation of the ResNet50 architecture is illustrated in Figure 7.2.

Layer	Num. Layers	Kernel Size	Activ. Function	Pool Size	Stride	Padding
Conv1D	96	11	ReLU	-	-	Same
MaxPool1D	-	-	-	1	1	Same
Conv1D	384	5	ReLU	-	-	Same
MaxPool1D	-	-	-	1	2	Same
Conv1D	256	11	ReLU	-	-	Same
MaxPool1D	-	-	-	3	1	Same
Conv1D	256	3	ReLU	-	-	Same
MaxPool1D	-	-	-	1	1	Same
Conv1D	96	3	ReLU	-	-	Same
MaxPool1D	-	-	-	1	1	Same
Dense	2048	-	ReLU	-	-	-
Dense	4096	-	ReLU	-	-	-
Dense	4096	-	ReLU	-	-	-
Dense	1	-	Sigmoid	-	-	-

Tab. 4.2: Updated AlexNet architecture after performing Hyperparameter Tunning.

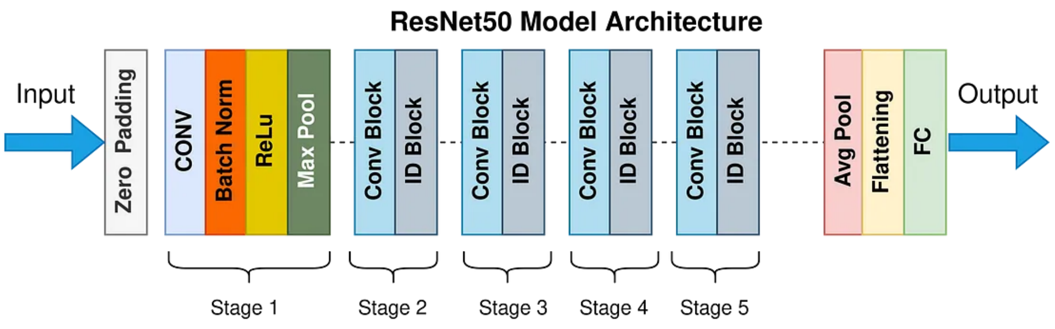


Fig. 4.5: ResNet50 architecture.

SincNet

The final model used for the 1D signal classification problem was the SincNet model. In this model, the Hyperparameter tuning process was employed using the KerasTuner optimizer framework, similar to the approach employed for the AlexNet model. The resulting architecture was carefully modified to better suit the challenges of deepfake detection. The new model comprises a total of five Convolutional Layers and five Fully Connected Layers. The first layer, SincConv1D, was specifically designed to improve the network's ability to capture patterns relating to signal-based problems while effectively filtering out noise and undesirable features. This layer reduces the number of parameters, enabling the extraction of highly selective filters and also reducing the computational demands of the model. Furthermore, the addition of more layers to the existing model improves its robustness, making it more capable of handling the complex task of deepfake detection. The new model architecture is presented in Table 4.3, offering a clearer understanding of the new model's structure. Additionally, Figure 7.3 shows the complete architecture of the updated model, along with its respective parameters, providing a clear representation of the changes made during the Hyperparameters tuning process.

Layer	Num. Layers	Kernel Size	Activ. Function	Pool Size	Stride	Padding
SincConv1D	64	1	LeakyReLU	-	1	Same
MaxPool1D	-	-	-	2	1	Same
Conv1D	64	3	LeakyReLU	-	-	Same
MaxPool1D	-	-	-	2	1	Same
Conv1D	128	3	LeakyReLU	-	-	Same
MaxPool1D	-	-	-	2	1	Same
Conv1D	128	3	LeakyReLU	-	-	Same
MaxPool1D	-	-	-	2	1	Same
Conv1D	64	3	LeakyReLU	-	-	Same
MaxPool1D	-	-	-	2	1	Same
Dense	256	-	LeakyReLU	-	-	-
Dense	512	-	LeakyReLU	-	-	-
Dense	512	-	LeakyReLU	-	-	-
Dense	256	-	LeakyReLU	-	-	-
Dense	1	-	Sigmoid	-	-	-

Tab. 4.3: Updated SincNet architecture after performing Hyperparameter Tunning.

4.5 2D Approach

In this section, the 2D models utilized in the deepfake detection process will be explained. An additional step was introduced before feeding the models, involving the conversion of the 1D data into 2D format through the spectrogram procedure. By performing this conversion, the input data shape transformed from (150, 1) to (26, 31, 1), as the data now represents pictures that capture frequency information. Finally, a detailed review of the AlexNet and ResNet50 models, adapted to process the transformed 2D data, will be presented.

AlexNet

For the first 2D model, the AlexNet model was employed. In this case, the model was utilized without the Hyperparameters tuning process. The architecture of the AlexNet model is depicted in Figure 2.12. However, due to the computational requirements of the model and the limitations of the available resources for the analysis, it was not possible to train the AlexNet model using the spectrograms as input data with the entire dataset.

<u>ResNet50</u>

In the 2D approach, the ResNet50 model was chosen as the second model for deepfake detection. Similar to the previous approach (1D ResNet50), the ResNet50 model was used without employing the KerasTuner framework. However, in this case, the model was adapted specifically for 2D data input. Figure 4.5 provides a general overview of the ResNet50 model, while Figure X shows a small part of the detailed representation of the specific architecture tailored for the 2D deepfake challenge.

For the final layer of every CNN architecture, the "Sigmoid" activation function was employed, specifically chosen to suit the binary classification problem at hand. This activation function is well-suited for tasks where the goal is to determine a binary outcome, such as distinguishing between real and fake videos in the context of deepfake detection. By using the "Sigmoid" activation, the models are able to provide output values between 0 and 1.

5 Experiments

In this chapter, the training process and performance evaluation of the previously introduced deepfake detection models are explored. Additionally, an extensive search to identify the best training hyperparameters were conducted, aiming to optimize each model's performance. Following the training process, the evaluation of each model's performance is presented. Key evaluation metrics, such as accuracy and F1-score, are analyzed to assess the models' effectiveness in distinguishing between real and deepfake videos. These metrics were chosen for their ability to provide a balanced view of the models' classification performance, considering both true positive and false positive rates.

In the first part of the training process, four different CNN models were trained. To identify the optimal training hyperparameters, the KerasTuner framework were employed, following by custom trial and error techniques. For the training optimizer, the Adam Optimizer were utilized, knowing for its efficiency in gradient-based optimization tasks. As the deepfake challenge is a binary classification problem, the BinaryCrossentropy loss function was selected, while the BinaryAccuracy metric is used as the evaluation metric during training. A batch size of 256 was applied across all models. It is important to note that the learning rate was different in the AlexNet 1D model compared to the other models, as it required a smaller value to achieve better convergence. Table 5.1 shows the optimal hyperparameters for all the trained models, representing the fine-tuned configurations that maximize their performance.

During the training process, an Early Stop callback with a patient equal to five were used, which helped to determine the ideal number of epochs for each model. More specifically, all models were stopped the training process after five epochs when there was no improvement in validation accuracy. This approach prevented overfitting, since identifies the most suitable stopping point, making sure the the models would achieve the best possible generalization. Furthermore, for the actual training process, 80% of the dataset was utilized. Within this 80%, 80% was used for the training phase, while the remaining 20% served as the validation set. The final 20% of the dataset was dedicated to the final model evaluation, providing an unbiased validation of the models' performance on unseen data.

During the evaluation of the models, key metrics such as accuracy and F1-score were very important. Additionally, the training time of each model was another crucial metric taken into account for the computational efficiency. All models were tested on the same test

Parameters	Values
Optimizer	Adam
Learning Rate	0.0001
Metric	BinaryAccuracy
Loss	BinaryCrossentropy
Batch Size	256

(a) AlexNet 1D

Parameters	Values
Optimizer	Adam
Learning Rate	0.001
Metric	BinaryAccuracy
Loss	BinaryCrossentropy
Batch Size	256

(b) ResNet50 1D

Parameters	Values
Optimizer	Adam
Learning Rate	0.001
Metric	BinaryAccuracy
Loss	BinaryCrossentropy
Batch Size	256

(c) SincNet 1D

Parameters	Values
Optimizer	Adam
Learning Rate	0.001
Metric	BinaryAccuracy
Loss	BinaryCrossentropy
Batch Size	256

(d) ResNet50 2D

Tab. 5.1: Optimal training Hyperparameters

set, providing a fair and consistent basis for comparison. The results obtained from this evaluation process are presented in Table 5.2, offering a better overview of each model's performance. The combined values of accuracy, F1-score, and training time allowed for a better understanding of the models' effectiveness and efficiency for the deepfake detection challenge.

It is important to be noted that during the training process, each signal is learned independently by the model. However, for the testing part, a video-level classification is required. To achieve this, all the individual signals from each video in the test set are gathered together. Then, a majority vote technique is employed to determine the final classification for the video. For example, if a testing video contains 1000 signals and the Deepfake Detector (model) classifies 700 signals as real and the remaining 300 as fake, the final classification for the video is determined as real, as the majority of signals are classified as real. This video-level classification approach ensures a better decision-making process based on the majority of individual signal predictions, providing a robust evaluation of the deepfake detection performance.

Models	**Accuracy**	**F1-score**	**Training time (seconds)**	**Epochs**
AlexNet 1D	74.2%	75.1%	72046	19
ResNet 1D	78.5%	78.9%	126083	21
SincNet	79.9%	81.5%	22255	28
ResNet 2D	70.1%	70.3%	1800136	25

Tab. 5.2: The results of every model on the FaceForensics++ dataset using only the deepfakes algorithm.

The results of the evaluation clearly indicate that the SincNet model outperforms the other models in terms of accuracy and F1-score, achieving 79.9% and 81.5%, respectively, making it the most efficient choice for deepfake detection. Furthermore, it achieved these impressive results in significantly less training time (22255 seconds) compared to the other models, and over just 28 epochs. The ResNet 1D model also performed well, with an accuracy of 78.5% and an F1-score of 78.9%, although it took a longer time to train (126083 seconds) over 21 epochs. The AlexNet 1D model showed reasonable results with an accuracy of 74.2% and an F1-score of 75.1%, taking 72046 seconds of training over 19 epochs. On the other hand, the ResNet 2D model exhibited the lowest accuracy and F1-score of 70.1% and 70.3%, respectively. Moreover, it had the longest training time (1,800,136 seconds) and was trained over 25 epochs. Based on these findings, the SincNet model was selected for further training and evaluation on the remaining four datasets (Face2Face, FaceShifter, FaceSwap, NeuralText). Four additional models were then created, using the SincNet model as the foundation for the deepfake detection task.

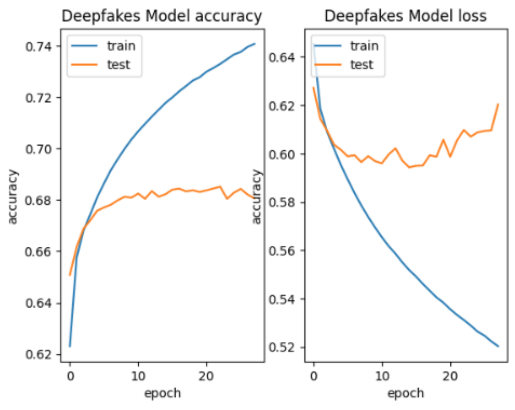
For the training process of the second part, the SincNet model was used following the same architecture and the same training Hyperparameters as in the first training part. Also in this part the Early Stop callback with a patient equal to five were used. The Figure 5.1, depict the accuracy and loss plots of the SincNet model for each dataset during the training process.

The plot shows the model's performance over a certain number of epochs, which are complete passes through the entire training dataset. The x-axis typically represents the number of epochs, while the y-axis represents the metric values. The number of epochs is different between every model due to the fact that the Early Stop callback was used. From the Figure 5.1 it is visible that the SincNet model trained in the FaceShifter dataset demonstrated the highest training accuracy and lowest training loss, making it the most effective model among those trained for Deep Fake detection.

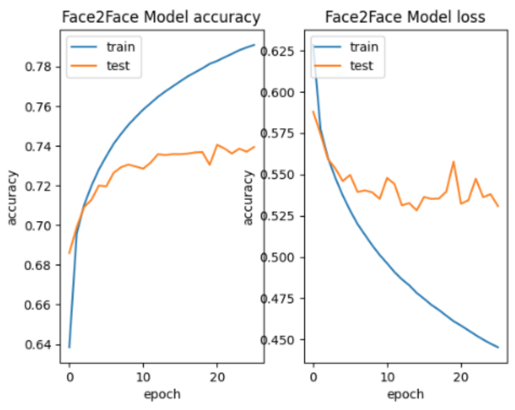
SincNet Models	Accuracy	F1-score	Training time (seconds)	Epochs
DeepFakes	79.9%	81.5%	22255	28
Face2Face	83.8%	82.2%	20488	26
FaceShifter	95.8%	95.9%	24879	31
FaceSwap	89.0%	90.2%	11513	17
NeuralText	62.3%	70.1%	12039	18

Tab. 5.3: The results of the SincNet model on all the different datasets within the FaceForensics++ dataset.

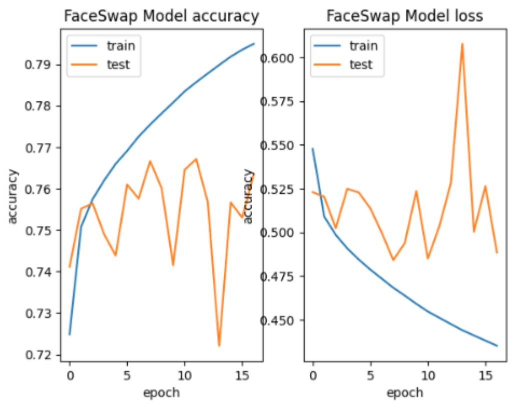
For the evaluation of the SincNet model on every dataset, the same metrics as before were used. The results obtained from this evaluation process are presented in Table 5.3, providing a gathered overview of the the model's performance on each dataset. The results demonstrate that SincNet model using the NeuralText dataset had the lowest accuracy, while using the FaceShifter dataset had the highest. It is worth noting that the FaceShifter deepfake algorithm is known for producing the most realistic appearances, making this



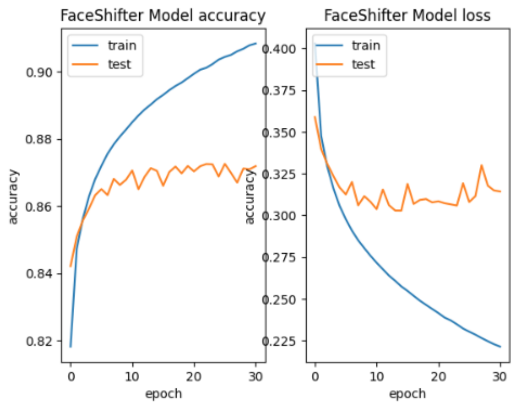
(a) Deepfakes



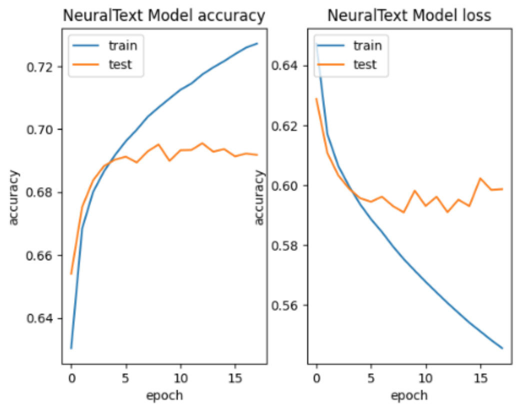
(b) Face2Face



(c) FaceSwap



(d) FaceShifter



(e) NeuralText

Fig. 5.1: Accuracy and loss training plots of the SincNet mode on every dataset.

approach particularly robust in distinguishing real and fake videos (95.8% accuracy). This happened due to the fact that the FaceShifter algorithm changes a lot the background information (the BVP signals dirived using the pyVHR) of the video in order to make the video as realistic as possible. On the opposite site, the NeuralText algorithm is not so robust and realistic as the other algorithms, so the background information didn't change a lot and that is why the accuracy using this dataset is low (62.3% accuracy). As a result, the proposed model has the ability to distinguish very accurately those deepfake videos that humans can't.

6 Conclusions

This thesis aimed to investigate the temporal aspect of deepfake detection, focusing on the physiological artifacts associated with the corruption of physiological signals, such as heart rate variability. The research questions explored revolved around the distinction between real and fake videos utilizing physiological signals, the performance difference between using raw BVP signals and spectrogram representations as input for 2D CNN models, and the predictive accuracy of the proposed model when dealing with more realistic videos manipulated by algorithms.

The research employed several state-of-the-art CNN models, including AlexNet, ResNet, and SincNet, along with various modifications and different data preprocessing techniques. The models were trained and tested on the FaceForensics+ dataset, which provided a comprehensive platform for evaluating the effectiveness of the proposed approaches.

The results obtained from the study provide valuable insights into the effectiveness of the proposed models for deepfake detection. The SincNet model demonstrated superior performance across all the different datasets within the FaceForensics++ dataset, achieving the highest accuracy and F1-score when trained on the FaceShifter dataset. This suggests that the model was highly effective in learning from this dataset and making accurate predictions. However, the performance of the model varied depending on the specific dataset used for training, indicating the importance of dataset selection in deepfake detection.

Furthermore, the results showed that the SincNet model outperformed other CNN models such as AlexNet and ResNet when trained on the same dataset. This highlights the effectiveness of the SincNet model in deepfake detection and its potential for further research and development in this field.

However, the study also revealed some limitations. The accuracy of the models was lower when trained on the NeuralText dataset, suggesting that the models may have difficulty learning from certain types of datasets.

In conclusion, the research questions posed in this study have been validated, and the findings can be summarized in three key points:

1. The physiological signals extracted from the videos prove to be sufficient for distinguishing real from forged videos. Depending on the training dataset, the accuracy varies between 62.3% and 95.8%.

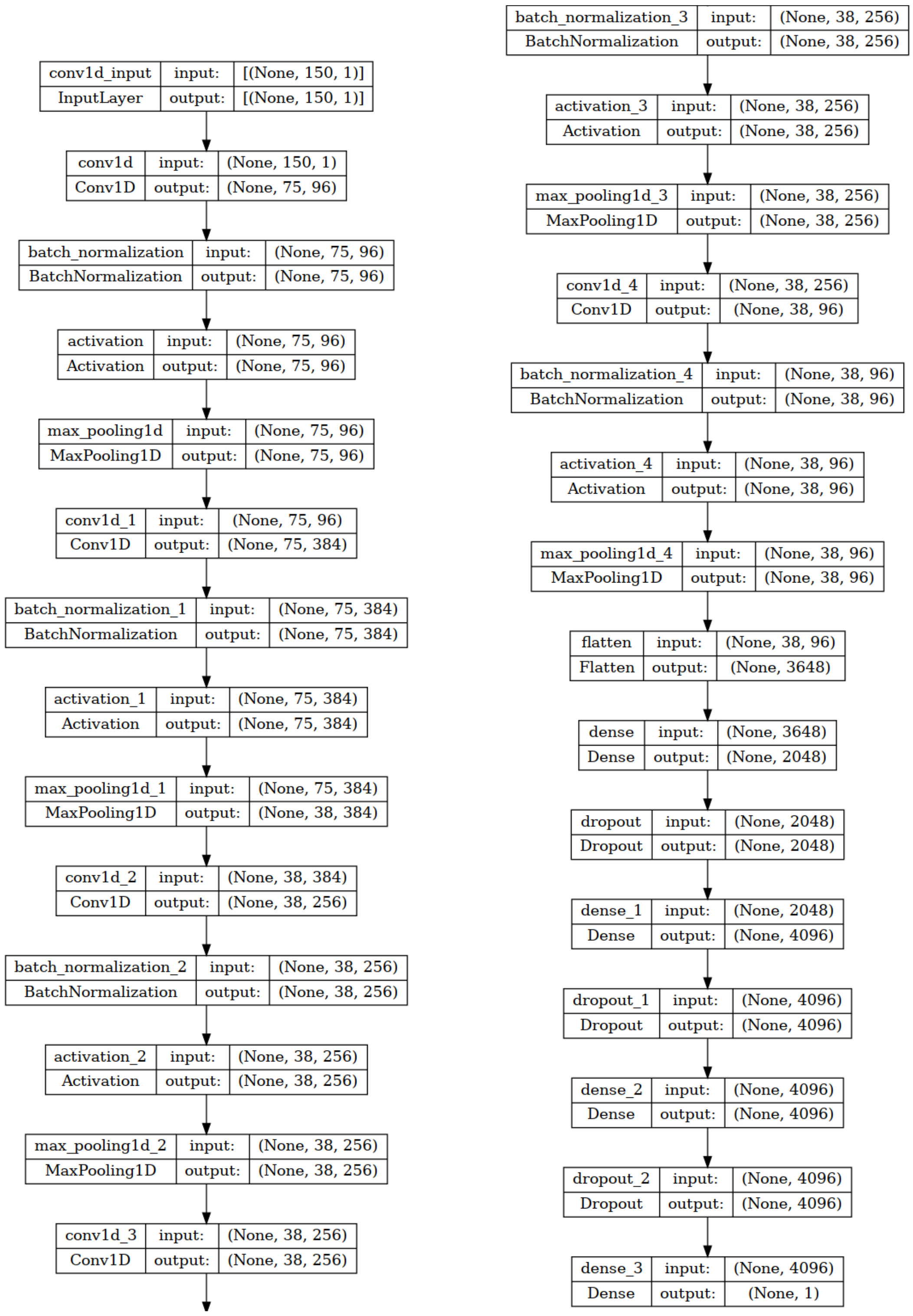
2. The testing results clearly demonstrate that all the CNN models trained with raw BVP signals outperform the model trained with spectrograms as input.

3. The proposed model (SincNet) performs significantly better when dealing with very realistic manipulated videos, such as FaceShifter, achieving an accuracy of 95.8%, compared to other less realistic manipulated videos, like NeuralText, where the accuracy was 62.3%.

Based on the results of this thesis and the ongoing progress in deepfake techniques, there are several potential areas for future work to improve deepfake detection and overcome limitations:

- **Expand Dataset Coverage:** One potential future direction is to train all the proposed CNN models on every dataset within the FaceForensics++ dataset.
- **Multi-Dataset Evaluation:** Another possibility is to train the SincNet model on four datasets within the FaceForensics++ dataset and test it on the fifth dataset, assessing its ability to adapt and perform well on previously unseen deepfake variations. With this approach five more models will be created.
- **Real-Time Deployment:** Developing an efficient real-time deployment strategy for the deepfake detection models.

7 Appendix



conv1d_input | InputLayer | input: [(None, 150, 1)] | output: [(None, 150, 1)]
conv1d | Conv1D | input: (None, 150, 1) | output: (None, 75, 96)
batch_normalization | BatchNormalization | input: (None, 75, 96) | output: (None, 75, 96)
activation | Activation | input: (None, 75, 96) | output: (None, 75, 96)
max_pooling1d | MaxPooling1D | input: (None, 75, 96) | output: (None, 75, 96)
conv1d_1 | Conv1D | input: (None, 75, 96) | output: (None, 75, 384)
batch_normalization_1 | BatchNormalization | input: (None, 75, 384) | output: (None, 75, 384)
activation_1 | Activation | input: (None, 75, 384) | output: (None, 75, 384)
max_pooling1d_1 | MaxPooling1D | input: (None, 75, 384) | output: (None, 38, 384)
conv1d_2 | Conv1D | input: (None, 38, 384) | output: (None, 38, 256)
batch_normalization_2 | BatchNormalization | input: (None, 38, 256) | output: (None, 38, 256)
activation_2 | Activation | input: (None, 38, 256) | output: (None, 38, 256)
max_pooling1d_2 | MaxPooling1D | input: (None, 38, 256) | output: (None, 38, 256)
conv1d_3 | Conv1D | input: (None, 38, 256) | output: (None, 38, 256)
batch_normalization_3 | BatchNormalization | input: (None, 38, 256) | output: (None, 38, 256)
activation_3 | Activation | input: (None, 38, 256) | output: (None, 38, 256)
max_pooling1d_3 | MaxPooling1D | input: (None, 38, 256) | output: (None, 38, 256)
conv1d_4 | Conv1D | input: (None, 38, 256) | output: (None, 38, 96)
batch_normalization_4 | BatchNormalization | input: (None, 38, 96) | output: (None, 38, 96)
activation_4 | Activation | input: (None, 38, 96) | output: (None, 38, 96)
max_pooling1d_4 | MaxPooling1D | input: (None, 38, 96) | output: (None, 38, 96)
flatten | Flatten | input: (None, 38, 96) | output: (None, 3648)
dense | Dense | input: (None, 3648) | output: (None, 2048)
dropout | Dropout | input: (None, 2048) | output: (None, 2048)
dense_1 | Dense | input: (None, 2048) | output: (None, 4096)
dropout_1 | Dropout | input: (None, 4096) | output: (None, 4096)
dense_2 | Dense | input: (None, 4096) | output: (None, 4096)
dropout_2 | Dropout | input: (None, 4096) | output: (None, 4096)
dense_3 | Dense | input: (None, 4096) | output: (None, 1)

Fig. 7.1: Custom AlexNet architecture

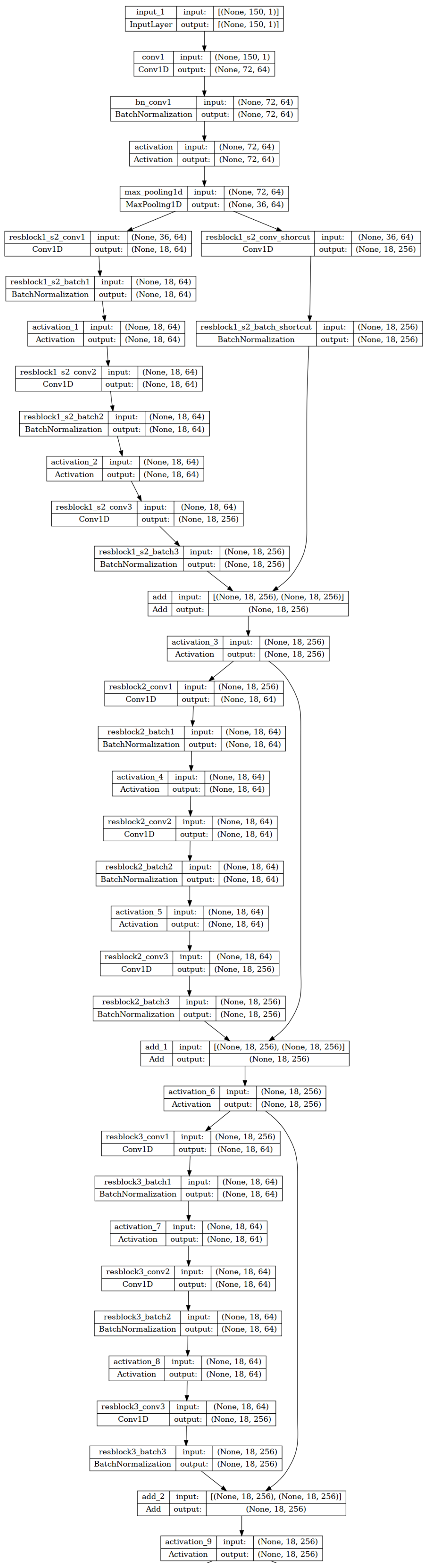

input_1 | input: | [(None, 150, 1)]
InputLayer | output: | [(None, 150, 1)]
conv1 | input: | (None, 150, 1)
Conv1D | output: | (None, 72, 64)
bn_conv1 | input: | (None, 72, 64)
BatchNormalization | output: | (None, 72, 64)
activation | input: | (None, 72, 64)
Activation | output: | (None, 72, 64)
max_pooling1d | input: | (None, 72, 64)
MaxPooling1D | output: | (None, 36, 64)
resblock1_s2_conv1 | input: | (None, 36, 64)
Conv1D | output: | (None, 18, 64)
resblock1_s2_conv_shorcut | input: | (None, 36, 64)
Conv1D | output: | (None, 18, 256)
resblock1_s2_batch1 | input: | (None, 18, 64)
BatchNormalization | output: | (None, 18, 64)
activation_1 | input: | (None, 18, 64)
Activation | output: | (None, 18, 64)
resblock1_s2_batch_shortcut | input: | (None, 18, 256)
BatchNormalization | output: | (None, 18, 256)
resblock1_s2_conv2 | input: | (None, 18, 64)
Conv1D | output: | (None, 18, 64)
resblock1_s2_batch2 | input: | (None, 18, 64)
BatchNormalization | output: | (None, 18, 64)
activation_2 | input: | (None, 18, 64)
Activation | output: | (None, 18, 64)
resblock1_s2_conv3 | input: | (None, 18, 64)
Conv1D | output: | (None, 18, 256)
resblock1_s2_batch3 | input: | (None, 18, 256)
BatchNormalization | output: | (None, 18, 256)
add | input: | [(None, 18, 256), (None, 18, 256)]
Add | output: | (None, 18, 256)
activation_3 | input: | (None, 18, 256)
Activation | output: | (None, 18, 256)
resblock2_conv1 | input: | (None, 18, 256)
Conv1D | output: | (None, 18, 64)
resblock2_batch1 | input: | (None, 18, 64)
BatchNormalization | output: | (None, 18, 64)
activation_4 | input: | (None, 18, 64)
Activation | output: | (None, 18, 64)
resblock2_conv2 | input: | (None, 18, 64)
Conv1D | output: | (None, 18, 64)
resblock2_batch2 | input: | (None, 18, 64)
BatchNormalization | output: | (None, 18, 64)
activation_5 | input: | (None, 18, 64)
Activation | output: | (None, 18, 64)
resblock2_conv3 | input: | (None, 18, 64)
Conv1D | output: | (None, 18, 256)
resblock2_batch3 | input: | (None, 18, 256)
BatchNormalization | output: | (None, 18, 256)
add_1 | input: | [(None, 18, 256), (None, 18, 256)]
Add | output: | (None, 18, 256)
activation_6 | input: | (None, 18, 256)
Activation | output: | (None, 18, 256)
resblock3_conv1 | input: | (None, 18, 256)
Conv1D | output: | (None, 18, 64)
resblock3_batch1 | input: | (None, 18, 64)
BatchNormalization | output: | (None, 18, 64)
activation_7 | input: | (None, 18, 64)
Activation | output: | (None, 18, 64)
resblock3_conv2 | input: | (None, 18, 64)
Conv1D | output: | (None, 18, 64)
resblock3_batch2 | input: | (None, 18, 64)
BatchNormalization | output: | (None, 18, 64)
activation_8 | input: | (None, 18, 64)
Activation | output: | (None, 18, 64)
resblock3_conv3 | input: | (None, 18, 64)
Conv1D | output: | (None, 18, 256)
resblock3_batch3 | input: | (None, 18, 256)
BatchNormalization | output: | (None, 18, 256)
add_2 | input: | [(None, 18, 256), (None, 18, 256)]
Add | output: | (None, 18, 256)
activation_9 | input: | (None, 18, 256)
Activation | output: | (None, 18, 256)

Fig. 7.2: Small part of the 1D ResNet50 architecture.

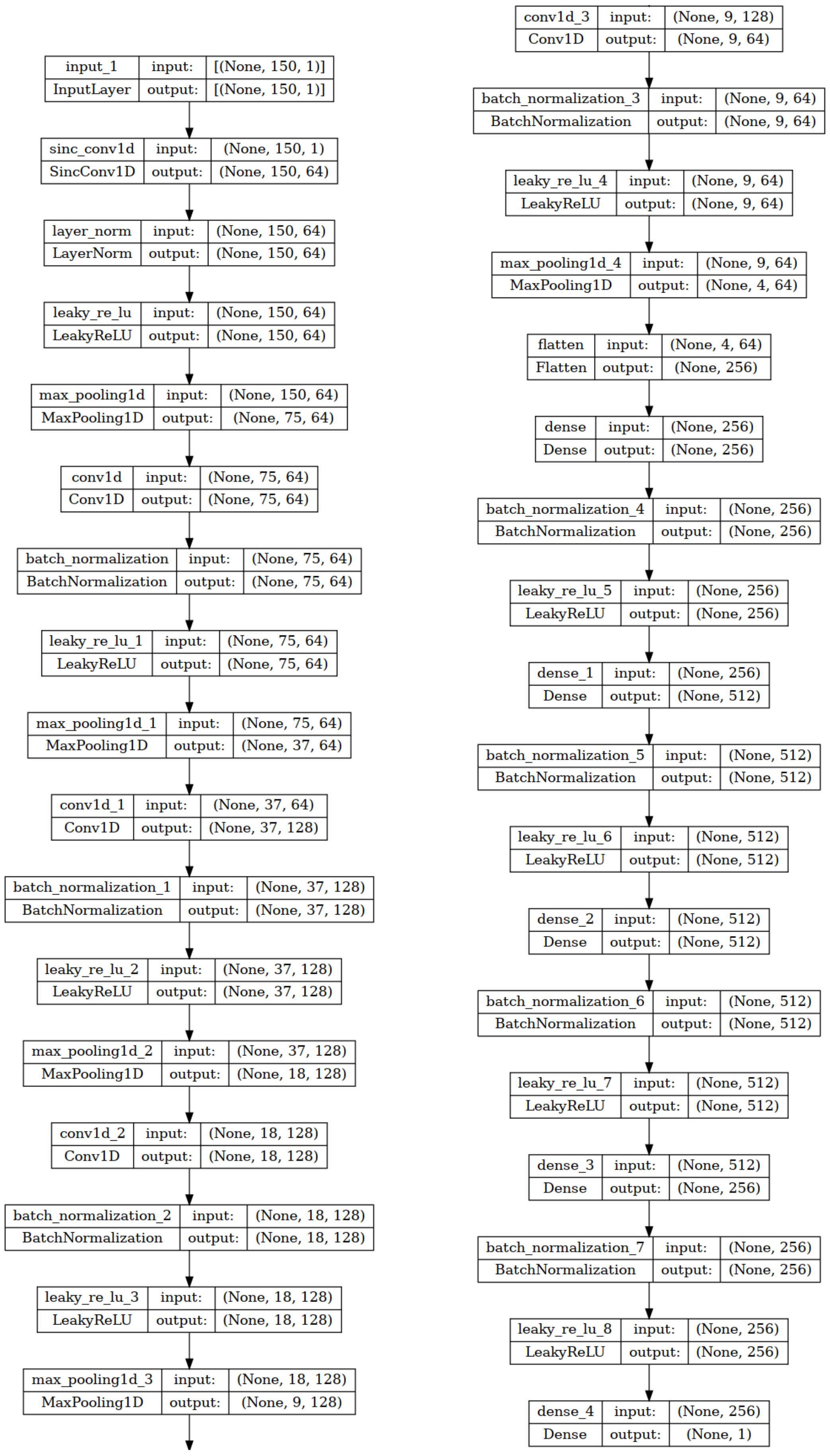

input_1 | input: | [(None, 150, 1)]
InputLayer | output: | [(None, 150, 1)]
sinc_conv1d | input: | (None, 150, 1)
SincConv1D | output: | (None, 150, 64)
layer_norm | input: | (None, 150, 64)
LayerNorm | output: | (None, 150, 64)
leaky_re_lu | input: | (None, 150, 64)
LeakyReLU | output: | (None, 150, 64)
max_pooling1d | input: | (None, 150, 64)
MaxPooling1D | output: | (None, 75, 64)
conv1d | input: | (None, 75, 64)
Conv1D | output: | (None, 75, 64)
batch_normalization | input: | (None, 75, 64)
BatchNormalization | output: | (None, 75, 64)
leaky_re_lu_1 | input: | (None, 75, 64)
LeakyReLU | output: | (None, 75, 64)
max_pooling1d_1 | input: | (None, 75, 64)
MaxPooling1D | output: | (None, 37, 64)
conv1d_1 | input: | (None, 37, 64)
Conv1D | output: | (None, 37, 128)
batch_normalization_1 | input: | (None, 37, 128)
BatchNormalization | output: | (None, 37, 128)
leaky_re_lu_2 | input: | (None, 37, 128)
LeakyReLU | output: | (None, 37, 128)
max_pooling1d_2 | input: | (None, 37, 128)
MaxPooling1D | output: | (None, 18, 128)
conv1d_2 | input: | (None, 18, 128)
Conv1D | output: | (None, 18, 128)
batch_normalization_2 | input: | (None, 18, 128)
BatchNormalization | output: | (None, 18, 128)
leaky_re_lu_3 | input: | (None, 18, 128)
LeakyReLU | output: | (None, 18, 128)
max_pooling1d_3 | input: | (None, 18, 128)
MaxPooling1D | output: | (None, 9, 128)
conv1d_3 | input: | (None, 9, 128)
Conv1D | output: | (None, 9, 64)
batch_normalization_3 | input: | (None, 9, 64)
BatchNormalization | output: | (None, 9, 64)
leaky_re_lu_4 | input: | (None, 9, 64)
LeakyReLU | output: | (None, 9, 64)
max_pooling1d_4 | input: | (None, 9, 64)
MaxPooling1D | output: | (None, 4, 64)
flatten | input: | (None, 4, 64)
Flatten | output: | (None, 256)
dense | input: | (None, 256)
Dense | output: | (None, 256)
batch_normalization_4 | input: | (None, 256)
BatchNormalization | output: | (None, 256)
leaky_re_lu_5 | input: | (None, 256)
LeakyReLU | output: | (None, 256)
dense_1 | input: | (None, 256)
Dense | output: | (None, 512)
batch_normalization_5 | input: | (None, 512)
BatchNormalization | output: | (None, 512)
leaky_re_lu_6 | input: | (None, 512)
LeakyReLU | output: | (None, 512)
dense_2 | input: | (None, 512)
Dense | output: | (None, 512)
batch_normalization_6 | input: | (None, 512)
BatchNormalization | output: | (None, 512)
leaky_re_lu_7 | input: | (None, 512)
LeakyReLU | output: | (None, 512)
dense_3 | input: | (None, 512)
Dense | output: | (None, 256)
batch_normalization_7 | input: | (None, 256)
BatchNormalization | output: | (None, 256)
leaky_re_lu_8 | input: | (None, 256)
LeakyReLU | output: | (None, 256)
dense_4 | input: | (None, 256)
Dense | output: | (None, 1)

Fig. 7.3: Custom SincNet architecture

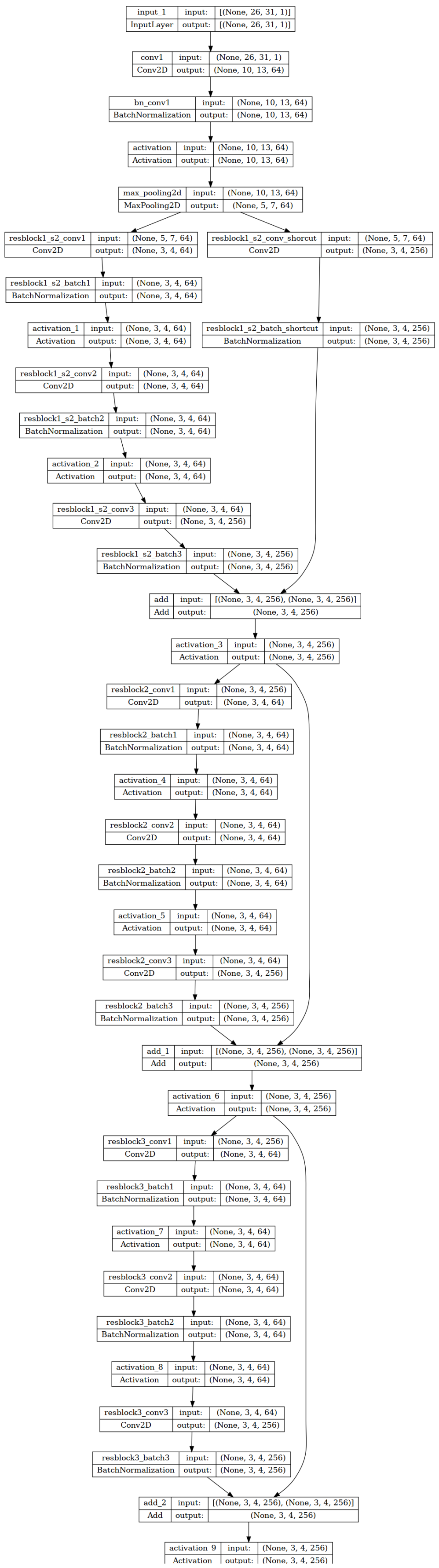

input_1 | InputLayer | input: [(None, 26, 31, 1)] | output: [(None, 26, 31, 1)]
conv1 | Conv2D | input: (None, 26, 31, 1) | output: (None, 10, 13, 64)
bn_conv1 | BatchNormalization | input: (None, 10, 13, 64) | output: (None, 10, 13, 64)
activation | Activation | input: (None, 10, 13, 64) | output: (None, 10, 13, 64)
max_pooling2d | MaxPooling2D | input: (None, 10, 13, 64) | output: (None, 5, 7, 64)
resblock1_s2_conv1 | Conv2D | input: (None, 5, 7, 64) | output: (None, 3, 4, 64)
resblock1_s2_conv_shorcut | Conv2D | input: (None, 5, 7, 64) | output: (None, 3, 4, 256)
resblock1_s2_batch1 | BatchNormalization | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
activation_1 | Activation | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
resblock1_s2_batch_shortcut | BatchNormalization | input: (None, 3, 4, 256) | output: (None, 3, 4, 256)
resblock1_s2_conv2 | Conv2D | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
resblock1_s2_batch2 | BatchNormalization | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
activation_2 | Activation | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
resblock1_s2_conv3 | Conv2D | input: (None, 3, 4, 64) | output: (None, 3, 4, 256)
resblock1_s2_batch3 | BatchNormalization | input: (None, 3, 4, 256) | output: (None, 3, 4, 256)
add | Add | input: [(None, 3, 4, 256), (None, 3, 4, 256)] | output: (None, 3, 4, 256)
activation_3 | Activation | input: (None, 3, 4, 256) | output: (None, 3, 4, 256)
resblock2_conv1 | Conv2D | input: (None, 3, 4, 256) | output: (None, 3, 4, 64)
resblock2_batch1 | BatchNormalization | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
activation_4 | Activation | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
resblock2_conv2 | Conv2D | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
resblock2_batch2 | BatchNormalization | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
activation_5 | Activation | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
resblock2_conv3 | Conv2D | input: (None, 3, 4, 64) | output: (None, 3, 4, 256)
resblock2_batch3 | BatchNormalization | input: (None, 3, 4, 256) | output: (None, 3, 4, 256)
add_1 | Add | input: [(None, 3, 4, 256), (None, 3, 4, 256)] | output: (None, 3, 4, 256)
activation_6 | Activation | input: (None, 3, 4, 256) | output: (None, 3, 4, 256)
resblock3_conv1 | Conv2D | input: (None, 3, 4, 256) | output: (None, 3, 4, 64)
resblock3_batch1 | BatchNormalization | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
activation_7 | Activation | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
resblock3_conv2 | Conv2D | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
resblock3_batch2 | BatchNormalization | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
activation_8 | Activation | input: (None, 3, 4, 64) | output: (None, 3, 4, 64)
resblock3_conv3 | Conv2D | input: (None, 3, 4, 64) | output: (None, 3, 4, 256)
resblock3_batch3 | BatchNormalization | input: (None, 3, 4, 256) | output: (None, 3, 4, 256)
add_2 | Add | input: [(None, 3, 4, 256), (None, 3, 4, 256)] | output: (None, 3, 4, 256)
activation_9 | Activation | input: (None, 3, 4, 256) | output: (None, 3, 4, 256)

Fig. 7.4: Small part of the 2D ResNet50 architecture.

Bibliography

[1]John Allen. “Photoplethysmography and its application in clinical physiological measurement”. In: *Physiological Measurement* 28 (Apr. 2007), R1–39.

[2]Giuseppe Boccignone, Donatello Conte, Vittorio Cuculo, et al. “pyVHR: a Python framework for remote photoplethysmography”. In: *PeerJ Comput. Sci.* 8 (2022), e929.

[3]Lamia Bouafif, Kais Ouni, and Noureddine Ellouze. “Implementation of a Speech Coding Strategy for Auditory Implants”. In: *Int. Arab J. Inf. Technol.* 7 (Dec. 2010), pp. 388–394.

[4]Umur Aybars Ciftci, Ilke Demir, and Lijun Yin. “FakeCatcher: Detection of Synthetic Portrait Videos using Biological Signals”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1.

[5]Umur Aybars Ciftci, Ilke Demir, and Lijun Yin. *How Do the Hearts of Deep Fakes Beat? Deep Fake Source Detection via Interpreting Residuals with Biological Signals.* 2020. arXiv: `2008.11363 [cs.CV]`.

[6]Ilke Demir and Umur Aybars Ciftci. “Where Do Deep Fakes Look? Synthetic Face Detection via Gaze Tracking”. In: *ACM Symposium on Eye Tracking Research and Applications.* ACM, May 2021.

[7]Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition.* 2015. arXiv: `1512.03385 [cs.CV]`.

[8]Martin Hemmer, Huynh Van Khang, Kjell G. Robbersmyr, Tor I. Waag, and Thomas J. J. Meyer. “Fault Classification of Axial and Radial Roller Bearings Using Transfer Learning through a Pretrained Convolutional Neural Network”. In: *Designs* 2.4 (2018).

[9]Seong-Hoon Kim, Zong Woo Geem, and Gi-Tae Han. “Hyperparameter Optimization Method Based on Harmony Search Algorithm to Improve Performance of 1D CNN Human Respiration Pattern Recognition System”. In: *Sensors* 20.13 (2020).

[10]Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems.* Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[11]Lingzhi Li, Jianmin Bao, Hao Yang, Dong Chen, and Fang Wen. “Advancing high fidelity identity swapping for forgery detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5074–5083.

[12]Yuezun Li, Ming-Ching Chang, and Siwei Lyu. *In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking*. 2018. arXiv: `1806.02877 [cs.CV]`.

[13]Yuezun Li and Siwei Lyu. *Exposing DeepFake Videos By Detecting Face Warping Artifacts*. 2019. arXiv: `1811.00656 [cs.CV]`.

[14]Z. Li, W. T. Nash, S. P. O Brien, et al. *cardiGAN: A Generative Adversarial Network Model for Design and Discovery of Multi Principal Element Alloys*. 2022. arXiv: `2202.00966 [cond-mat.mtrl-sci]`.

[15]Gabor Manhertz and Akos Bereczky. “STFT spectrogram based hybrid evaluation method for rotating machine transient vibration analysis”. In: *Mechanical Systems and Signal Processing* 154 (2021), p. 107583.

[16]Yisroel Mirsky and Wenke Lee. “The Creation and Detection of Deepfakes: A Survey”. In: *CoRR* abs/2004.11138 (2020). arXiv: `2004.11138`.

[17]Keiron O’Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: `1511.08458 [cs.NE]`.

[18]Mirco Ravanelli and Yoshua Bengio. *Speaker Recognition from Raw Waveform with SincNet*. 2019. arXiv: `1808.00158 [eess.AS]`.

[19]Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, et al. *FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces*. 2018. arXiv: `1803.09179 [cs.CV]`.

[20]J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. Nießner. “Face2Face: Real-time Face Capture and Reenactment of RGB Videos”. In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2016.

[21]Justus Thies, Michael Zollhöfer, and Matthias Nießner. *Deferred Neural Rendering: Image Synthesis using Neural Textures*. 2019. arXiv: `1904.12356 [cs.CV]`.

[22]İbrahim TOPAL. “ESTIMATION WITH ARTIFICIAL NEURAL NETWORK ON ELECTRONIC WORD OF MOUTH: OPINION SEARCHING”. In: *Yönetim Bilimleri Dergisi* 18.35 (2020).

[23]Wenjin Wang, Albertus C. den Brinker, Sander Stuijk, and Gerard de Haan. “Algorithmic Principles of Remote PPG”. In: *IEEE Transactions on Biomedical Engineering* 64.7 (2017), pp. 1479–1491.

[24]Xin Yang, Yuezun Li, and Siwei Lyu. *Exposing Deep Fakes Using Inconsistent Head Poses*. 2018. arXiv: `1811.00661 [cs.CV]`.

[25]Tianchen Zhao, Xiang Xu, Mingze Xu, et al. *Learning Self-Consistency for Deepfake Detection*. 2021. arXiv: `2012.09311 [cs.CV]`.

List of Acronyms

CNN	Convolutional Neural Network
BVP	Blood Volume Pulse
GAN	Generative Adversarial Network
RGB	Red Green Blue
HEAR-Net	Heuristic Error Acknowledging Network
PPG	Photoplethysmography
LED	Light-Emitting Diode
ECG	Electrocardiography
rPPG	remote Photoplethysmography
BSS	Blind Source Separation
CHROM	Chrominance-base
HRV	Heart Rate Variability
ROI	Region Of Interest
DL	Deep Learning
BPM	Beats Per Minute
PSD	Power Spectral Density

POS	Plane Orthogonal Skin
PCA	Principal Component Analysis
DFT	Discete time Fourier Transformation
STFT	Short-Time Fourier Trasformation
ANN	Artificial Neural Network
ReLU	Rectified Linear Unit
MFCC	Mel-Frequency Cepstral Coefficient
PCL	pair-wise self-Consistency Learning
ConvNet	Convolutional Network
AI	Artificial Intelligence
LSTM	Long-Sort Term Memory
LRCN	Long-sort Reccurent Convolutional Network
ROC	Receiver Operating Characteristic
AUROC	Area Under Receiver Operating Characteristic
SVM	Support Vector Machine
FF	FaceForensics
Hz	Hertz

List of Figures

List of Tables