HAW HAMBURG

BACHELOR THESIS

Deepfake detection via facial landmark motion analysis during person-specific word pronunciation

May 17, 2024

Jannik Zamboni

First examiner: Prof. Dr. Marina Tropmann-Frick
Second examiner: Prof. Dr. Martin Schultz

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Department Technik und Informatik
Berliner Tor 7
20099 Hamburg

Abstract

Deepfakes have become increasingly more popular and realistic over the last few years thanks to the fast advancement in machine learning architectures such as generative adversarial networks (GANs). The creation and detection of these realistic deepfakes remains a cat and mouse game due to the ever-improving nature of the GAN training process. To break out of this cycle, a focus on person specific features instead of general features to detect deepfakes is necessary. This thesis focuses on facial landmark features that are in motion during the pronunciation of a select number of words of a specific target person and utilizes different convolutional neural network (CNN) architectures for its classification. It can be shown that this approach is feasible, produces results with a high confidence and could also be expanded on more words. The trained model is also resistant against various video resolutions and could successfully detect future unseen video resolution formats.

Zusammenfassung

Deepfakes sind in den letzten Jahren dank der rasanten Fortschritte bei Architekturen des maschinellen Lernens wie Generative Adversarial Networks (GANs) immer beliebter und realistischer geworden. Die Erstellung und Erkennung dieser realistischen Deepfakes bleibt ein Katz- und Mausspiel, was in der Natur des GAN-Trainingsvorgangs liegt. Um diesen Kreislauf zu durchbrechen, ist eine Konzentration auf personenspezifische Merkmale anstatt allgemeiner Merkmale zur Erkennung von Deepfakes notwendig. Diese Arbeit konzentriert sich auf Gesichtsmerkmale, die während der Aussprache einer ausgewählten Anzahl von Wörtern einer bestimmten Zielperson in Bewegung sind, und verwendet verschiedene Architekturen von Convolutional Neural Networks (CNNs) für deren Klassifizierung. Es kann gezeigt werden, dass dieser Ansatz praktikabel ist, Ergebnisse mit hoher Zuverlässigkeit liefert und auf weitere Wörter ausgedehnt werden kann. Das trainierte Modell ist außerdem resistent gegen verschiedene Videoauflösungen und könnte erfolgreich Videos von unbekannten Videoauflösungsformaten erkennen.

Contents

List of Figures

List of Tables

1 Introduction

We are living in a time where we see the world through the lense of technology. The same technology that is used in filters on social media, to improve the lighting in photos or to add or remove parts in an image to make it more appealing to list a few examples. In recent years these processes were greatly improved by the advancement of machine learning, which were branded under artificial intelligence, and delivered new capabilities in the synthesis and manipulation of convincing looking media even up to the point of creating media that has never been seen before and that challenges our ability to distinguish a well-made illusion from the truth.

This challenge becomes apparent in the rise of DeepFake manipulation that utilizes deep learning techniques in machine learning to create fake videos based on a swap of the face of the involved person with another person's face. Examples are political figures that deliver fabricated speeches or celebrities partaking in movies they have not been cast for. The term "Deepfake" originated from the username of a Reddit user named "deepfakes" who claimed in 2017 to have used deep learning algorithms to transpose celebrity faces into porn videos. (Tolosana et al., 2020) Over the last few years there have been great improvements in the quality, resolution, and its ability to conceive a human. Through the fusion of voice manipulations and facial expressions it challenges our understanding of the integrity of digital content.

To solve this problem deep learning networks were trained that classify an image as real or as a fake. These networks were trained on various sources including fake and original media content and utilized GAN-specific fingerprints, artifacts, and a range in different color features among others. This led to a cat and mouse game due to the ever-improving generation of fake media content as well as the classification thereof.

Due to the nature of these training processes that require vast computational resources, to complete the process in a reasonable amount of time, and that make use of a diverse training dataset of many people, a generalized network is trained that can be used with a certain confidence for different people. However, due to the variation in the training data of these individuals, the resulting network may not be able to imitate

each person-specific motion in every detail and those motions might be substituted by a generalized pattern or a specific dominant but random motion from the training data.

The main hypothesis that is being followed in this bachelor thesis is that the facial landmark motion during a word-specific pronunciation might be a way to identify an original video from a fake one if the detector is only trained on the input data of one specific person. Other research questions are whether this identifier would be independent of video resolutions, as that is something other detectors are struggling with, and what benefits different architectures might have. However the detector should also stay competitive to other previous works ideally at the same time.

The person of interest for this project will be Barack Obama as there are already some samples of deepfake available as well as archived original videos from the time of his US presidency. A classifier that is able to identify the facial landmark motion for a selected number of the most common words will be created for this project as proof of work.

Structure of this thesis

The second chapter of this thesis gives an overview of different categories of digital image manipulation techniques as well as some different categories for deepfakes. It also talks about methods that have been used to detect deepfakes such as anomalies in the spatial or frequency domain or in biological signals. At last this chapter also briefly touches how these detection methods could be evaded.

In chapter three the concepts and theory of how neural networks are designed and what mathematical components are necessary in their creation. Some different network architectures as welll as frameworks are also introduced that are used in the project for this thesis.

The fourth chapter talks about the process of data preparation that was done for this thesis as well as how some of the mistakes that were made could be fixed.

In the results chapter the experiments done for this project are explained. A large focus of this part is on the experiments that failed and it ends with a successful training of a model.

The thesis ends in the sixth chapter where the research questions are evaluated, the findings are put in context and an outlook for future research is given.

2 Related works

When it comes to research on different techniques and methods that can be used to manipulate digital media such as images, audio or video the list is very long. As audio is not the focus of this bachelor thesis any techniques concerning audio only will be mentioned briefly. Videos consist of multiple images so any technique for image manipulation can apply to videos as well. Therefore a closer look at image manipulation techniques will be taken in this section with a focus on face manipulation. It is important to understand the limitations of different detection techniques so some variations may be added as there is no single solution that is able to detect all manipulations.

2.1 Categories of manipulation

Facial manipulations of images can be categorized in four main groups. (Tolosana et al., 2020) These are entire face synthesis, identity swap, attribute manipulation and expression swap. Entire face synthesis involves the generation of a non-existent face images that achieve a high level of realism. During identity swap the face of one person in a video is replaced by the face of another person. Therefore this would be the category that deepfakes are in. If only certain attributes such as glasses, gender or age are manipulated it would be considered attribute manipulation. For expression swap facial expressions of one person are replaced by the facial expression of another person without swapping or changing the identity of the person in the video.
There are other categories of image manipulation that use original content in a way that changes the narrative of the original. These include lookalikes, doctoring, splicing, omission, isolation or misrepresentation parts or entire videos. (Ajaka et al., 2019)
Lookalikes, also known as impersonators or doppelgangers, of a target person can create videos that lack signs of manipulation as the video itself is real. In combination with low resolution, bad lighting, fog or rain the difficulty to differentiate truth from fake is increased.
Doctoring includes alterations of a video by changing the speed, cropping, adding or

deleting visual information or dubbing the audio, which means the process of replacing the original dialogue in a video with a translated version of the dialogue in a different language (Amberscript, 2023). Dubbing can also be used to create a false narrative through the choice of words that may not match the original meaning. (Brannon et al., 2023)
Splicing describes the process of editing together different videos or rearranging parts of a video to create an altered version of the story whereas omission means editing out portions of a video in a way that changes the presented narrative.
Isolation involves the presentation of a video in a different context that can be achieved by cutting a short clip out of a longer video. Misrepresentation means presenting an unaltered video with a different narrative or description.
All of these categories could include facial manipulations but as the narrative is already changed this would not need to be the case. If the faces are unaltered any detection method that focuses solely on the faces would be unable to make correct predictions.

2.2 Detection

The detection of image manipulations relies on their extracted features in the spatial or frequency domain as well as on biological signals. (Juefei-Xu et al., 2021) Three critical factors for the practicality of a detector that is supposed to be deployed in the wild are its generalization, meaning its ability to accurately predict new data, robustness against various adversarial attacks, which means any sort of transformation or manipulation aimed to deceive the detector and cause misclassification, and the explainability of the detection results when it comes to human interpretation. (Juefei-Xu et al., 2021)

Spatial domain detection involves both visible and invisible artifacts on the spatial domain. This includes image forensics based detection methods that focus on differences between chrominance components, by highpass filtering to remove image content and investigating the disparities in image residuals (H. Li et al., 2018), photo response non uniformity pattern, which looks at the noise in digital camera images from a certain sensor and therefore works similar to a fingerprint of a camera (Koopman et al., 2018), or leveraging local motion features from real videos to identify if the local motion of keypoints is consistent. (G. Wang et al., 2020) Deep neural network (DNN)-based detection methods are data-driven that extract spatial features to improve generalization in the detection of images created by various DNNs, but suffer from additive noises or especially adversarial noise attacks. Some methods focus on obvious artifact clues

that are distinct patterns of individuals like facial and head movements. (S. Agarwal, El-Gaaly et al., 2020) Others focus on locating manipulated regions and making them visible so that future detectors can focus on these regions. (Y. Huang et al., 2020) Another angle of detection is facial image preprocessing, that aims at extracting local features of convolutional traces in a facial image that is then analyzed by a simpler classifier. (Guarnera et al., 2020) Most popular are spatial detection attempts but with increasingly realistic deepfakes these attempts might be less effective as each new generation of GANs and DNNs could learn to reduce or avoid the artifacts and fingerprints that current DNN spatial detection techniques rely on. (Juefei-Xu et al., 2021)

Frequency domain detection focuses on artifacts that are created by GANs due to limitations or simplifications in their architecture. This could be a limited frequency of saturated pixels or artifacts due to the upsampling in a network design. (McCloskey und Albright, 2018) (Zhang et al., 2019) These artifacts could be destroyed by simple perturbation attacks. (Yu et al., 2018) Frequency based detection methods are less effective against compression, reconstruction, blurring but generalize well on unknown synthetic techniques with similar architectures. (Zhang et al., 2019)(Yu et al., 2018)

Detection based on biological signals involves the consistency between visual and acoustic signals. This can be letters such as M, B, P that have to start with closed lips. (S. Agarwal, Farid et al., 2020) Another example is that faked faces are sometimes fixed in their size or that synthesized faces lack eye blinking. (Y. Li und Lyu, 2018) (Y. Li et al., 2018) Other attempts include determining the heart rate from a video or the presence of blood flow based on subtle color differences in the human skin. (Qi et al., 2020)(Hernandez-Ortega et al., 2020)

Evasion

Evasion methods of deepfake detection include adversarial attacks that add imperceptible adversarial perturbations to significantly reduce accuracy of detectors. (Carlini und Farid, 2020) There are multiple other variants of this type but all add noise or reduce image quality. (S. Wang et al., 2019) (Juefei-Xu et al., 2021) A different method focuses on removing fake traces in the frequency domain which improves the synthesis quality significantly. (Jiang et al., 2020) Newer trained GANs can generate images with realistic frequencies. (Jung und Keuper, 2020) A third method uses advanced image filtering or generative models to get rid of fake traces without a reduction in image quality. (Neves et al., 2019)

3 Fundamentals

The goal of machine learning is to have a computer model mimic the behavior that is done by human experts to achieve a desired outcome. (IBM, 2021) For this to work it is not required to define the necessary steps for this process but instead it is important to define the input with synchronized output that the model can attempt to extract the desired connections from. Ideally the model can then be used in a more general context if the training dataset and its context are similar enough and within the same constraints as the new context.

3.1 Neural Networks

In machine learning it is common to loosely base the architecture for the learning process on the biological neurons in the human brain (Kaste, 2023). This concept is abstracted and transferred on a mathematical activation function that calculates an output for a given input. If the output is above a certain threshold, the neuron will be activated and data is sent to the next layer in a network of multiple neurons (IBM, 2021). Each neuron is connected to at least one other neuron and has a weight, threshold and bias assigned to it. The calculation from an input to an output throughout the entire network is called forward propagation (Lin, 2023). The weights are randomly initialized and the bias is initialized with 0.

During forward propagation the input is passed through the network only in one direction through the hidden layers to the output layer. The network does not have any loops that might prevent the successful computation of the output (deepai.org, 2024b). In Figure 3.1 there are 4 inputs x_1 to x_4 that are connected to the first hidden layer. Each of the connections between the neurons has a weight assigned to it and each of the neurons in the hidden layer has a bias as well. When an input is received each neuron in the first hidden layer will calculate the weighted sum of all its connected

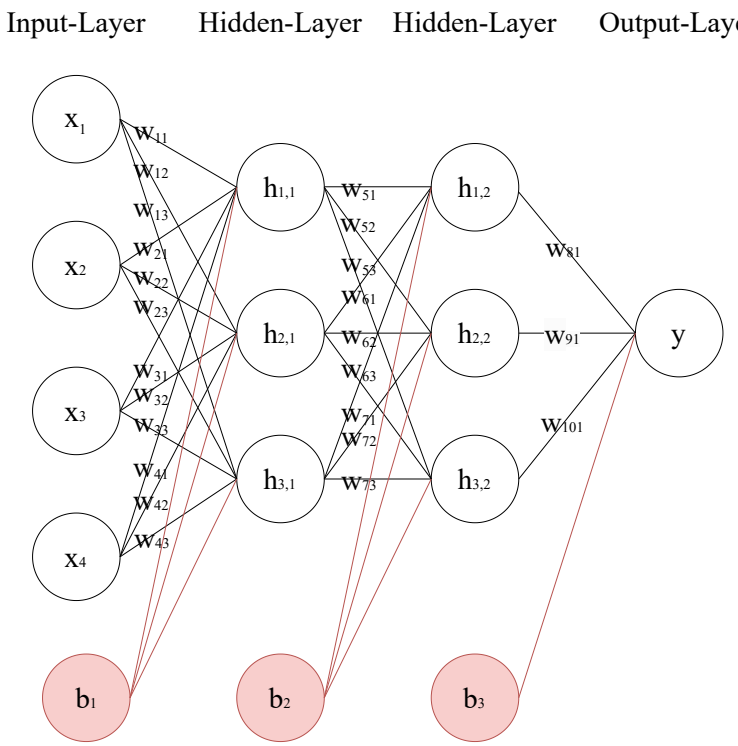

Figure 3.1: Neural Network

input values and will add the bias. Malhotra defines the equation in (2018) as:

$$a(x) = b + \sum_i w_i * x_i \tag{3.1}$$

This result is then passed to the activation function which Malhotra defines in (2018) as:

$$h(x) = g(a(x)) \tag{3.2}$$

This function calculates the non-linear output of the neuron (Iuhaniwal, 2019). This process is repeated for all neurons of the hidden layers as well as the output neuron. The inputs for the second layer of hidden neurons are the outputs of the previous neuron layer.

During the training of a network the provided input data is passed to the network and a result is calculated (Oppermann, 2024b). The combination of input and output will be measured by a loss function, that determines how good the model is performing for its

designated task (Kaste, 2023). To improve and optimize the parameters of a network, that are represented by the weights, a back propagation algorithm is applied. The goal of this algorithm is to achieve a gradient descent in the error surface and thereby approach a minimum of the loss function by optimizing the weights of the network (Kaste, 2023). The loss function is used to calculate the partial derivative for all the weights which are then updated in the direction of a negative gradient. The change of the weights can be descriped as:

$$\triangle W = \mu * \triangledown E(W) \tag{3.3}$$

The learning rate determines how wide each step can be taken and is essential for how fast a local minimum can be reached and also if other suitable minimum are skipped (Kaste, 2023).

Under- and Overfitting

Overfitting is a machine learning behavior that occurs when the model adapts too well on the training dataset and looses its ability to generalize well on new unseen data. To prevent this behavior the dataset is usually split into training dataset and test dataset in order to test if the accuracy between both datasets differs. In this case the model likely adapted too close to the training dataset. (IBM, 2024)(AWS, 2024b)

Similar to overfitting is underfitting which is a behavior that occurs when the model failed to identify dominant trends or datapoints. This happens in cases where there are too many relevant features or the training process was stopped early. As a result the accuracy for both the training dataset as well as any test dataset or new unseen dataset is low. (IBM, 2024)(AWS, 2024b)

3.1.1 Regularization

"Regularization is a set of methods for reducing overfitting in machine learning models. Typically, regularization trades a marginal decrease in training accuracy for an increase in generalizability." (Murel, 2023) Common examples are L1, L2 and elastic net regularization for linear models as well as dropout for neural networks and early stopping for model training.

L1 regularization is also known as L1 norm or Lasso regression (Murel, 2023). It is supposed to penalize high-value and correlated coefficients (Pykes, 2023). The cost

function is altered and a penalty term (also known as a regularization term) added to it. This penalty term is the absolute weights of each individual feature. It is known to perform feature selection by shrinking the coefficients of less important features to zero which will make some features obsolete. (Javatpoint, 2017) This results in a sparse model where only a subset of features are used (Islam et al., 2024).

L2 regularization is also known as Ridge regression (Gupta, 2017). It is similar to L1 regularization but makes use of the squared weights instead of their absolute values (Javatpoint, 2017). Therefore this technique is more prone to outliers (Pykes, 2023). L2 regularization shrinks the coefficients of less important features but does not set them to zero (Murel, 2023). This results in a model where all features are used, but the less important features have smaller coefficients (Islam et al., 2024).

Elastic net regularization combines both penalty terms of L1 and L2 regularization into the squared errors loss function Murel, 2023). The L1 and L2 norm are muliplied by two hyperparameters, α and λ, and the L1 norm is used to perform feature selection, whereas the L2 norm is used to perform feature shrinkage (Dhumne, 2023).

Dropout can be utilized in neural networks to randomly drop out nodes including their input or output connections during training from the network. This way several variations of the network architecture are trained and the network without any left out nodes is used for tests. After the training process, the result relies on an approximation of the average modified training architectures and generalizes therefore better when compared to an trained network without dropout. (Murel, 2023)

Early stopping is a regularization technique that is used during the training of a model. After every epoch the validation accuracy is checked if it has plateaued within a set number of epochs. The training is stopped in that case and a further increase in the validation error is prevented (Murel, 2023).

3.1.2 Loss functions

A loss function measures how well a neural network has done a certain task such as regression or classification. For classification tasks there are two popular loss functions that were also used in this project. These are binary cross entropy and categorical cross entropy (Chakravarthy, 2020). In general a small loss is better than a large loss (Chakravarthy, 2020).

Binary crossentropy is used in case of binary decisions between two classes. The loss provides insights of how accurate the predictions of the model are (Yathish, 2022). This is measured for each class and the final loss contains the average of both involved classes. A mathematical representation for binary crossentropy where $\hat{y}$ is the predicted value according to (Chakravarthy, 2020) is:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * log(\hat{y}_i) + (1 - y) * log(1 - \hat{y}_i)) \tag{3.4}$$

For categorical crossentropy the function compares multiple classes against each other and focuses on how the predictions are distributed compared to the true distribution (Chakravarthy, 2020). Categorical crossentropy is very similar to binary crossentropy when the number of classes is 2. The mathematical equation stated in Yathish, 2022 is:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{M}(y * log(\hat{y}_i) + (1 - y) * log(1 - \hat{y}_i)) \tag{3.5}$$

3.1.3 Optimization functions

In principle optimizers are methods that are utilized to update the weights of a neural network during backpropagation to reduce the loss (Doshi, 2019).

Gradient descent is the most basic optimization algorithm (Doshi, 2019). The gradient is calculated for the full dataset and the average is used to update the weights. Due to this procedure it may take a lot of resources to calculate when the dataset is very large (Baeldung, 2023). It might also get trapped at a local minimum (Doshi, 2019).

Stochastic gradient descent (SGD) is similar to gradient descent but does not calculate every entry in the dataset before updating the weights. Instead a portion that is chosen at random is used (Baeldung, 2023). The main advantage of this is an increased speed compared to gradient descent. However the result can be more erratic as the convergence can happen on a local minimum instead of the true minimum and the steps are taken after a random sample of the dataset (Vungarala, 2023).

The Adam optimizer combines the strengths of Momentum Update, which introduces a fraction of the weight adaptation of the previous time step to continue convergence in case of a shallow local minimum or a change in the gradient direction, and RMSprop, which introduces an adaptive learning rate based on a division between the current

gradient and the moving average of the squared sum of the previous gradients that descends over time (Sharma, 2023). This causes the learning rate to increase when the variance of gradients is low and the learning rate to decrease when the variance of gradients is high. (R. Agarwal, 2023)

The steps for Adam are to initialize the weights, use the loss function to calculate the gradient as well as the moving averages of squared values, to correct the bias that are part of the moving average and to update the weights accorrdingly until a threshold or iteration count is reached to end the calculation (deepai.org, 2024a). These calculations are shown as follows:

$$m_0 = 0, v_0 = 0 \tag{3.6}$$

In equation 3.6 at time $t = 0$ an initialization with 0 of the first and second impulseterm m_0 and v_0 is taking place (Oppermann, 2024c).

$$m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1)\nabla_\theta L(\theta) \tag{3.7}$$

The equation 3.7 is showing the first impulse m_t. β_1 is a hyperparameter that is also multiplied as $1 - \beta_1$ with the current gradient (Oppermann, 2024c). It functions similar to stochastic gradient descent (SGD) with impulse.

$$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2)\nabla_\theta L(\theta)^2 \tag{3.8}$$

Equation 3.8 has v_t as a second impulse. β_2 is another hyperparameter that is multiplied with the squared current gradient. This effectively works similar to RMSprop (Oppermann, 2024c).

$$\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_t + 1} + 1\epsilon^{-5}} m_t + 1 \tag{3.9}$$

The equation 3.9 combines both parts of 3.2 and 3.3 to have the advantages of SGD with impulse and RMSprop combined in the Adam optimizer (Oppermann, 2024c).

$$m_{t+1} \leftarrow \frac{m_{t+1}}{1 - \beta_1^t} \qquad v_{t+1} \leftarrow \frac{v_{t+1}}{1 - \beta_2^t} \tag{3.10}$$

The equations in 3.10 show the bias correction that needs to take place as the first and second impulse were initialized with zero. This can cause large steps in the update of the weights in the first few steps which can be avoided by this bias correction (Oppermann, 2024c).

3.1.4 Activation functions

The role of an activation function is to enable a neural network to learn patterns in data that can be very complex. As not all patterns can be predicted by a linear classifier (Jain, 2019). Without an activation function the output of a neuron would follow the format $w * x + b$ that is of degree 1 and therefore linear (Jain, 2019).

The Sigmoid or Logistic Sigmoid function is defined as:

$$LogisticSigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3.11}$$

It squishes all outputs between [0,1] and shows a saturation towards high and low inputs. This makes it prone to the vanishing gradient problem. As the output is also not centered around zero it tends to have poorer convergence compared to other functions (Dubey et al., 2021). This function plays a central role in binary classification (Topper, 2023). It is also usually used in the last layer of a neural network.

The Softmax function is generally used in the last layer of a neural network for multi class classification (Franco, 2024). Its purpose is to provide a probability distribution in the interval [0,1] for all classes (Dubey et al., 2021). The combined values of the distribution is equal to 1 (Franco, 2024). The function itself according to Lang, 2023 is defined as:

$$Softmax(x_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} for k = 1, ..., K \tag{3.12}$$

The input to this function is a vector of K elements, where Z is the representation of this vector and z_k the representation of an element of said vector.

ReLU stands for Rectified Linear Unit and is an activation function which is the most often used activation function for deep learning tasks (Krishnamurthy, 2024). The activation itself for inputs above 0 are linear. ReLU is defined as $R(x) = max(0, x)$ or in a more precise definition as stated in (Oppermann, 2024a):

$$R(x) = \begin{cases} x & if \quad x \geq 0 \\ 0 & otherwise \end{cases} \tag{3.13}$$

"ReLU has been shown to accelerate the convergence of the gradient descent toward the global minimum of the loss function compared to other activation functions." as stated by (Oppermann, 2024a) while also being computationally less expensive.

3.1.5 Convolutional Neural Networks

CNNs are generally used with a more dimensional input as it would be the case with a 2-dimensional image. This image will be processed by a kernel that is slid across the image and multiplied with the input to reduce the kernel input to a single value. Usually this kernel size is a square of 2,3 or 4 leading to a total size of 4, 9 or 16. The stride controls how many steps are necessary to slide the kernel across the image. The kernel generally starts on the top left, moves to the right while doing calculations, then slides one step down and moves all the way to the left but without any calculations done and starts again until it reaches to bottom right of the image. This process reduces a 10 by 10 image with a 3 by 3 kernel to a 8 by 8 image and is referred to as convolution (Ganesh, 2019).

Each image can also have multiple channels that are used for colors. This is processed by running the kernel over the image once for each channel. In addition multiple kernels of the same size but with different randomly initialized weights and biases can be used as well. If the 10 by 10 image mentioned above would have 3 channels for colors and is processed by 5 different kernels, this leads to a 8 by 8 image with 15 channels. The input image is therefore transformed and convoluted into a smaller but deeper object. This process will happen for each convolutional layer. These resulting images are called feature maps, as each of the 15 feature maps can contain information about a different feature of the original input data.

A group of convolutional layers is followed by another layer that is supposed to further reduce the size of the image. For this step the max-pooling layer is common that halves the input image size with a 2 by 2 window size. For this it takes for each 2 by 2 segment of pixels only the maximum value. The channel number is not increased by this. Before being passed to the final steps of the network, the input is flattened to 1 dimension, and the following layers behave like the neural network described above (Skansi, 2018).

This architecture has the benefit of reducing the amount of trainable parameters significantly compared to a standard neural network. The CNN trains noticeably faster because of this (Mahajan, 2020).

3.1.6 General Adversarial Networks

GAN is an architecture that involves two models that are adversaries of another. One generative model G, is supposed to generate new authentic data based on the original

data distribution and random noise as input, whereas another model D, is estimating the probability of a sample being from the original dataset or from G.
In this setup both models are being trained alternately. During this training G is being optimized to maximize the probability of mistakes being made by D. The training process for D is instead optimizing of minimizing the probability of making mistakes. This training process is continued until the samples generated by G are realistic enough that D can no longer distinguish a fake from an original (AWS, 2024a).

This architecture is frequently used in generating images, such as transferring the style of one image to another, change black-and-white images to color or upscale images or videos to a higher resolution (Goodfellow et al., 2014).

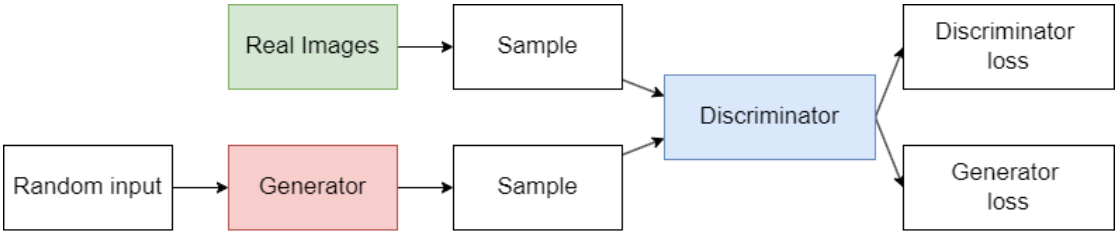

Figure 3.2: GAN architecture (Google, 2022)

3.1.7 VGG 16 model

The VGG 16 model is a CNN with 16 layers, that was designed to classify 1000 classes in a large image dataset (Simonyan und Zisserman, 2015). It has a 224x224x3 input layer followed by 13 convolutional layers with ReLu activation function which is followed by 3 fully connected layers (FC). These last 3 layers have 4096 neurons for the first two layers and 1000 neurons for the third. The network ends with a softmax activation function (Datagen, 2024b).

Table 3.1: VGG-16 architecture (Datagen, 2024b)

Input	224 x 224 RGB image
Block 1	3x3 Conv,64
	3x3 Conv,64
	2x2 MaxPool
Block 2	3x3 Conv, 128
	3x3 Conv, 128
	2x2 MaxPool
Block 3	3x3 Conv, 256
	3x3 Conv, 256
	3x3 Conv, 256
	2x2 MaxPool
Block 4	3x3 Conv, 512
	3x3 Conv, 512
	3x3 Conv, 512
	2x2 MaxPool
Block 5	3x3 Conv, 512
	3x3 Conv, 512
	3x3 Conv, 512
	2x2 MaxPool
Block 6	FC 4096
	FC 4096
	FC 1000
	Softmax

3.1.8 ResNet 50 model

ResNet is shortened from Residual Network and ResNet-50 is a convolutional network that has a total of 50 layers (Datagen, 2024a). 48 of those are convolutional layers, one is a max pool layer and another an average pool layer. Residual networks were designed to solve a vanishing gradient problem that occurred as architectures became increasingly deeper (Kundu, 2023).

The vanishing gradient problem is manifesting in the backpropagation during training when small gradients (i.e. 10^{-5})) are multiplied with each other (Mukherjee, 2022). This leads to changes to the last layers of the network which are getting smaller with each multiplication during backpropagation towards the beginning of the network.
The exploding gradient problem is similar to the vanishing gradient problem, as the gradients can reach orders of up to 10^4 or more. Depending on the number of multiplications the gradient can move towards infinity (Mukherjee, 2022).
Both these problems slow down the learning process up to the point where the learning process can get stuck.

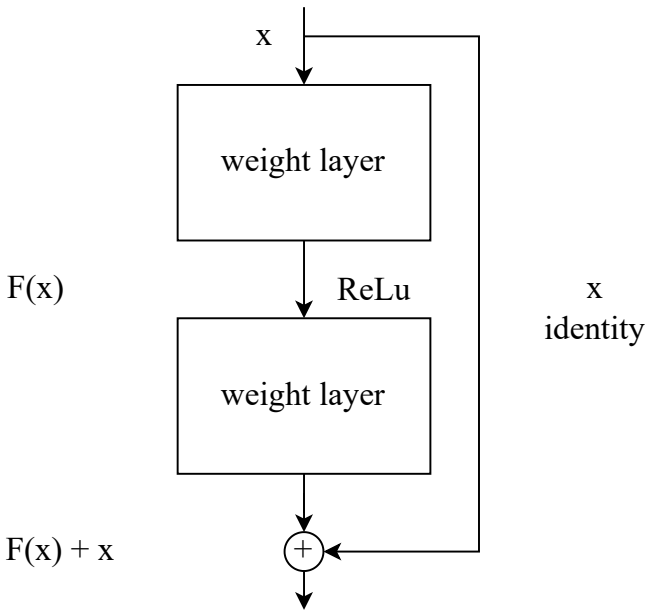

Figure 3.3: Residual Block (He et al., 2015)

The residual network can solve this problem by including residual blocks in its architecture. These blocks add skip connections to the network that add the activation of one layer

to the output of a later layer (He et al., 2015). Due to this some layers, that might have a small gradient, can be skipped if these do not contribute anything to the result. The benefit of this is that the network can be much deeper than conventional CNNs (Kundu, 2023).

Table 3.2: ResNet-50 architecture (He et al., 2015)

Input	224 x 224 RGB image
Block 1	7x7 Conv,64
Block 2	3x3 MaxPool
Block 3 (Residual)	$\begin{bmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{bmatrix}$ x 3
Block 4 (Residual)	$\begin{bmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{bmatrix}$ x 4
Block 5 (Residual)	$\begin{bmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{bmatrix}$ x 6
Block 6 (Residual)	$\begin{bmatrix} 1x1, 512 \\ 3x3, 512 \\ 1x1, 2048 \end{bmatrix}$ x 3
Block 7	2x2 AveragePool
	FC 1000
	Softmax

3.1.9 DenseNet121 model

DenseNet121 is part of a group of CNNs that have a pattern between the layers of the architecture called dense connectivity, which creates a connection between each layer while using the output of previous layers as the concatenated input for a layer (NocodingAI, 2023). This improves the feature reusability, reduces the problem of the vanishing gradient problem, has a positive influence on feature propagation and requires less parameters in the network (G. Huang et al., 2016). The name DenseNet stands for dense convolutional neural networks. The name DenseNet121 derives from its 121 layers.

3.2 Whisper

Whisper is a name that is based on web-scale supervised pretraining for speech recognition (Radford et al., 2022). It is an automatic speech recognition system (ASR) that was trained on "680,000 hours of multilingual and multitask supervised labeled audio data" (OpenAI, 2022), which were collected from various unspecified sources from the internet (OpenAI, 2022). It can provide transcriptions in English and multiple other languages as well as providing direct translations from non-English audio into an English transcript (OpenAI, 2022).

"The Whisper architecture is a simple end-to-end approach, implemented as an encoder-decoder Transformer. Input audio is split into 30-second chunks, converted into a log-Mel spectrogram, with a 25 ms window and a stride of 10 ms, and then passed into an encoder. A decoder is trained to predict the corresponding text caption, intermixed with special tokens that direct the single model to perform tasks such as language identification, phrase-level timestamps, multilingual speech transcription, and to-English speech translation" as stated in (OpenAI, 2022).

Due to the large dataset that features diverse audio data of Whisper, of which the English language makes up about two thirds of, the model is robust against additive noise but is outperformed by other models under low noise (Radford et al., 2022)(OpenAI, 2022).

Whisper can take only 30-second audio context at once and in order to solve the problem of transcribing longer audio it relies on predictions of the timestamp tokens and shifts the 30-second window across the audio data (Radford et al., 2022). With this approach an inaccurate transcription or timestamp may impact transcriptions in any subsequent window negatively (Radford et al., 2022).
To avoid failures in long-form transcription a set of heuristics was developed that is based on beam search and the temperature for selecting tokens. This made the model more reliable but did not solve the issue that the model potentially ignores the first few words in the audio input. To solve this issue the initial timestamp token constraints were set to be between 0.0 and 1.0 second (Radford et al., 2022).

whisper-timestamped

The original Whisper model is limited in its prediction of timestamps on a phrase-level instead of a word-level. As the latter is required in this project to extract the facial landmark motion during the pronunciation of one word, an extension to Whisper is

needed to achieve this. The chosen extension to solve this problem is called whisper-timestamped which is compatible with other original versions of Whisper. The approach of whisper-timestamped is based on the dynamic time warping (DTW) technique, which is a "popular technique for comparing time series, providing both a distance measure that is insensitive to local compression and stretches and the warping which optimally deforms one of the two input series onto the other" as stated in (Giorgino, 2009). This technique is then applied to cross-attention weights (Louradour, 2023). It improves the original Whisper model by including a confidence score for each word, word-level timestamps and a detector for voice activity to avoid hallucinations that occur when a word or phrase appears in the transcription for a silent audio part (Louradour, 2023).

3.3 MediaPipe

MediaPipe is a framework that includes several machine learning tasks that can be deployed on multiple devices such as mobile, web, desktop and others. It is published by Google as open source and is used in Google's own products such as Google Lense, Google Meet, YouTube and Google Photos.

Under MediaPipe solutions preview are several on-device machine learning solutions made available that are easily integrated with low-code API. These solutions range from multiple solutions for vision tasks such as gesture recognition, image classification and face landmark detection to text tasks and audio tasks. (Google, 2024e)

The facial landmark detection task is used in the extraction of facial landmarks for this project.

MediaPipe Face Landmarker

The MediaPipe Face Landmarker task is able to detect face landmarks and facial expressions in images and videos (Google, 2024b). It can be used to identify human facial expressions, apply facial filters and effects, and create virtual avatars. "This task uses machine learning (ML) models that can work with single images or a continuous stream of images. The task outputs 3-dimensional face landmarks, blendshape scores (coefficients representing facial expression) to infer detailed facial surfaces in real-time, and transformation matrices to perform the transformations required for effects

rendering" as stated in (Google, 2024b). The total number of 3-dimensional face landmarks provided by this task is 478.

Figure 3.4: Facial Landmark Detection Task

3.4 TensorFlow

"TensorFlow is an open-source platform for machine learning using data flow graphs. Each node in the graph represents mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them" as stated by (Nvidia, 2024).

"There are three distinct parts that define the TensorFlow workflow, namely preprocessing of data, building the model, and training the model to make predictions. The primary method is by building a computational graph that defines a dataflow for training the model. The second, and often more intuitive method, is using eager execution, which follows imperative programming principles and evaluates operations immediately." as written by (Nvidia, 2024).

Keras

Keras is a deep learning API written in Python that focuses on debugging speed, code elegance and conciseness, maintainability, and deployability. It is capable of running on top of Tensor Flow, JAX or PyTorch (Google, 2024c)
Keras main strength is its consistency and simpliciy that helps in reducing the cognitive load while minimizing the number of user actions required for common use cases (Google, 2024a).

Keras Tuner

"KerasTuner is an easy-to-use, scalable hyperparameter optimization framework that solves the pain points of hyperparameter search. Easily configure your search space with a define-by-run syntax, then leverage one of the available search algorithms to find the best hyperparameter values for your models. KerasTuner comes with Bayesian Optimization, Hyperband, and Random Search algorithms built-in, and is also designed to be easy for researchers to extend in order to experiment with new search algorithms." as stated in (Google, 2024d).

3.5 Wav2Lip

Wav2Lip is a new model, which is an adapted and trained version of SyncNet (Raina und Arora, 2022). Wav2Lip is speaker-independent and able to provide lip-sync accuracy that matches real synced videos (Prajwal et al., 2020). This model is created with the utilization of a lip-sync discriminator for adequate penalization of wrong lip shapes as well as a second discriminator, with an architecture similar to LipGAN, for better visual quality (Prajwal et al., 2020). With the lip-sync discriminator it is possible to train a generator to generate accurate and realistic lip-motion more consistently. The second discriminator is needed to reduce visible artifacts around the morphed regions of the lip shapes (Prajwal et al., 2020)

4 Data preprocessing

4.1 Collecting videos and transcribing words

The original data has been collected from the presidential weekly addresses during the presidency of Barack Obama. Each of these videos has then been briefly screened to identify the presence of additional faces or different speakers than the target person. From the selection removed were all videos that did not have the face of Barack Obama visible throughout the whole video such as during the display of images or diagrams as shown in figure 4.1. Furthermore, the videos each had a part with the logo of the White House in the beginning as well as in the end. Unfortunately, these sections in the videos did not have a fixed duration but instead showed various lengths that changed slightly over the course of the years. To get rid of the undesired sections, the videos were clipped with the help of the python library moviepy. It was possible to identify silent parts in the videos that were longer than the given minimum silence length of 4 seconds, which only happened in the beginning and the end of the videos, and to only keep the desired part of the clip. At the end of this step were 372 videos remaining.

Figure 4.1: Selection of removed videos

These videos ranged from 5.3 MB to 77 MB in size as the video data rate and video total bit rate ranged from 186 kBit/s or 314 kBit/s up to 1767 kBit/s or 1893 kBit/s. The duration of the videos ranged from 2:08 min to 7:56 min. The framerate per second remained the same but a small portion of 3 videos were in a resolution of 640x360 pixels instead of 1280x720 pixels.

After this step the audio was extracted for each video with the python library pydub and saved in the WAV-format. This format was selected as it does not use any compression and therefore removes the risk of losing information compared to other formats like MP3.

To receive a transcription of the audio files a modified version of the Whisper model is used. This model named whisper-timestamped can provide word-level timestamps and confidence additional to the transcription. It is recommended to use the aid of hardware acceleration with GPU that is supported by PyTorch as this step is fairly time consuming (PyTorch, 2024).
Even in the standard version of Whisper there are several different models available. These differ mainly in the language selection, as there are models that are solely trained on the English language as well as other models trained on multiple languages. In this project the medium.en model has been selected as it provided better accuracy for the word timestamps compared to smaller models as well as also having the benefit of only expecting English audio input.
The result of the transcription is saved in the JSON-format as a text file. The file size for these JSON-files ranged from 63 KB to 252 KB.

The resulting files of the previous step were sorted and the most common 150 words searched for. In this step a conversion to lower case letters as well as regular expressions to drop non-letters were utilized to treat “Hello”, “hello” and “hello!” as the same word. This was necessary as the transcription from Whisper includes various punctuation marks that were not removed by the whisper-timestamped model and identification of words.

The total number of words in all 372 videos that were kept in the sorting process was 229092 and the number of unique words was 9182. The most common word “the” appeared 10246 times and the 150th most common word appeared only 235 times.

The most common 150 words still made up 134921 of all words in all videos or 58.89%. As this reduced the workload of words to process only by half a further reduction in the number of words to be used in the scope of this project is necessary.
Continuing with the most common 25 words as shown in table 4.1 this reduced the

representation among all words to 33.48% as these words still appeared combined 76707 times. The number of appearances between the most common word and the 25th most common words was an order of magnitude as “the” appears 10246 times and “but” appeared only 1081 times.
Even the most common 5 words only reduced the representation among all words to 16.56% as the combined occurrences accumulated to 37944. However, the fact that all most common 25 words occur more than 1000 times has been used later in the creation of the training dataset.

Table 4.1: 25 most common words

1. the	10246	2. and	8665	3. to	8563	4. of	5580	5. a	4890
6. in	3969	7. that	3877	8. we	3400	9. our	3367	10. for	2822
11. this	2189	12. on	1705	13. is	1677	14. have	1632	15. i	1491
16. it	1439	17. more	1413	18. are	1378	19. will	1339	20. their	1338
21. as	1259	22. you	1147	23. with	1128	24. thats	1112	25. but	1081

4.2 Creating sub clips

In the previous section the timestamps on a word level were already identified and stored in the JSON-files.

When getting to the point of creating sub clips for these timestamps and trying to isolate each word to a different clip it became apparent that the timestamps themselves were not very accurate. Sometimes they were shorter or longer than the actual duration of the word. In some cases, this happened because the following word came after a punctuation mark, end of sentence or paragraph or due to some other form of pause during speech. Sometimes the start and end times for a word were too short isolating only the first half of a word and dropping the second half.
This was the case even with the extension whisper-timestamped that was used to identify the timestamps on a word-level.

Part of the reason for this can be explained by an earlier attempt to isolate the words into sub clips, that was done before the silence parts with the logos in the beginning and end of each video were removed. This caused a shift throughout the entire prediction of the timestamps by several seconds. In these cases, the words were still transcribed correctly but the prediction for the beginning of a word or even sentence could be

marked after that part was already over. However even after these silent parts were removed some inconsistencies remained in the predicted timestamps on a word-level.

To account for some inaccuracy in the prediction, that cannot be fully removed without a model specifically trained for this task, a buffer for each word was introduced. This buffer was based on the average word duration that an average speaker of the English language takes when speaking roughly 140 words per minute. This leads to roughly 0.42 seconds on average per word. This buffer was then divided in half and added to the beginning and end of each word. With this buffer the probability of the timestamps wrapping the full word greatly increased.

It is important to point out that this also caused shorter words to contain parts of other words in almost all instances. As this is going to be the case in all stages of training and testing as well as during the prediction when the model is deployed, it might lower the potential negative impact it has on the actual results of the model.

The sub clips for each individual word were then saved to disk. Similarly to the JSON-files these clips varied greatly in size from 50 KB to 225 KB. This was caused mainly by different video lengths and bit rate. In the initial attempts of creating the sub clips the amount of all words let to over 200k+ files. In order to be able to find files for a specific video quickly and to avoid the need to open and load a single folder with thousands of files, each original video received its own folder.
The naming convention for the files was the date of the weekly address video followed by the word that was isolated in the subclip and then ten digits that are incremented for each word in the video. A template for this might look like “MM_DD_YYYY_WORD_10DIGITS”. The third word of the sub clip created on 1st January 2011 would therefore be named “01_01_2011_the_0000000003.mp4”.

Even though one video that was split up into sub clips showed similar variations to the different originals in video data rate and video total bit rate. This caused the file size for the sub clips to range between 36 KB and 220 KB for the video mentioned in the naming example above. The framerate per second remained the same but a small portion of subclips were in a resolution of 640x360 pixels instead of 1280x720 pixels.

For all 372 processed videos were a total of 134536 sub clips created for the 150 most common words. These are 385 words fewer than compared to the 150 most common words counted in the JSON files. The reason is that some videos were split into sub clips with very inaccurate timings compared to other sub clips and some had elements in them that prevented the target face to be seen at all times. These faulty clips were

removed. The total size of the remaining sub clips is 14.8 GB on disk at the end of this step.

4.3 Facial landmark extraction

With the MediaPipe Face Landmarker task it was possible to receive 3-dimensional face landmarks for 478 points of interest on each face. The initial idea for this step was to create the frames separately for each video. After the attempt of the process for the first video this resulted in 6593 items totaling 5.7 GB. To process all 372 videos in this way would have resulted in approximately 2 TB of data. Therefore this approach was quickly dropped. In addition the number of words processed was also lowered from 150 to 25 of the most common words.

Instead of extracting the face landmarks from a photo it was then attempted to extract the face landmarks directly from video. To do so the library OpenCV2 has been used to capture each video frame. Each frame captured this way was then converted to the RGB color code and changed into a PIL object. This object was then passed to the MediaPipe image object type which could then be used by the detector for the facial landmarks. It was not possible to extract the facial landmarks directly from the frames captured by OpenCV2 due to formatting errors.
During the extraction there were errors in 46 sub clips as the face was not visible in the entire video. These sub clips were then dropped.

After the successful extraction there were then 478 3-dimensional normalized landmark data points per frame available in a numpy array. Other features that the MediaPipe Face Landmarker task was able to extract, such as face blendshapes or facial transformation matrixes, were dropped as they were of no further use.

The numpy arrays were saved in a JSON-file format for each word and each file had an average size of 1 MB. An attempt to use the CSV-file format instead resulted in a similar file size and did not yield any significant improvements. There were at this stage 76276 facial landmark files available that totaled 77.1 GB in size.

Preparation of lists

With the already existing lists for the most common 25 words these are split into the most common 5 (group A) and the most common 6 to 25 words (group B).
For each of these lists 1000 files for each word were then processed and added to the list. The last list of group B only had 996 occurrences on its own and so the remaining 4 were added from words of the same group. Now all 25 lists had 1000 entries.
The words of group B were then split into 5 new lists evenly in a way that 200 entries per word of the 20 words in group B were in each of the new lists with a total of 4000 files per list. These 5 lists of group B were then concatenated together to form one list of 20.000 entries for group B.

Table 4.2: Most common 5 words - group A

1. the	1000	2. and	1000	3. to	1000	4. of	1000	5. a	1000

Table 4.3: Most common 6-25 words - group B

6. in	1000	7. that	1000	8. we	1000	9. our	1000	10. for	1000
11. this	1000	12. on	1000	13. is	1000	14. have	1000	15. i	1000
16. it	1000	17. more	1000	18. are	1000	19. will	1000	20. their	1000
21. as	1000	22. you	1000	23. with	1000	24. thats	1000	25. but	1000

Binary encoding of words

As the words couldn't be passed as letters into the already existing numpy array of the facial landmarks, a representation in binary form was chosen instead. For this the words in question were added to a dictionary that contained the most common 25 words and had a binary representation of their index value added to a dictionary next to the word itself.
This binary encoding was then used in the numpy arrays in the top row as a representation of which word was linked to the rest of the array.

The word "the" did translate in this binary representation to "0" whereas "and" was represented by "1". As each row had 90 columns, and could therefore contain up to 90 numbers, which were not fully filled by this representation, the other columns were filled with zeros.

After this step the structure of the facial landmarks looked similar to:

	frame 1	frame 2	frame 3	...	frame 90
binary word representation	0. 0. 0.	0. 0. 0.	0. 0. 0.	...	0. 0. 1.
1st facial landmark	x1 y1 z1	x1 y1 z1	x1 y1 z1	...	x1 y1 z1
2nd facial landmark	x1 y1 z1	x1 y1 z1	x1 y1 z1	...	x1 y1 z1
3rd facial landmark	x1 y1 z1	x1 y1 z1	x1 y1 z1	...	x1 y1 z1
4th facial landmark	x1 y1 z1	x1 y1 z1	x1 y1 z1	...	x1 y1 z1
5th facial landmark	x1 y1 z1	x1 y1 z1	x1 y1 z1	...	x1 y1 z1
...	...	...	...	...	...
478th facial landmark	x1 y1 z1	x1 y1 z1	x1 y1 z1	...	x1 y1 z1

4.4 Creating TensorFlow dataset with tensors

With the list of words from group A and group B containing the extracted facial landmarks including the binary word representations in the same file that could be loaded from disk, both lists were used to create a dataset in TensorFlow with "from_tensor_slices". Lists of labels were created for each of these datasets. The labels for the words of group A received "True" and the labels for the words of group B "False". Each dataset was then combined with its corresponding label with the "tensorflow.data.Dataset.zip" function and both datasets saved separately. After this step both datasets were then concatenated with "tensorflow.data.Dataset.concatenate" creating a dataset with 25k entries of a tuple of facial landmark features and binary word combinations as well as a label of type "Boolean".

This dataset could then be saved to disk. Even though it contained 25k entries for facial landmarks and binary word combinations, that made up roughly 25 GB in data when saved in the JSON-format, the new saved TensorFlow dataset format made up only 4.1 GB.

Solving dataset shape problems during training

The dataset was in a shape of (479, 90) when the first problems in training arose. This happened because the CNN expects an input in the form of (None, 479, 90, 1). The first dimension "None" was a placeholder for the batch size and number of files presented for

one training step. The last dimension “1” was important as it represented the number of channels an image can have. Without any channels, that are typically used for color representations such as RGB with 3 channels, the image input is incomplete.

To solve this the dimension was added with only one channel. The numpy function “reshape” is used for this as shown in codeblock 4.1. Each element in the dataset was individually reshaped with the following code snippet:

Codeblock 4.1: Reshaping dataset

```
elem = next(iter(dataset))
x, y = elem
landmark, word = x
landmark.numpy().reshape(landmark, (479,90,1))
```

After this step all elements were in the desired shape. The first dimension of the CNN input was added by batching the dataset before passing it to the training step. In order to run the above code the dataset was split into 5 parts of 5000 files each and later recombined to solve space constraints in the RAM.

Creation of a balanced dataset with 10000 samples

To solve some problems that occurred due to previously imbalanced datasets the dataset with 25000 samples is reduced to 10000 samples in section 5.4. All of the 5000 “True” samples are taken as well as additional 5000 “False” samples. The other 15000 “False” samples are dropped. The resulting dataset is now balanced with 5000 samples for two classes. In addition to that only 10 words are used in total for this dataset which are evenly distributed in the new dataset between true and false labels.

Creation of dataset for 2 words 1000 samples

After some problems occurred with the training results of the previous dataset version, a new dataset was created in section 5.4 that consisted of only 2 words with 1000 samples each. This appeared to be the best way to test the binary classification problems of the model.

For this the two most common words “the” and “and” were used as shown in table 4.1. Due to the way the previous datasets were created this was achieved by using the first 2000 elements in the dataset of group A. Labels for this new dataset were assigned with “True” to “the” and “False” to “and” for binary classification.

4.5 Preparation of fake dataset

In this step a new dataset was created, that is used in section 5.7, with a different approach compared to the previously used datasets. With a list of the most common 20 words, all sub clips of the videos were searched through and the first 1000 occurrences for each of the 20 words selected. These selected videos were then copied to a new folder that then contained 20000 video clips. This was possible as the naming convention allowed direct access to the word of each sub clip. (see 4.2) The audio of all these files was extracted as done in previous steps. (see 4.1) For future reference the file names were saved in a separate list as well.

Table 4.4: most common 20 words for fake dataset creation

1. the	1000	2. and	1000	3. to	1000	4. of	1000	5. a	1000
6. in	1000	7. that	1000	8. we	1000	9. our	1000	10. for	1000
11. this	1000	12. on	1000	13. is	1000	14. have	1000	15. i	1000
16. it	1000	17. more	1000	18. are	1000	19. will	1000	20. their	1000

These 20000 video clips are then processed with Wav2Lip to create the deepfakes for the training dataset. The videos and audios are shifted by 20 throughout the process so that each word is faked 50 times with each other word. This works as the first 1000 video clips belong to the first word and the next 1000 video clips to the second word.

The result of this process was then checked to avoid any errors that reduce the effectiveness of the training data. It turned out that this was a good idea as 381 errors were found. 190 word combinations were created too many and 191 word combinations were created too few.

This was solved by deleting word combinations that were too many and by creating lists of the already used files in the creation of deepfakes as well as the list that was saved earlier in this step for future reference. A comparison between these lists showed the files that could be processed again to fill the missing word combinations. With these steps all required word combinations have been created in the correct amount.

The 20000 video clips and 20000 deepfake video clips that have been available after the previous steps, were passed to the facial landmark extraction process and the results for deepfakes and originals saved separately from each other. (see 4.3) At this stage appeared 2 errors in the extraction process as the faces were not always visible. This was solved manually by identifying 2 files that have not been used yet and adding these files into the process.

As the part of the dataset for the videos have been already prepared, the next step was the creation of the labels both for the words and for the boolean values. The previously created and saved file names were used to get a list of the corresponding words for each video. This worked without any problems as the videos and lists in this part have not been shuffled yet.

The word labels as well as the boolean labels were then one hot encoded with the functions of the sklearn.preprocessing library. This ensured that the labels were transformed from categorical values to numerical floating point values. As the input for the videos was already provided to the model as floating point values, an error occurred when different input formats such as a combination of floating point and integer values were provided. This proved to be important as both the video data as well as the word label were provided as input to the multi input model.

The dataset was then combined from eight different parts as the 20000 files were split up into parts of 5000 files.

4.6 Fixing mistakes of the previous dataset

Experiments revealed that the previously created dataset had not been properly assembled as the first eight parts were supposed to be the original videos as extracted facial landmarks in JSON-format as well as the Wav2Lip videos. Due to a copying error this turned out to be twice the original videos in 4.5. and had to be corrected.

During experimentation with the dataset of the previous step an error occured in the built in tensorflow.keras.utils.split_dataset function as the provided dataset that was supposed to be split contained more than the common (train, test) tuple. Instead the train part consisted itself of another tuple of facial landmarks and word label. It was attempted to fix this issue with a custom function to split the dataset which seemed promising at first but later kept crashing the Jupyter environment.

While investigating this issue an execution of python code in the command line revealed that during the custom function a segmentation fault occurred that caused the crash and reset of the Jupyter environment.
This was finally fixed by separating the facial landmarks, word label and boolean label from each other, splitting and shuffling them separately with the same seed. For the shuffling with the same seed tests were made to ensure that the three data parts were still consistent to each other. The separation of the dataset into three parts also had the benefit, that these could be transformed into a numpy array. This allowed for a high control of the input of the test and training data. As the x-argument of tensorflow.keras.model.fit allowed for a list of arrays this was a perfect fit for the facial landmark and word label numpy arrays.

Codeblock 4.2: Shortened training comparison

```
#shortened example for loading the dataset into the model training
#loading via numpy arrays with high control
model.fit(x=[ds_x1_train, ds_x2_train], y=ds_y_train,
          validation_data=([ds_x1_test, ds_x2_test], ds_y_test))

#loading via tuple of tensorflow dataset
model.fit(ds_train_batches, validation_data=ds_test_batches, epochs=30)
```

It is worth mentioning that the numpy arrays were not saved but the TensorFlow dataset format was still used due to its high compression. The size of the final dataset was 12.8 GB for 40.000 files with an individual file size of roughly 1 MB.

5 Results

5.1 First experiments and early problems

When the first dataset was finished after section 4.9. the first attempt of training could be started. The first model was a very basic CNN-model as shown in table 5.1 that consisted of three pairs of convolution and maxpooling layers. The data was then flattened and given to a dense layer with a sigmoid activation function. This function has the property that it maps most inputs to either close to zero or close to one which is helpful for binary classification. The final layer in this first model was a dense layer with 2 neurons that were combined with a categorical crossentropy loss function. The first attempt of the training revealed that the shape of the input did not have enough dimensions to fulfill all the requirements for the required input shape. This was solved in section 4.4. in more detail.

Table 5.1: Simple CNN-architecture

Input	479 x 90 x 1
Block 1	3x3 Conv,32
	2x2 MaxPool
Block 2	3x3 Conv, 64
	2x2 MaxPool
Block 3	3x3 Conv, 64
	FC 64
	FC 2
	Sigmoid

Problems with value error and low accuracy

After the input shape was fixed the next error that was found was a ValueError as the input was not batched before the training as the dataset had a size of 4.1 GB, that consisted of 25.000 pairs of facial landmarks and boolean values, and fit both in RAM and VRAM of the training computer. However this affected the input-shape which was (479,90,1) instead of (None, 479, 90,1). This was easily solved by batching the dataset that was in the same step also shuffled and repeated to an infinite dataset. In the second version of the model the loss function was replaced against binary_crossentropy and the optimizer to Adam instead of SGD. A first training result with a batch size of 100 and 50 steps per epoch for 1 epoch resulted in an accuracy of only 0.2122 and was therefore considered unsuccessful.

When searching for the origin of this low training result, a prediction of one input sample of the dataset revealed that the predicted result was an array ([[4.300631]], dtype=float32) which was significantly different from the training label tf.Tensor([False], shape=(1,), dtype=bool). It was assumed that there might have been a problem due to a false split of x_train and x_label in the tensor before the training. Some of the values of the dataset were sampled, converted to a list of tuples and the tuple contents checked which did not result in any noticeably errors.

The training was run again with the same setup for batch size, steps per epoch and epoch including the batching and shuffling as it was done in the previous experiment. The difference was that 100 values of the dataset batches were taken and their values separated for x_train and x_test. The training for these 100 values resulted in an accuracy of 0.7800.
The difference seemed to be in the input type which was in the first attempt a tensor of type tf.float32 and in the second attempt a tensor of type float32 but as a numpy array.

In a third attempt this resulted back to the low accuracy of 0.2026 as the dataset of tensors was used again which confirmed the previous thesis that there might be some problems in the dataset or the model.

5.2 Experimenting with different model architectures

To check if the model might be the problem a plan was made to test the training data with other well known model architectures. The first model architecture that

was considered was the VGG16. The input shape, the number of classes and the loss function were changed compared to the original VGG16 model. Created in a similar way were the modified ResNet50 and modified DenseNet121.

The modified VGG 16 model was then trained with a batch size of 128, 3 epochs, and 196 steps per epoch. The accuracy stayed below 0.2 in the first attempt of this modified model. A second attempt was started for 5 epochs with the highest result being 0.2029 accuracy. It became clear very quickly that an increase in the number of epochs would not result in better results.
A second attempt was made with sparse categorical crossentropy as the loss function and two neurons in the last dense layer. This attempt resulted after 5 epochs in an accuracy of roughly 0.8 with the highest score at 0.8013.
This made it seem as if the model might have been the problem instead of the data as scores of up to 0.8 accuracy were reachable.

The second model architecture tested was the ResNet50. The modified ResNet50 model was also trained with a batch size of 128, 5 epochs, and 157 steps per epoch. The difference to the 196 steps per epoch results in the step calculation being done on the split train data length for the ResNet50 model instead of the full data length for the VGG16 model. With a sigmoid activation function and a single neuron for the output layer the model scores an accuracy of 0.7975 as the highest value. Due to the split between training and test data the validation accuracy was also calculated and reached 0.8076. The model does not score better than the modified VGG16 model, but it was of interest that the validation accuracy was close to the accuracy as that hinted at no problem of overfitting.

The third model architecture tested was the DenseNet121. The DenseNet121 was trained with a batch size of 128, 5 epochs and 157 steps per epoch. The loss function used was binary crossentropy and the activation function softmax. During the training the peak for the accuracy was 0.7982 and the highest score of the validation accuracy was 0.8070. The validation accuracy is again close to the accuracy similar to the ResNet50 model.

5.3 Testing different functions for a modified VGG16 model

A series of tests was done with the modified VGG16 model to test different loss funtions and neurons in the last layer of the model and their impact to the accuracy of the model.

For these tests the softmax activation function and 1 neuron in the last layer were combined with either binary crossentropy, which resulted in an accuracy of 0.2, or sparse categorical crossentropy as the loss functions, that resulted in an accuracy of 0.8. Also the sigmoid activation function and 2 neurons in the last layer were combined with binary crossentropy, that resulted in a value error, and again sparse categorical crossentropy, which led to a result of 0.8.
The solve the value error for the combination of the sigmoid activation function and binary crossentropy as the loss function, the number of neurons in the last layer was reduced to 1. This resulted in an accuracy of 0.8. This combination was then kept as the default going forward with other experiments.

As several different model architectures have been trained with different activation and loss functions scoring close to 0.8, it was assumed that the model architecture is not the problem. This assumption was supported by the observation that the training accuracy did not reach significantly beyond the accuracy of 0.8 in all tested model architectures. Instead this pointed at something being not as intended with the training data.

The dataset that was being used until this point consisted of 25.000 files that were based on the most common 25 words with 1.000 files each. The first 5 words and therefore 5.000 files were labeled as "True" whereas the other 20.000 files were labeled as "False". The original idea was that the model had more data to identify and therefore to learn better when something is false. Unfortunately this led right into the problem of overfitting. 20% of the labeled data were labeled "True" and 80% were labeled "False". This were the same proportions that were found during the training in the accuracy and validation accuracy. It appeared as if the model had just predicted everything as false to reach a high accuracy. This was confirmed by a confusion matrix shown in figure 5.1.

To solve this unintended problem a new dataset was created with the first 10.000 files of the previous dataset. See section 4.4. for more details.

5.4 Experimenting on more balanced dataset

The training of the new balanced dataset with the modified VGG16 model, binary crossentropy and sigmoid activation function resulted in the accuracy being stuck at 0.5041 with the validation accuracy frozen at 0.4835. This remained mostly the same

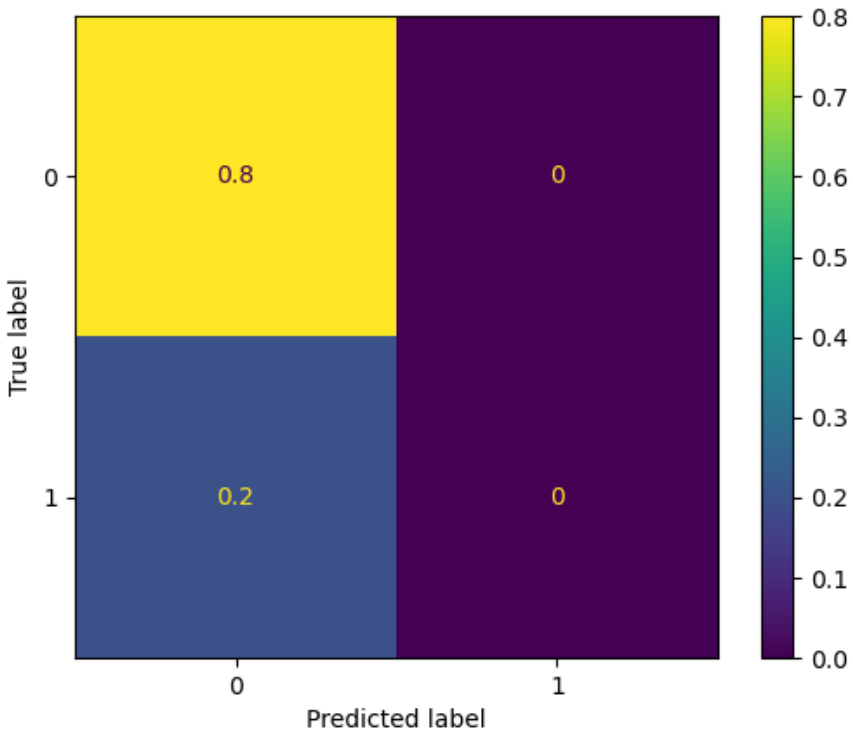

Figure 5.1: Confusion matrix of modified VGG16 model

with very minor changes in the accuracy, that was still stuck around 0.5, even with multiple restarts of the training process and up to 20 epochs.

As the model architecture had been tested before and seemed to not be the problem, the cause for the accuracy problems with the new balanced dataset were assumed to be still in the dataset, even though the main issue of unbalanced classes was fixed. This was attributed to errors in the creation process of the dataset as well as too few differences between the facial landmarks due to the grouping of multiple words into one class. Another possible error was the fact that the word label was represented as a binary row at the top of the facial landmark array. Due to the computations in the convolutional layers, this first line was calculated into the other data rows and therefore likely unrecognizable at the end of the layers when the binary decision is made.

In order to solve this without the need to recreate the entire dataset from the beginning, the dataset was further reduced to only 2.000 files. See section 4.4 for further details.

Experiments on small dataset

During the training of this shortened dataset the expectation was that the accuracy should be much higher compared to the previous tests. However the accuracy had the same problems and was also stuck at the same precision at around 0.5.
This proved that the grouping of multiple words into one class could not be the cause of the current problem as there were now only 2 words involved. Before attempting to change the dataset again, an attempt was made to use a very simple CNN model with the idea in mind that the models might have been too complex for the problem. In addition making another test with a simpler model was still cheaper compared to the overhaul of the entire dataset creation process.

This simple model from table 5.1 was adjusted to the input shape of the current dataset, the number of neurons at the end was changed to one and the sigmoid and binary crossentropy functions were used again. As the dataset and model had both been simplified over the past experiments this had a positive effect on the training times. Therefore the first attempt was done with 100 epochs and had an accuracy of 0.5006 and a validation accuracy of 0.6025 after the first epoch. This quickly improved over the next epochs and reached up to an acurracy of 0.8131 and a validation accuracy of 0.7500 after 100 epochs. After training for additional 200 epochs the accuracy reached a peak of 0.9381 and a validation accuracy of 0.8700 at epoch 294.
Compared to the previous problem that was a success. It showed that a simpler model was able to score a higher accuracy than a more complex model did.

Table 5.2: modified VGG-16 model

Input	479 x 90 x 1
Block 1	2x(3x3 Conv,64) + 2x2 MaxPool
Block 2	2x(3x3 Conv,128) + 2x2 MaxPool
Block 3	3x(3x3 Conv,256) + 2x2 MaxPool
Block 4	3x(3x3 Conv,256) + 2x2 MaxPool
Block 5	3x(3x3 Conv,512) + 2x2 MaxPool
Block 6	3x(3x3 Conv,512) + 2x2 MaxPool
	Flatten
Block 7	2x(FC 4096)
	FC 1
	Sigmoid

To follow up on the new findings the modified VGG16 model was simplified and the number of convolutional and maxpooling layers reduced. In particular this effected blocks 4-6 of table 5.2. The training was then done over 100 epochs that resulted in an accuracy of 0.8600 and a validation accuracy of 0.7925. This was already better than the accuracy of the simpler model in the previous experiment when compared after 100 epochs.
The difference between the accuracy and the validation accuracy for both this and the previous experiment showed that the model had been adapted better to the training data compared to the validation data in both experiments. This hints at the existence of overfitting in the model.
Overfitting might have also been linked to the learning process of the more complex modified VGG16 model that either adapted its neuron connections too well or too slow in the model training and was therefore unable to reach an accuracy higher than close to 0.5.

One additional finding during the training process of the modified and simplified VGG16 model was that the training process was in some experiments stuck at an accuracy under 0.5. Even with additional epochs no progress in the calculated loss and validation loss could be measured. If the same model was trained again with a clean restart, which means that there were no information of the previous training progress available, this continued to be the case in some instances but in others the model was able to overcome this barrier and reach better scores.
When a training run that successfully overcame this barrier and was allowed to train for a total of 500 epochs the best validation accuracy of 0.9275 and an accuracy of 0.9825 was reached after 468 epochs.

5.5 Experiments to solve overfitting

To investigate this behavior more closely and to also attempt to prevent overfitting through the introduction of dropout layers or other forms of regulization to the kernel, the activity and the bias, a series of experiments was started where each of these measures was added incrementally.
The basis for these experiments was the modified VGG16 model that was not simplified yet as shown in table 5.2. Each experiment was done over 20 epochs.
In the first experiment of this series a dropout layer was added after the third max pooling layer. After 20 Epochs the accuracy did not rise above 0.5025 and the validation

accuracy was stuck at 0.4900.
In the second experiment a second dropout layer was added bevor the flatten layer towards the end of the model. This did not provide any significant increases but the accuracy reached 0.5075 and the validation accuracy 0.4700.
With additional dropout layers after every max pooling layer the accuracy reached 0.5044 and the validation accuracy 0.4825. Dropout layers alone showed no major improvements of the training accuracy. Also the training progress generally peaked between one and three epochs and didn't improve afterwards.
The experiments were then continued with an added L2-bias regularizer after every dense and convolutional layer. This spread the progress during the training over 10 epochs instead of up to three but the accuracy of 0.5013 and validation accuracy of 0.4950 cannot be described as a real improvement.
For the next experiment L1 and L2 kernel regularizer were added after the convolutional and dense layers. This changed The accuracy to 0.5069 and the validation accuracy to 0.4725. The progress during the training halted again after the third epoch and didn't change further after that.
As a next attempt both dense layers with 4096 neurons at the end of the model in block 7 were dropped. This reduced the number of trainable parameters from 90 million to 14 million. The accuracy of 0.5031 and validation accuracy of 0.4875 showed no significant improvements and froze after the second epoch.
With an added L2 activity regularizer after every convolutional and dense layer the training progress didn't freeze after a few epochs anymore but the accuracy of 0.5013 and validation accuracy of 0.4950 cannot be described as being better.
As dropout layers and L1 and L2 regularizers were added throughout the model without any improvements to the model's accuracy, the next experiments focused again on simplifying the model architecture.
In a first step the last block of convolutional and max pooling layers was dropped which was block 6 in table 5.2. This resulted in a mostly unchanged accuracy of 0.5050 and a validation accuracy of 0.4800.
Dropping again the last block or block 5 of convolutional and max pooling layers resulted in 0.4988 accuracy and 0.5050 validation accuracy.
Only after the block 4 or the last block of convolutional and max pooling layers was dropped again, which resulted in a reduction by a total of 3 blocks of the model, the results started to improve significantly. After 20 epochs the accuracy reached 0.6619 and the validation accuracy 0.6825.
This model's training was then continued for another 80 epochs to a total of 100 epochs and reached an accuracy of 0.8225 and a validation accuracy of 0.7525.

Therefore the overfitting problem was successfully reduced, even though the model architecture had to be simplified again.

Table 5.3: simplified VGG-16 model

Input	479 x 90 x 1
Block 1	2x(3x3 Conv,64) + 2x2 MaxPool + Dropout
Block 2	2x(3x3 Conv,128) + 2x2 MaxPool + Dropout
Block 3	3x(3x3 Conv,256) + 2x2 MaxPool + Dropout
	Flatten
	FC 1
	Sigmoid

5.6 Scaling experiments up to balanced dataset

As the previous experiments in sections 5.4. and 5.5. were done on the dataset with 2.000 files, the findings of those experiments were tried to be scaled up to the larger balanced dataset with 10.000 files.

The same modified VGG16 model with 3 convolutional blocks was used. During training with a batch size of 128, a train/test split of 80/20 and training for 20 epochs with 63 steps per epoch, the model reached an accuracy of 0.5745 and a validation accuracy of 0.5890. The model did not improve much after the fifth epoch and the training was then continued for another 80 epochs to see if this plateau could be overcome. This resulted in the model reaching an accuracy of 0.5935 and a validation accuracy of 0.5955 which meant that the difference between the predictions on the train and test data were very close to each other. The measures against overfitting as experimented in the previous section were doing a good job of preventing overfitting over the first 100 epochs.
As the current result was an improvement compared to previous results the training is continued for another 400 epochs to see how the model behaves if the likelihood of overfitting increases as the model adapts more and more to the training data over the numerous additional epochs. The training resulted in an accuracy of 0.9484 and a validation accuracy of 0.5755. This is the greatest difference in all the experiments that have been run up to this point. The best validation accuracy in this training actually occurred already in epoch 146 with a validation accuracy of 0.6220 and an accuracy of 0.6259.

Based on these findings the current model was able to predict these ten words in the dataset correctly in 62% of occurrences when accounting for both seen and unseen data.

However the current model could still run into a frozen training progression as a rerun of the same model in a clean environment showed. The training progress froze after three epochs for the accuracy and after 2 epochs for the validation accuracy. This is a phenomenon that occurred in several of the experiments based on the modified VGG16 model.

5.7 Final experiments with deepfake dataset

As all of the datasets up to this point have always been splits between different sets of original facial landmarks, that were labeled differently into “True” and “False” parts to test different approaches, this step would be to finally include actual deepfakes in the dataset. This was done by creating a new balanced dataset of 40.000 files that included 20.000 original data entries as well as 20.000 deepfakes as described in section 4.5. The dataset was created for a new model that was designed to receive multiple inputs of facial landmarks and word labels. The concept of this is shown in figure 5.2. The shown model has multiple convolutional layers as well as multiple iterations of the layers in the dotted box. The separate inputs for the facial landmarks and the word labels received separate computations through different layers before the flattened results were concatenated together. This changed the way the model needed to be built from a sequential model to a model that was built with the help of the functional API of Keras. Additionally this model received tuneable parameters with the utilization of keras tuners. With these tuners in place, the optimal parameters of the model were searched during different trials of the model training phase.

After three trials of 20 epochs on a portion of 500 samples of the dataset were run, the validation data was at 0.9802 which was the highest any model had ever reached before. This was problematic as tests for the evaluation of the best epoch revealed that the model reached an accuracy of 0.6409 and a validation accuracy of 0.9802 after the first epoch. It was clear at this point that something had to have gone wrong either with the split of the training and test data or with the creation of the entire dataset as the validation accuracy was already close to 100% right out of the box. This might have been a problem of the sampling of 500 samples but the problem persisted even with different shuffling seeds of the dataset.

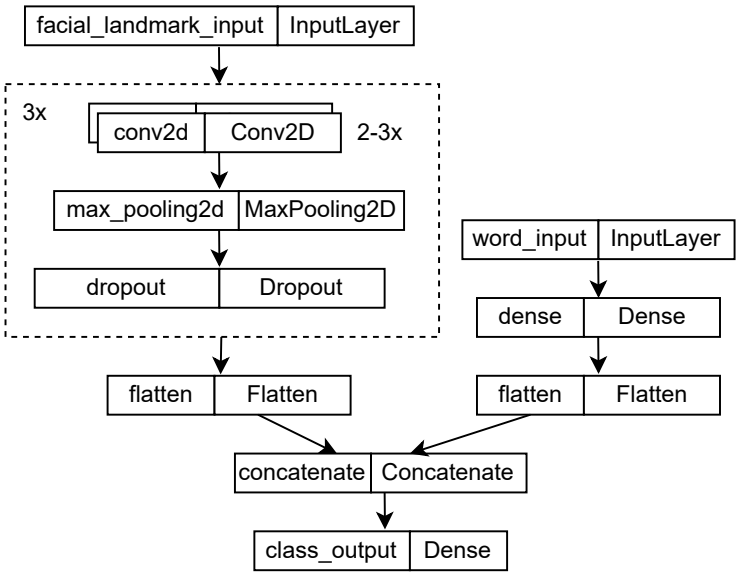

Figure 5.2: Multi input model

At this point the model could not be trained on more than 500 samples at the same time as the splitting of the model caused errors. The source of these errors lied in the custom splitting function that caused a memory leak. These issues were fixed in section 4.6. and the dataset now contained both original as well as deepfake entries.

For the experiments the dataset had to be split into three parts before it could be passed to the model. These parts were created from the elements that were stored in the dataset which were added to three different lists and then converted into a numpy array for the required input control into the multi input model as shown in codeblock on page 32. The Keras tuner were now also configured to be able to train the dropout values and kernel regularizers. A total of 7 parameters were trainable in the current setup as shown in the codeblock below.

Codeblock 5.1: Keras tuner parameter

```
1        #Tune the kernel_regularizer for the first 15 runs without shuffle
2        hp_kernel_regularizer_l1 = hp.Choice('kernel_regularizer_l1',
3            values=[1e-3, 1e-4, 1e-5], default=0.0001)
4        hp_kernel_regularizer_l2 = hp.Choice('kernel_regularizer_l2',
5            values=[1e-3, 1e-4, 1e-5], default=0.0001)
```

```
        #Tune the activity_regularizer
        hp_activity_regularizer_l2 = hp.Choice('activity_regularizer_l2',
            values=[1e-3, 1e-4, 1e-5], default=0.001)
        #Tune the bias_regularizer
        hp_bias_regularizer_l2 = hp.Choice('bias_regularizer_l2',
            values=[1e-3, 1e-4, 1e-5], default=1e-05)
        #Tune the Dropout-Layer
        hp_dropout = hp.Choice('dropout', values=[0.15, 0.2, 0.25], default=0.2)
        #Tune the learning rate for the optimizer
        hp_learning_rate = hp.Choice('learning_rate',
            values=[1e-2, 1e-3, 1e-4], default=0.0001)
        #Tune the batch size for the optimizer
        batch_size = hp.Int("batch_size", 32, 128, step=32, default=96)
```

The search for the ideal parameters with the Keras tuner was run for 15 trials which resulted in the highest validation accuracy of 0.7859. The model was then trained with early stopping to get the best epoch. After 9 epochs the model training found the best validation accuracy which was at 0.8664 with an accuracy of 0.8725. A confusion matrix is shown in figure 5.3.

The search with the Keras tuner was then run again for another 100 trials. After the completion of these trials the model was trained for a total of 150 epochs with the highest result being a validation accuracy of 0.8622 and an accuracy of 0.8720 that were both slightly lower compared to the previous experiment.

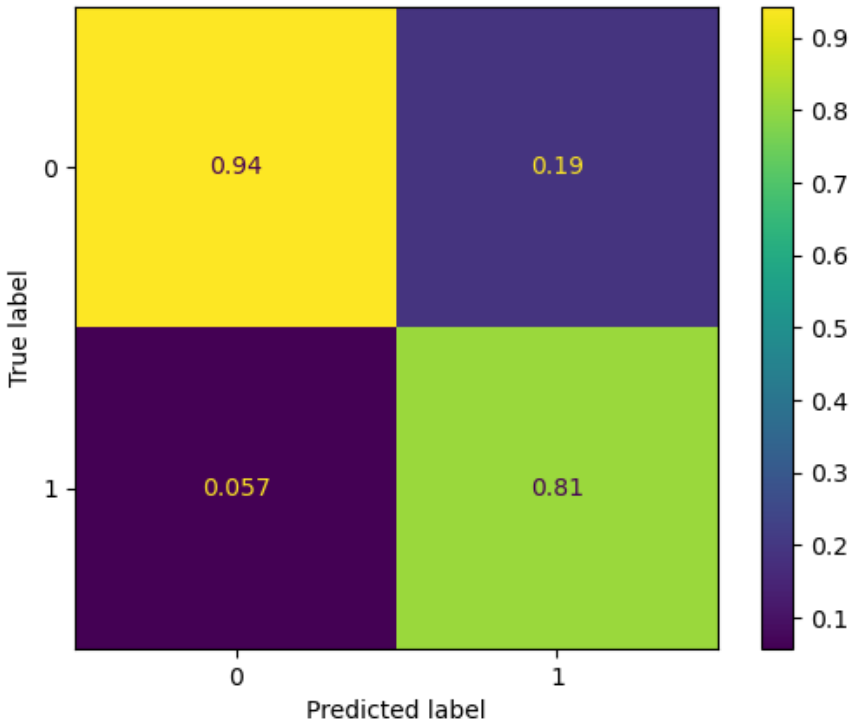

Figure 5.3: Confusion matrix for final run

6 Discussion

This section presents and discusses the results of the experiments according to the research hypotheses. In the main research hypothesis it was proposed that the facial landmark motion during a word-specific pronunciation might be a way to identify an original video from a fake one if the classifier is only trained on the input data of one specific person. To test this hypothesis training data for Barack Obama was collected and a CNN model trained on this data. The evaluation of the model proved that it is possible to train a classifier for the word pronunciation of one specific person.

It was found that the model is able to predict in 86.64% of cases across all of the involved most common 20 words, that were taken from the presidential weekly addresses during the presidency of Barack Obama from 2009 to 2017, whether the word was pronounced correctly based on the specific facial landmark motions of the target. In addition the model is not limited to a specific video resolution as the extraction of facial features has been successful with different video resolutions which confirmed another hypothesis.

Throughout the experiments several model architectures have been tested. However it turned out that the architecture can become too complex for meaningful results after the first 3 convolutional blocks. In addition a multi input model was needed to combine the input of facial landmark and words to not loose information of the word during computations.

The analysis shows that it is possible to use person specific facial landmark motions to detect the correct pronunciation of a target person and thereby being able to identify a deepfake video. This finding is consistent with the results of previous research of (S. Agarwal et al., 2022). However the accuracy of the model trained in this project stayed significantly below the results of other research like (S. Agarwal et al., 2022) while also focusing on one specific person compared to a general approach as shown in (S. Agarwal, El-Gaaly et al., 2020) or (Y. Huang et al., 2020).

The results contribute insights with regard to detection measures that prove whether a video is actually depicting one specific individual or someone else. This detection

method is harder to evade as the trained model is person specific instead of being able to generalize well for various different persons.

However, some limitations should be acknowledged for this thesis. The experiments were limited on training data that had only one face in it that faced the camera directly. The model also hasn't been tested against any videos in the wild and therefore might not fully consider the problems that can appear in real situations. The transcription model used was trained with a focus on English and might behave different in other languages. In addition the prediction of the word time stamps wasn't native to the model and were slightly off and were buffered. Hence, caution should be taken with generalizing the findings and applying them to real-life situations.

Summary

In summary, this thesis proved that it is possible to create a deep learning model that can detect a deepfake based on the facial movements during the pronunciation of a select number of words for one specific person only. It detailed the process of how a video needs to be prepared in order to be added to a dataset that can be used in the training process, highlighted some problems to avoid on the way and showed what model architectures of convolutional neural networks can be successfully trained on the processed data of facial landmarks. Even though the final model cannot compete at the same level as current state of the art deepfake detection models, this approach broke out of the cycle of generalized GAN training for detection purposes. Due to this the model is less likely to loose effectiveness against unseen generative models as it is both person specific and resistant to changes in video resolutions which increases the longevity of the relevance of this model compared to other more generalized detection models.

Future work

While this thesis highlights useful insights about the potential of detecting a video based on person specific word-pronunciation, future research can extend this research in several ways. Future experiments could attempt to improve the accuracy of the current model architecture by reducing the facial landmark features to the most prominent ones. This could allow the model architecture to become deeper and more complex as

well as greatly decrease the training times of the model as the input shape and number of parameters would be decreased significantly.
Other experiments might focus on the minimum number of samples that are required per word to be detected with a certain minimum in the accuracy. Research in this direction might make this approach of deepfake detection more accessible to the public and interested individuals who are seeking a personal deepfake detection model.
Future work could also focus on different languages. This might be interesting as different languages vary in their vocalizations and therefore different patterns of facial landmark motions might exist. This could also be coupled with an analysis of the most prominent features. To a likely smaller degree this could also be based on different accents of the same language.
As the experiments relied heavily on accurate time stamps for each word this might also be an option for future experiments. Less noise due to more accurate predictions due to improvements of the same model or a different one could improve the accuracy of the model significantly.
Another limitation in this thesis was the lack of testing the trained model against videos in the wild. An analysis on videos such as other videos of the target person that might not face the camera directly, deepfake videos created through other means than used in this thesis or also including video footage of impersonators in the analysis could build on the findings of this thesis.

References

Agarwal, R. (2023). *Complete Guide to the Adam Optimization Algorithm*. Zugriff 4. Mai 2024 unter https://builtin.com/machine-learning/adam-optimization

Agarwal, S., El-Gaaly, T., Farid, H., & Lim, S. (2020). Detecting Deep-Fake Videos from Appearance and Behavior. *CoRR, abs/2004.14491*. https://arxiv.org/abs/2004.14491

Agarwal, S., Farid, H., Fried, O., & Agrawala, M. (2020). Detecting Deep-Fake Videos from Phoneme-Viseme Mismatches. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2814–2822. https://doi.org/10.1109/CVPRW50498.2020.00338

Agarwal, S., Hu, L., Ng, E., Darrell, T., Li, H., & Rohrbach, A. (2022). Watch Those Words: Video Falsification Detection Using Word-Conditioned Facial Motion. https://arxiv.org/abs/2112.10936

Ajaka, N., Kessler, G., & Samuels, E. (2019). *Seeing istn't believing: The Fact Checker's guide to manipulated video*. Zugriff 17. April 2024 unter https://www.washingtonpost.com/graphics/2019/politics/fact-checker/manipulated-video-guide/

Amberscript. (2023). *Dubbing: What Is It and How Does It Work?* Zugriff 17. April 2024 unter https://www.amberscript.com/en/blog/what-is-dubbing-and-how-does-it-work/

AWS. (2024a). *What is a GAN?* Zugriff 24. April 2024 unter https://aws.amazon.com/what-is/gan/#:~:text=A%20generative%20adversarial%20network%20system,is%20fake%20or%20real%20data

AWS. (2024b). *What is overfitting?* Zugriff 23. April 2024 unter https://aws.amazon.com/what-is/overfitting/?nc1=h_ls

Baeldung. (2023). *Differences Between Gradient*. Zugriff 4. Mai 2024 unter https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch

Brannon, W., Virkar, Y., & Thompson, B. (2023). Dubbing in Practice: A Large Scale Study of Human Localization With Insights for Automatic Dubbing. *Transactions of the Association for Computational Linguistics*, *11*, 419–435. https://doi.org/10.1162/tacl_a_00551

Carlini, N., & Farid, H. (2020). Evading Deepfake-Image Detectors with White- and Black-Box Attacks. *CoRR*, *abs/2004.00622*. https://arxiv.org/abs/2004.00622

Chakravarthy, S. (2020). *Loss Functions in Deep Learning Models*. Zugriff 4. Mai 2024 unter https://srinivas-yeeda.medium.com/loss-functions-in-deep-learning-models-129866be93e

Datagen. (2024a). *ResNet-50: The Basics and a Quick Tutorial*. Zugriff 3. Mai 2024 unter https://datagen.tech/guides/computer-vision/resnet-50/

Datagen. (2024b). *Understanding VGG16: Concepts, Architecture, and Performance*. Zugriff 3. Mai 2024 unter https://datagen.tech/guides/computer-vision/vgg16/

deepai.org. (2024a). *Adam*. Zugriff 5. Mai 2024 unter https://deepai.org/machine-learning-glossary-and-terms/adam-machine-learning

deepai.org. (2024b). *Feed Forward Neural Network*. Zugriff 4. Mai 2024 unter https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

Dhumne, S. (2023). *Elastic Net Regression detailed guide !* Zugriff 23. April 2024 unter https://medium.com/@shruti.dhumne/elastic-net-regression-detailed-guide-99dce30b8e6e

Doshi, S. (2019). *Various Optimization Algorithms For Training Neural Network*. Zugriff 4. Mai 2024 unter https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2021). A Comprehensive Survey and Performance Analysis of Activation Functions in Deep Learning. *CoRR*, *abs/2109.14545*. https://arxiv.org/abs/2109.14545

Franco, F. (2024). *The Softmax Activation Function*. Zugriff 5. Mai 2024 unter https://medium.com/@francescofranco_39234/the-softmax-activation-function-137c321461ca

Ganesh, P. (2019). *Types of Convolution Kernels : Simplified*. Zugriff 24. April 2024 unter https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37

Giorgino, T. (2009). Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package. *Journal of Statistical Software*, *31*(7). https://doi.org/10.18637/jss.v031.i07

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks. https://arxiv.org/abs/1406.2661

Google. (2022). *Overview of GAN Structure*. Zugriff 16. Mai 2024 unter https://developers.google.com/machine-learning/gan/gan_structure

Google. (2024a). *About Keras 3*. Zugriff 24. April 2024 unter https://keras.io/about/

Google. (2024b). *Face landmark detection guide*. Zugriff 24. April 2024 unter https://developers.google.com/mediapipe/solutions/vision/face_landmarker

Google. (2024c). *Keras: Simple. Flexible. Powerful.* Zugriff 24. April 2024 unter https://keras.io/

Google. (2024d). *KerasTuner*. Zugriff 24. April 2024 unter https://keras.io/keras_tuner/

Google. (2024e). *On-device machine learning for everyone*. Zugriff 24. April 2024 unter https://developers.google.com/mediapipe

Guarnera, L., Giudice, O., & Battiato, S. (2020). DeepFake Detection by Analyzing Convolutional Traces. *CoRR*, *abs/2004.10448*. https://arxiv.org/abs/2004.10448

Gupta, P. (2017). *Regularization in Machine Learning*. Zugriff 23. April 2024 unter https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *CoRR*, *abs/1512.03385*. http://arxiv.org/abs/1512.03385

Hernandez-Ortega, J., Tolosana, R., Fiérrez, J., & Morales, A. (2020). DeepFakesON-Phys: DeepFakes Detection based on Heart Rate Estimation. *CoRR*, *abs/2010.00400*. https://arxiv.org/abs/2010.00400

Huang, G., Liu, Z., & Weinberger, K. Q. (2016). Densely Connected Convolutional Networks. *CoRR*, *abs/1608.06993*. http://arxiv.org/abs/1608.06993

Huang, Y., Juefei-Xu, F., Wang, R., Xie, X., Ma, L., Li, J., Miao, W., Liu, Y., & Pu, G. (2020). FakeLocator: Robust Localization of GAN-Based Face Manipulations via Semantic Segmentation Networks with Bells and Whistles. *CoRR*, *abs/2001.09598*. https://arxiv.org/abs/2001.09598

IBM. (2021). *Dubbing*. Zugriff 18. April 2024 unter https://www.ibm.com/de-de/topics/neural-networks

IBM. (2024). *What is overfitting?* Zugriff 23. April 2024 unter https://www.ibm.com/topics/overfitting

Islam, R., Mazumdar, S., & Islam, R. (2024). An Experiment on Feature Selection using Logistic Regression. https://arxiv.org/abs/2402.00201

Iuhaniwal, V. (2019). *Forward propagation in neural networks: Simplified math and code version*. Zugriff 4. Mai 2024 unter https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250

Jain, V. (2019). *Everything you need to know about Activation Functions in Deep learning models*. Zugriff 5. Mai 2024 unter https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

Javatpoint. (2017). *Regularization in Machine Learning*. Zugriff 23. April 2024 unter https://www.javatpoint.com/regularization-in-machine-learning

Jiang, L., Dai, B., Wu, W., & Loy, C. C. (2020). Focal Frequency Loss for Generative Models. *CoRR*, *abs/2012.12821*. https://arxiv.org/abs/2012.12821

Juefei-Xu, F., Wang, R., Huang, Y., Guo, Q., Ma, L., & Liu, Y. (2021). Countering Malicious DeepFakes: Survey, Battleground, and Horizon. *CoRR*, *abs/2103.00218*. https://arxiv.org/abs/2103.00218

Jung, S., & Keuper, M. (2020). Spectral Distribution Aware Image Generation. *CoRR*, *abs/2012.03110*. https://arxiv.org/abs/2012.03110

Kaste, J. (2023). *Künstliche neuronale Netzwerke zur adaptiven Fahrdynamikregelung* (1. Aufl.). Springer Vieweg.

Koopman, M., Macarulla Rodriguez, A., & Geradts, Z. (2018). Detection of Deepfake Video Manipulation.

Krishnamurthy, B. (2024). *An Introduction to the ReLU Activation Function*. Zugriff 5. Mai 2024 unter https://builtin.com/machine-learning/relu-activation-function

Kundu, N. (2023). *Exploring ResNet50: An In-Depth Look at the Model Architecture and Code Implementation*. Zugriff 3. Mai 2024 unter https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f

Lang, N. (2023). *Was ist die Softmax-Funktion?* Zugriff 5. Mai 2024 unter https://databasecamp.de/ki/softmax

Li, H., Li, B., Tan, S., & Huang, J. (2018). Detection of Deep Network Generated Images Using Disparities in Color Components. *CoRR*, *abs/1808.07276*. http://arxiv.org/abs/1808.07276

Li, Y., Chang, M., & Lyu, S. (2018). In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking. *CoRR*, *abs/1806.02877*. http://arxiv.org/abs/1806.02877

Li, Y., & Lyu, S. (2018). Exposing DeepFake Videos By Detecting Face Warping Artifacts. *CoRR*, *abs/1811.00656*. http://arxiv.org/abs/1811.00656

Lin, T. (2023). *Forward Propagation: The Neural Network Predictions*. Zugriff 3. Mai 2024 unter https://medium.com/@chuntcdj/forward-propagation-the-neural-network-predictions-36cdd1a5306e

Louradour, J. (2023). whisper-timestamped. https://arxiv.org/abs/2212.04356

Mahajan, P. (2020). *Fully Connected vs Convolutional Neural Networks*. Zugriff 8. Mai 2024 unter https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5

Malhotra, A. (2018). *Tutorial on Feedforward Neural Network — Part 1*. Zugriff 4. Mai 2024 unter https://medium.com/@akankshamalhotra24/tutorial-on-feedforward-neural-network-part-1-659eeff574c3

McCloskey, S., & Albright, M. (2018). Detecting GAN-generated Imagery using Color Cues. *CoRR, abs/1812.08247*. http://arxiv.org/abs/1812.08247

Mukherjee, S. (2022). *The Annotated ResNet-50*. Zugriff 3. Mai 2024 unter https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758

Murel, E. K., Jacob Ph.D. (2023). *What is regularization?* Zugriff 23. April 2024 unter https://www.ibm.com/topics/regularization

Neves, J., Tolosana, R., Vera-Rodríguez, R., Lopes, V., & Proença, H. (2019). Real or Fake? Spoofing State-Of-The-Art Face Synthesis Detection Systems. *CoRR, abs/1911.05351*. http://arxiv.org/abs/1911.05351

NocodingAI. (2023). *DenseNet*. Zugriff 3. Mai 2024 unter https://medium.com/nocoding-ai/densenet121-760df192f12d

Nvidia. (2024). *TensorFlow*. Zugriff 6. Mai 2024 unter https://www.nvidia.com/en-us/glossary/tensorflow/

OpenAI. (2022). *Introducing Whisper*. Zugriff 24. April 2024 unter https://openai.com/research/whisper

Oppermann, A. (2024a). *Activation Functions in Deep Learning: Sigmoid, tanh, ReLU*. Zugriff 5. Mai 2024 unter https://artemoppermann.com/activation-functions-in-deep-learning-sigmoid-tanh-relu/

Oppermann, A. (2024b). *Backpropagation: Training der neuronalen Netzwerke*. Zugriff 4. Mai 2024 unter https://artemoppermann.com/de/training-der-kuenstlichen-neuronalen-netze/

Oppermann, A. (2024c). *Optimierung in Deep Learning: AdaGrad, RMSProp, ADAM*. Zugriff 5. Mai 2024 unter https://artemoppermann.com/de/optimierung-in-deep-learning-adagrad-rmsprop-adam/

Prajwal, K. R., Mukhopadhyay, R., Namboodiri, V. P., & Jawahar, C. (2020). A Lip Sync Expert Is All You Need for Speech to Lip Generation In the Wild. https://doi.org/10.1145/3394171.3413532

Pykes, K. (2023). *Fighting Overfitting With L1 or L2 Regularization: Which One Is Better?* Zugriff 23. April 2024 unter https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization

PyTorch. (2024). *PyTorch*. Zugriff 16. Mai 2024 unter pytorch.org

Qi, H., Guo, Q., Juefei-Xu, F., Xie, X., Ma, L., Feng, W., Liu, Y., & Zhao, J. (2020). DeepRhythm: Exposing DeepFakes with Attentional Visual Heartbeat Rhythms. *CoRR, abs/2006.07634*. https://arxiv.org/abs/2006.07634

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. https://arxiv.org/abs/2212.04356

Raina, A., & Arora, V. (2022). SyncNet: Using Causal Convolutions and Correlating Objective for Time Delay Estimation in Audio Signals. https://arxiv.org/pdf/2203.14639

Sharma, P. (2023). *What is momentum in Machine Learning?* Zugriff 4. Mai 2024 unter https://www.tutorialspoint.com/what-is-momentum-in-machine-learning

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/abs/1409.1556

Skansi, S. (2018). *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence* (1. Aufl.). Springer. https://doi.org/https://link.springer.com/book/10.1007/978-3-319-73004-2

Tolosana, R., Vera-Rodríguez, R., Fiérrez, J., Morales, A., & Ortega-Garcia, J. (2020). DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection. *CoRR*, *abs/2001.00179*. http://arxiv.org/abs/2001.00179

Topper, N. (2023). *Sigmoid Activation Function: An Introduction*. Zugriff 5. Mai 2024 unter https://builtin.com/machine-learning/sigmoid-activation-function

Vungarala, S. K. (2023). *Stochastic gradient descent vs Gradient descent: Exploring the differences*. Zugriff 4. Mai 2024 unter https://medium.com/@seshu8hachi/stochastic-gradient-descent-vs-gradient-descent-exploring-the-differences-9c29698b3a9b

Wang, G., Zhou, J., & Wu, Y. (2020). Exposing Deep-faked Videos by Anomalous Co-motion Pattern Detection. *CoRR*, *abs/2008.04848*. https://arxiv.org/abs/2008.04848

Wang, S., Wang, O., Zhang, R., Owens, A., & Efros, A. A. (2019). CNN-generated images are surprisingly easy to spot... for now. *CoRR*, *abs/1912.11035*. http://arxiv.org/abs/1912.11035

Yathish, V. (2022). *Loss Functions and Their Use In Neural Networks*. Zugriff 4. Mai 2024 unter https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9

Yu, N., Davis, L., & Fritz, M. (2018). Attributing Fake Images to GANs: Analyzing Fingerprints in Generated Images. *CoRR*, *abs/1811.08180*. http://arxiv.org/abs/1811.08180

Zhang, X., Karaman, S., & Chang, S. (2019). Detecting and Simulating Artifacts in GAN Fake Images. *CoRR*, *abs/1907.06515*. http://arxiv.org/abs/1907.06515

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel

Deepfake detection based on facial landmark motion analysis during person specific pronunciation

selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Hamburg, 17. Mai 2024