Generative Adversarial Networks in Lip-Synchronized Deepfakes for Personalized Video Messages

Johan Liljegren
Pontus Nordqvist

Master's thesis
2021:E33

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Master's Thesis

Centre for Mathematical Sciences

Degree Project in Mathematics for Engineers, FMAM05

Generative Adversarial Networks in Lip-Synchronized Deepfakes for Personalized Video Messages

Authors

Johan Liljegren
johan.liljegreen@gmail.com

Pontus Nordqvist
p.nordq@gmail.com

Submitted on July 1, 2021

Supervisor

Dr. Carina Geldhauser

Lund University

Supervisor

Michael Truong

Sinch AB

Examiner

Prof. Claus Führer

Lund University

Abstract

The recent progress of deep learning has enabled more powerful frameworks to create good-quality deepfakes. Deepfakes, which are mostly known for malicious purposes, have great potential to be useful in areas such as the movie industry, education, and personalized messaging. This thesis focus on lip-synchronization, which is a part of a broader pipeline to develop personalized video messages, using deepfakes. For this application, the deep learning framework Generative Adversarial Networks (GAN), adapted to a given audio and video input, was used. The objectives were to implement a structure to perform lip-synchronization, investigate what variations of GANs excel at this task, and also how different datasets impact the results.

Three different models were investigated: firstly, the GAN architecture LipGAN was reimplemented in `Pytorch`, secondly, a GAN variation, WGAN-GP, was adapted to the LipGAN architecture, and thirdly, a novel approach that takes inspiration from both models, L1WGAN-GP, was developed and implemented. All models were trained using the dataset GRID and benchmarked by the metrics PSNR, SSIM, and FID-score. Lastly, the influence of the training dataset was tested by comparing our implementation of LipGAN with another implementation trained on another dataset, LRS2.

WGAN-GP did not converge and resulted in suspected mode collapse. For the two other models, we showed that the LipGAN implementation performed best in terms of PSNR and SSIM, whereas L1WGAN-GP performed better than LipGAN according to the FID-score. Yet, L1WGAN-GP produced samples that were polluted by artifacts. Our models trained on the GRID dataset showed bad generalization performance compared to the same model trained on LRS2. Additionally, the models trained on less amount of data were outperformed by models that were trained on the full dataset.

Finally, our results suggest that LipGAN was the best performing network, and with it we successfully managed to produce satisfying lip-synchronization.

Keywords: *Generative Adversarial Networks, GAN, Lip-Synchronization, Deepfake, Deep Learning, Autoencoder, WGAN-GP, L1WGAN-GP, Skip-Connections, FID-score.*

Acknowledgments

Firstly, we would like to thank our supervisor from LTH, Carina Geldhauser. For every contribution that you have made to our thesis in reviewing papers, weekly meetings, inviting us to AI Lund seminars, scheduling meetings with experts, and inviting us to Russian seminars.

Secondly, we would like to thank our supervisors at Sinch, Michael Truong and Simon Åkesson, for helping us with the technical aspects of the thesis. A special thanks go out to the director of ML and AI at Sinch, Pieter Buteneers for coming with clever suggestions on how to improve our thesis. Also thanks to Sinch in general for providing us with the necessary equipment and computational power to complete the thesis. We would also like to thank all the other master students at Sinch for making us never feel alone in our struggles.

We would also like to thank some people, presented in no particular order; Henning Petzka for providing insight on GANs and FID-score. Our roommates, Tim and Arvid, for enduring us working at home for 5 months. The lovely people at Delphi Pizzeria for providing us with nutritious food for the duration of the thesis. Johan's sister Sofie for proofreading our report, sorry for all the Swenglish. And most importantly to our families for their everlasting support.

Lastly, we would like to thank everyone we have been in contact with for actually replying to their emails in a pandemic.

Johan Liljegren & Pontus Nordqvist
Kämnärsrätten, June 7th, 2021

Symbols

A Audio segment from the data distribution $\mathbb{P}_r$

A' Time-unsynced audio segment from the data distribution $\mathbb{P}_r$

D Discriminator in case of classification or else critic

D^* Optimal discriminator in case of classification or else critic

d L^2 distance between two fixed embedding representations

E Fixed representation in embedding

$\mathbb{E}$ Expected value

f Generic function

G Generator

G^* Optimal Generator

g Test image

H Horizontal and vertical dimension of image

h_n Hidden layer with index n

$\mathcal{H}$ Feature map, result from the convolution operation

K Kernel matrix for CNNs

k Optimization iteration

$\boldsymbol{l}$ Labels for data

$\mathcal{L}$ Loss function

M Number of frequency channels in mel spectrogram

m Margin for contrastive loss

N Number of skip connections / size of input / size of mini-batch

P Probability

$\mathbb{P}_g$ Generated data distribution from the generator

$\mathbb{P}_r$ Real data distribution from the true data

$\mathbb{P}_z$ Input data distribution to generator

r Reference image

$\mathcal{R}$ Regularization term

S True frame from data distribution $\mathbb{P}_r$

S' True time-unsynced frame from data distribution $\mathbb{P}_r$

$\hat{S}$ Faked frame from data distribution $\mathbb{P}_g$

T Time window in mel spectrogram

t Time

W_1 Wasserstein-1 distance, also known as earth mover's distance

$\boldsymbol{x}$ Input to perceptron.

$\hat{\boldsymbol{x}}$ Output of an autoencoder with input x

$\tilde{\boldsymbol{x}}$ Input to residual block

$\mathcal{X}$ Desired distribution in generative models

$\boldsymbol{y}$ Output

$\mathcal{Z}$ Prior distribution in generative models

α Time step for the shifting of the frames

β Exponential decay rates for momentum estimates in ADAM

δ Small scaling factor in Leaky ReLU

Γ Joint distribution

η Learning rate

λ Gradient penalty coefficient

ξ Weight clipping bounds

Φ Mapping function for a residual block

φ Activation function

$\boldsymbol{\omega}$ Weights in a nerual network

Acronyms

ADAM Adaptive moments estimation, optimizer

ANN Artificial neural network

ASR automatic speech recognition

CNN Convolutional neural network

DCGAN Deep convolutional generative adversarial network

DGM Deep generative models

FFT Fast fourier transform

FID Fréchet inception distance

GAN Generative adversarial network

GPU Graphics processing unit

MFC Mel-frequency cepstrum

MFCC Mel-frequency cepstral coefficient

MLP Multi-layer perceptron

NLP Natural language processing

PSNR Peak signal-to-noise ratio

ReLU Rectified linear unit

SGD Stochastic gradient descent

SSH Secure Shell Protocol

SSIM Structural similarity index measure

TTS Text-to-speech

VAE Variational autoencoders

WGAN Wasserstein generative adversarial network

WGAN-GP Wasserstein generative adversarial network - gradient penalty

Contents

CHAPTER 1

Introduction

This thesis was carried out in collaboration with Sinch Sweden AB and was done remotely due to the Covid-19 pandemic. Access to specific hardware and other resources was granted through laptops and SSH connections. For this thesis, the workload was divided equally between the two authors. Programming tasks were done using pair programming.

1.1 Disposition

In this initial chapter, we present the problem and outline the aim of this work. The second chapter, chapter 2 Background, is about the theoretical background of the thesis. This chapter will describe the basic knowledge required to understand the concepts of deep learning. A third chapter, chapter 3 GAN, describes the mainly researched component of the thesis. Different variations of GANs and their properties will be described. After this, a chapter about the technologies and the datasets used in the project, chapter 4 Technologies & Datasets, is presented. The technologies presented concern both the software and hardware used to realize the models that were used. Following this, a chapter about the methodology used in the thesis, chapter 5 Methodology, follows. This chapter presents the pre-processing used, the implementation details and ends with an introduction to the experiments conducted. Before the results are presented, chapter 6 Metrics describes the theoretical background of the different metrics used. Following this, the chapter containing the results for the experiments conducted in this thesis, chapter 7 Results, is presented. chapter 8 Discussion & Further Work, containing a discussion about the results and limitations of the model, as well as suggestions for further work. These include extensions of this project and different approaches to similar problems in the field of deepfakes. Lastly, the thesis ends with chapter 9 Conclusion, containing all drawn conclusions for this thesis.

1.2 Problem Setting

Sinch have the ambition to launch a service that utilizes *deepfakes* to create personalized video messages. This service is intended to provide customers with an automated video message that offers a more personal feeling than traditional automated messages, such as generic text messages. Deepfake is a portmanteau of the words *deep learning* and *fake*, and denotes synthetic media that has been created by *machine learning* or *deep learning*. A deepfake often contains the altering of a person's face so that it is swapped with someone

else's face in a video.

In the service that Sinch wants to provide, the face remains the same but the audio is swapped to contain automated speech making, the video personalized for the customer. The input should be in the form of a text sentence that produces audio for an already existing video. This audio and video are then combined in a lip-synchronization procedure so that the audio matches the person in the video. The realization of this service is part of a long pipeline that would include several stages and segments. In this thesis, the focus is put on the lip-synchronization stage, see figure 1.1.

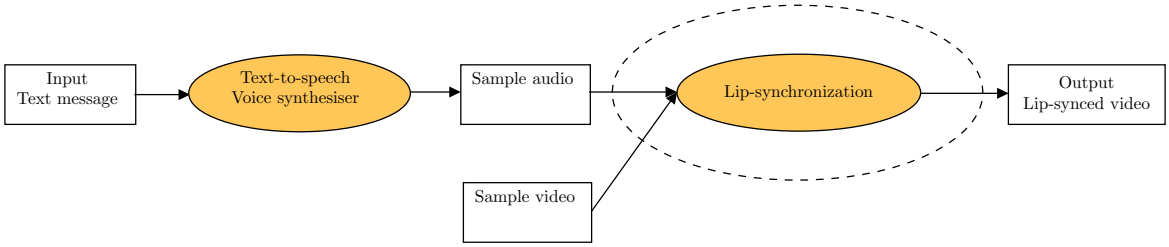


Figure 1.1: A sketch of our vision of a pipeline for a personalized video message service. This thesis focuses on lip-synchronization which is marked by the dashed circle.

This thesis aims to provide Sinch with a proof of concept that, for a given video sample and audio sample, deep learning can be used to provide satisfying lip-synchronization with low amounts of visual artifacts. This will be done with the deep learning structure *General Adversarial Network* (GAN). Several different implementations of GANs that use different loss functions and frameworks exist. An initial model will be implemented and some different variations will be tested and compared to this model. The performance of the different implementations will be evaluated by quantitative metrics and qualitative inspection of the outputs.

Further, some other interesting parameters will be investigated. The amount of data required and if this can be minimized to benefit the calculation time. Also how the chosen dataset contributes to the properties of the model performance. These goals can be formulated as:

- Given an input video and an input audio sample implement a GAN that produces realistic lip-synchronized videos, with little to no visual artifacts.
- Investigate different GAN architectures and compare these to each other through quantitative measure and qualitative assessment.
- Investigate how the amount of data affects the model's performance and output.
- Investigate the impact of the chosen dataset and how it affects the model.

1.3 Previous Work

Deepfakes and automated lip-synchronization have risen to prominence ever since deep learning became more popular due to recent improvements in hardware. A famous example is generating deepfakes of the former president of the United States, Barack Obama. This project was introduced by Suwajanakorn et al. [1] in 2017 and utilizes traditional computer vision. The example of President Obama was used due

to the extensive amount of video material available for training, which is of importance to produce a good deepfake. The author emphasizes the problem of creating a credible fake video due to the human attentiveness to details in the mouth area. An extension was made by Kumar et al. [2] later the same year, called ObamaNet, that integrates text-to-voice synthesizing to the model. It also converts the original version from a traditional computer vision model to a deep learning approach. They use the deep learning models recurrent neural networks for audio processing and a special type of autoencoders called U-Net for video processing. This was a part of the paradigm shift during the 2010s when traditional computer vision models were outperformed by deep learning.

Many recent works have focused on different techniques and approaches to produce lip-synchronizing methods. Yao et al. [3] use a phoneme search approach to target mumbling and unwanted words in videos to remove them. This is done by a large pipeline that contains deep learning structures such as a GAN. Zhou et al. [4] have a supervised learning approach to lip-synchronization. In this work, audio and video from different persons are labeled and then used to create audio- and video embeddings, which are combined by a temporal GAN.

Several implementations of a data-driven unsupervised learning approach have been made. These learn the lip movements from audio directly from the input without any additional techniques such as phoneme searching. These include the work by Chung et al. [5] that uses encoders for audio and video separately which are then concatenated to a single embedding. This embedding is then decoded by a single decoder to produce a video. Similarly, Chen et al. [6] uses encoder and decoders in the same manner but with a more advanced pipeline that uses more networks. In 2019 Prajwal et al. [7] introduced LipGAN, a network used to perform automated translation of videos. It utilizes the same data-driven approach and a GAN in combination with autoencoders.

1.4 Ethical Considerations

Deepfake is a word that often brings negative associations. It is known for being used with malicious intent to produce revenge porn videos or fake news. Though there can be good uses for deep fakes as well, such as automated translation, which was mentioned above, one should tread carefully when using techniques like this and it is up to the user to practice good ethics. Developing the science behind deepfakes can additionally contribute to techniques that detect deepfakes.

Since software like this clearly could be used for malicious intent, images or videos produced with such methods should be transparent with their origin. A solution to clearly indicate that a produced video is of deepfake origin would be to use watermarks, both visual and in the audio. This could be audio in a frequency range that is not audible for humans. Ethical concerns are raised by some authors and sound remarks have been made by, for example, Yao et al. [3] and Fried et al. [8].

CHAPTER 2

Background

In this chapter, parts of the background theory for this thesis are introduced, including machine learning, deep learning, artificial neural networks, and their components. Additionally, the specialized network classes convolutional neural network and autoencoders with skip connections are presented. Lastly, the theory surrounding audio representation for machine learning applications is introduced.

2.1 Machine Learning & Deep Learning

Machine learning is a study of artificial intelligence that can be defined as follows; "The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience" [9]. In present time, computer programs, or algorithms, are generally not near-human intelligence in learning. However, in some specific learning problems, they produce efficient and powerful solutions. These include speech recognition, data mining and image segmentation [9].

For a machine learning algorithm to learn from its experience it needs some form of feedback. Regarding this problem, there are several approaches; *supervised learning*, *reinforcement learning* and *unsupervised learning*. Supervised learning labels all inputs as either true or false, whereas in unsupervised learning there are no labels at all. In reinforcement learning, the feedback is in the form of an evaluation if the performed move was either good or bad, without conveying the correct move [10].

Several models that implement machine learning algorithms exist. One of these models is the *artificial neural network* (ANN). This implementation builds on the idea to model the biological neurons in a human's brain, and hence create a network of individual artificial neurons [10]. This is a *divide-and-conquer* approach to problems where several neurons can be combined in a network to perform one task. When several of these networks are connected in independent layers and grow dense, they form what is known as deep neural networks and the employment of these is known as deep learning [11].

2.2 Artificial Neural Networks

2.2.1 Perceptron

Artificial neural networks are, much like the human brain, made up of several small calculating units. These units can be connected both in series and in parallel to form dense networks. The most basic ANN consists of only input, one calculating unit, and output. This network is called a perceptron [9] and can be used to solve problems like linear regression. Though it is almost trivial in its structure, it can be used to show the theory of neural networks.

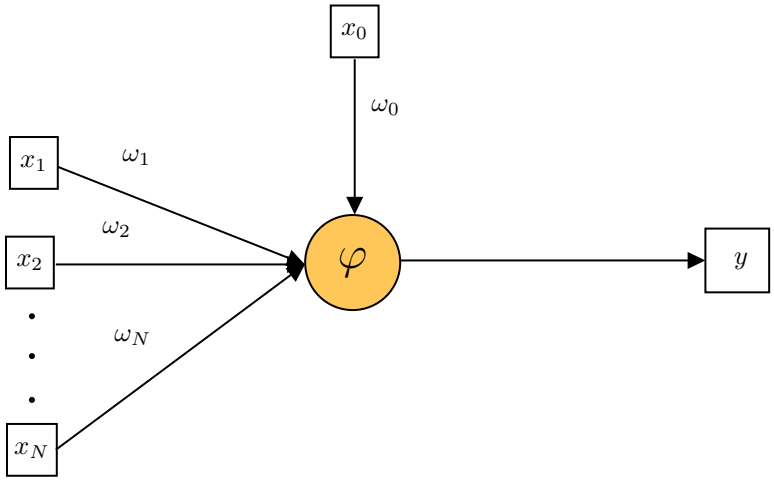


Figure 2.1: Visualization of a perceptron with input $\boldsymbol{x}$, bias x_0 and, output y.

A perceptron is shown in figure 2.1. The network can be traversed by

$$y = \varphi\left(\sum_{n=1}^{N} \omega_n x_n + \omega_0\right) \tag{2.1}$$

where $x_1, x_2, ..., x_N$ are inputs to the network and $x_0 = 1$ is a bias node, included to prevent the weights from being all zeros. The weights of an ANN are denoted as ω_n and signifies the adjustable parameters in the network. As training commences it is the weights that get an updated value and thus represent the past experience of the neural network. Lastly, φ is known as the activation function [12].

A vast number of activation functions exist and are used in different scenarios, such as the Sigmoid, Tanh, Softplus and ReLU [12]. The latter is a popular choice in deep learning [12], since it is computational efficient which is important in a dense network. The ReLU, or *Rectified Linear Unit*, is defined for a scalar input a as

$$\varphi_{\text{ReLU}}(a) = \max(0, a). \tag{2.2}$$

Further, ReLU comes in variants such as Leaky ReLU, which is defined as

$$\varphi_{\text{Leaky-ReLU}}(a) = \begin{cases} a & \forall a > 0, \\ \delta a & \text{otherwise} \end{cases} \tag{2.3}$$

where $\delta > 0$ is a small scaling factor [13]. A single perceptron is most often not desired since it is only capable to perform simple tasks. More complex problems can be solved by increasing either the number of *hidden layers* or the number of *hidden nodes*. When the number of hidden layers is larger than one, the network is called a *multi-layer perceptron* (MLP), and is shown in figure 2.2.

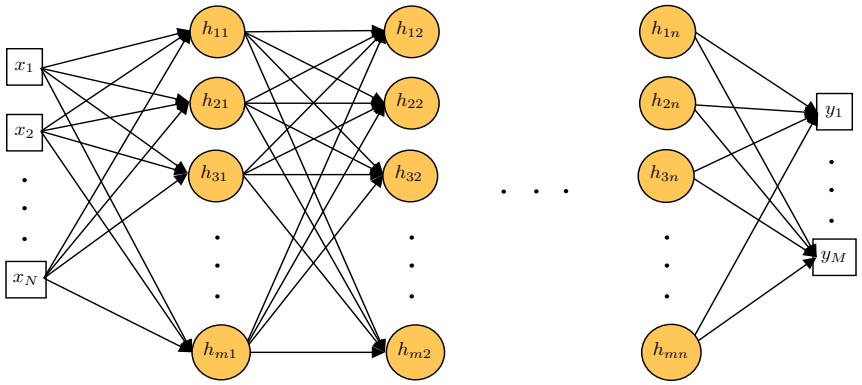


Figure 2.2: Fully connected multi-layer perceptron with m hidden nodes and n hidden layers, N inputs, and M outputs.

In theory, these networks can grow infinitely large and dense to solve increasingly complex problems which is stated by the *universal approximation theorem*, which says that a MLP can approximate any continuous function with a compact definition range [11]. This theorem accredits that MLPs in specific, and ANNs in general, are powerful tools of approximation.

2.2.2 Loss functions

As stated in section 2.1 Machine Learning & Deep Learning, a machine learning algorithm improves with experience, which requires a metric to measure its performance. Therefore, the network should be trained in a way that the updated weights perform the desired task with a better result than previous iterations. This improvement in quality is quantified by a *loss function* $\mathcal{L}$.

There are large freedoms in choosing the loss function, it can for example be a metric or a measure of some important property of special interest in the network. Though, it should be noted that in an ANN, the loss function serves as the *objective function* of an optimization problem, which is being minimized by the network. This means that the choice of the loss function is of considerable importance [12].

2.2.3 Gradient descent

When a loss function that measures the success of the network is acquired, the calculations in the neural network are an optimization problem. The objective is to find the optimal weights $\boldsymbol{\omega}^*$ by minimizing the loss function $\mathcal{L}$ for every $\boldsymbol{\omega} \in \mathbb{R}$ i.e

$$\boldsymbol{\omega}^* = \arg\min \mathcal{L}(\boldsymbol{\omega}). \tag{2.4}$$

This can be done in a manner of taking a small step $\eta > 0$ towards the steepest direction of the negative gradient of the objective function. An analogy to real life is reaching the bottom of a valley by taking small steps in the steepest downward direction. This method is called *gradient descent* and was proposed

by Cauchy [14] in 1847, as an iterative method to solve optimization problems. A step from k to $k+1$ in gradient descent, is defined as

$$\boldsymbol{\omega}^{(k+1)} = \boldsymbol{\omega}^{(k)} - \eta \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}). \tag{2.5}$$

For deep learning applications, the constant η is usually called the *learning rate*. This learning rate is often chosen as a small constant [11] and can be tuned to make training more efficient.

Gradient descent is an effective method to minimize an objective function under most circumstances. However, this approach is prone to some difficulties such as getting stuck in a *local minimum* or suffering from *vanishing gradient* problems if the slope is or close to completely flat.

2.2.4 Stochastic gradient descent

Stochastic gradient descent (SGD) is a variation of the original gradient descent that introduces a stochastic element to the algorithm. This randomization takes shape as sampling random subsets of the data into *mini-batches*. The step now becomes

$$\boldsymbol{\omega}^{(k+1)} = \boldsymbol{\omega}^{(k)} - \frac{1}{N} \sum_{n=1}^{N} \eta \nabla_{\boldsymbol{\omega}} \mathcal{L}_n(\boldsymbol{\omega}), \tag{2.6}$$

where N is the mini-batch size. When all mini-batches have been used once for updating the weights one *epoch* has been performed [12]. The mini-batch approach prevents the algorithm to get stuck in local minimums since the gradients are updated several times for each iteration instead of only once. This also makes SGD converge faster than gradient descent. While SGD remedies these problems the computational efficiency is still unsatisfactory [11].

2.2.5 Optimizers & Momentum

Gradient descent algorithms can be further enhanced in terms of convergence speed and avoiding local minimums by introducing momentum as an addition to the algorithms. These momentum-based additions, such as *AdaGrad*, *RMSProp*, *ADAM*, and others, are known as *optimizers* [12].

Momentum in gradient descent was proposed by Polyak [15] as a method to speed up the convergence of iterative methods. Momentum in gradient descent refers to the momentum in mechanics. Momentum makes use of both current and past gradients. This can be seen as not only taking the acceleration, gradient from the current step, into account but also the velocity, past gradients, when taking a step [11].

A robust optimizer is ADAM which was introduced by Kingma et al. [16] in 2014. ADAM stands for adaptive moment estimation. It utilizes running averages of past gradients and past squared gradients. These are weighted by the decay constants β_1 and β_2 which can be tuned as *hyperparameters*. The functionality of ADAM is that it learns the optimal learning rate during training. ADAM is used in several implementations of ANNs by for example Prajwal et al. [7], Chen et al. [6] and Karras et al. [17].

2.2.6 Learning

Networks like the MLP belong to a type of ANNs that are known as *feed-forward networks*. Since there is no feedback in the hidden layers or the different perceptrons at all, the only direction in the network is forward. When an ANN is traversed from input to output it performs, what is called a *forward pass*. The predictions made by the network are then used to calculate the loss $\mathcal{L}$. In supervised learning, a label is passed into the

loss to form a score that represents a performance measure of the network.

Since there is no internal feedback in the network, it lacks a way to update its weights during a forward pass, which represents the learning process. In a feed-forward network, the gradients can be calculated after the forward pass by using the *back-propagation* algorithm. This algorithm was introduced by Rumelhart et al. [18] as a computationally efficient algorithm to calculate the gradients of a neural network, which are used in the gradient descent algorithm. The weights are updated after each forward pass in the opposite direction, backward, and thus this is called a backward pass.

The forward pass in combination with the backward pass is what represents one iteration through the network. Every time the weights are updated the network has learned from experience and thus fulfills the definition of machine learning stated in section 2.1 Machine Learning & Deep Learning. A flowchart of the learning process of an ANN is displayed in figure 2.3.

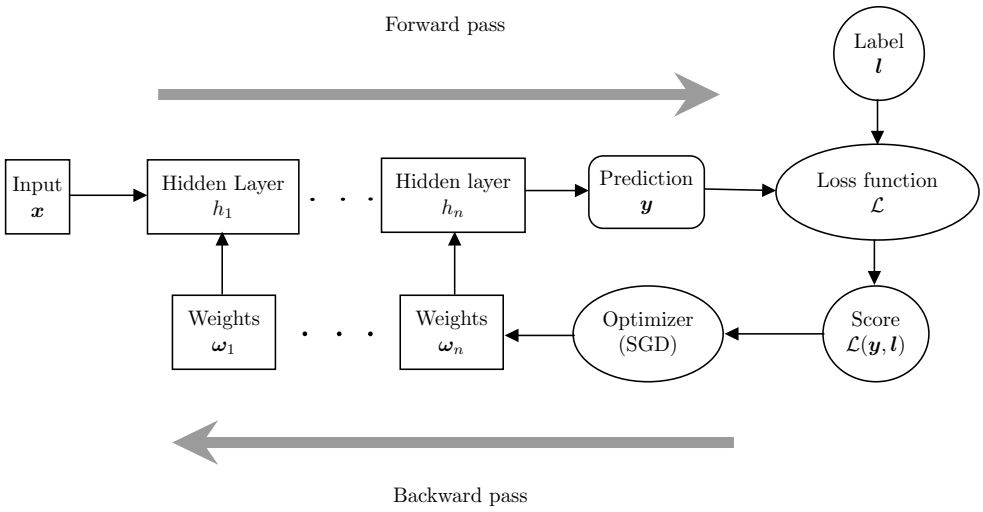


Figure 2.3: Flowchart of the learning process of an artificial neural network using back-propagation.

2.3 Convolutional Neural Networks

Convolutional neural networks (CNN) is a type of deep-learning model that is commonly used in computer vision applications [12]. In the context of image recognition, LeCun [19] is credited for being the first to use CNN with back-propagation. CNNs are designed to exploit data that has a grid-like topology [11]. This could for example be time-series vectors or images, which can be viewed as a two-dimensional grid of pixels. Data with this structure have spatial relations, which means that they are related to other data points close to its vicinity.

The main operation in a convolutional neural network is convolution which is defined, for two dimen-

sions, as

$$\mathcal{H}(i,j) = (X * K)(i,j) = \sum_{m=1}^{M}\sum_{n=1}^{N} X(i,j)K(i-m, j-n) \tag{2.7}$$

where X is an input matrix, K is the *kernel* matrix and the output $\mathcal{H}$, is called a *feature map*. If the convolution is used in the context of images, then X would be an image with dimensions $M \times N$. Convolution in this case can be viewed as the kernel floating across the image and collecting data in a region combining it into a feature [11]. A visualization of this can be seen in figure 2.4.

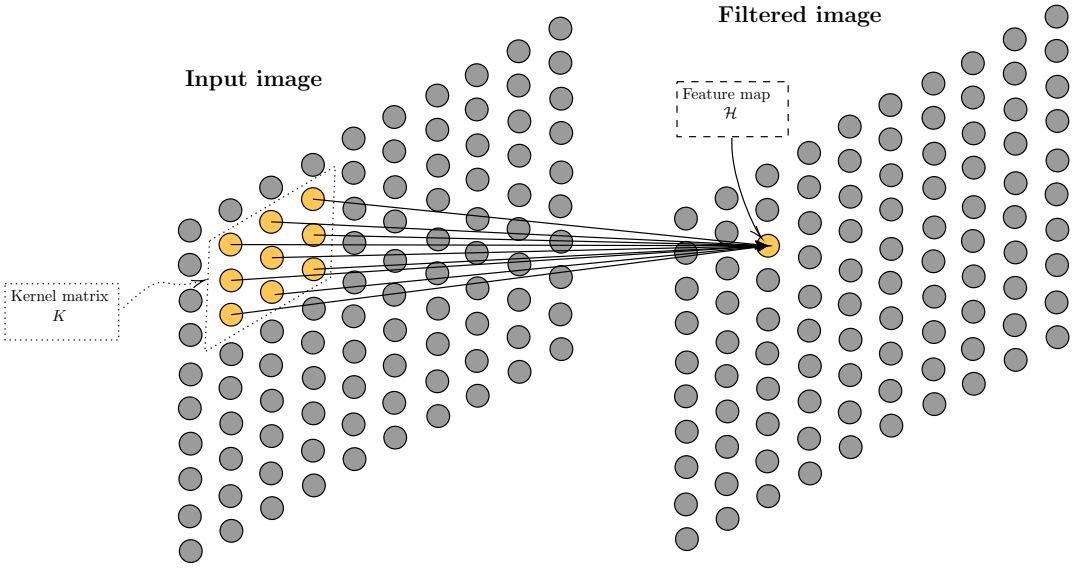


Figure 2.4: Feature extraction in a CNN.

During a forward pass in neural networks, such as the MLP, the hidden layers use matrix multiplications to traverse from input to output. In this case, all inputs interact with all outputs and thus creates a network with many parameters. If a CNN is used, the kernel matrix K can be made smaller than the input which leads to *sparse connectivity*. This lowers the number of parameters in the network, while still being able to preserve the features of the data [11]. This enhances the computational complexity of a CNN in comparison to other network types.

There are some other components of a CNN layer that needs to be taken into consideration. The data can be *padded* with zeros around its boundaries. This is done to preserve information close to the boundaries and to enforce specific dimensions of the resulting data [11]. *Stride* can be introduced to make the kernel skip a defined amount of steps to avoid overlapping and since the number of convolutions shrinks so does also the number of parameters. Finally, a *pooling-filter* can be applied as a final component to a CNN-layer. A pooling filter ensures that the output of the prior components is invariant to small translations of the input [11]. In other words, the pooling filter makes the features invariant to their location. For example, in image recognition, it could be more important whether an image contains a specific object rather than where it is located in the data. However, in the case of lip-synchronization, the main problem of the thesis mentioned in section 1.2 Problem Setting, features must remain at the same location.

The results of a CNN with many hidden layers, a *deep convolutional neural network*, is a feature map that is compressed to low dimensions. Since the convolution is a linear operation the CNN process can be inversely calculated by using the transpose of the matrix defined by the convolution [11]. The kernel of the CNN produces a down-sample of the input size and the transpose convolution corresponds to an up-sample of the data size.

2.4 Autoencoders

Conceptually, an *autoencoder* can be seen as a trained feed-forward ANN where the input $\boldsymbol{x}$ gets mapped to the output $\hat{\boldsymbol{x}}$ [11]. However, before the input $\boldsymbol{x}$ is transformed into the output $\hat{\boldsymbol{x}}$, it gets encoded into a compressed representation and then decoded again to the non-compressed representation. The compressed representation is denoted as the *embedding* while the part which transforms the input $\boldsymbol{x}$ to the embedding is denoted as the *encoder* and the part which transforms the embedding to the output $\hat{\boldsymbol{x}}$ is denoted as the *decoder*. When a specific input gets passed through to the embedding, its representation in the embedding is denoted as its fixed representation E. Additionally, it shall be noted that an autoencoder can have several encoders and decoders, as long as they all map to and map from the same embedding [5]. A schematic over the structure of an autoencoder can be seen in figure 2.5.

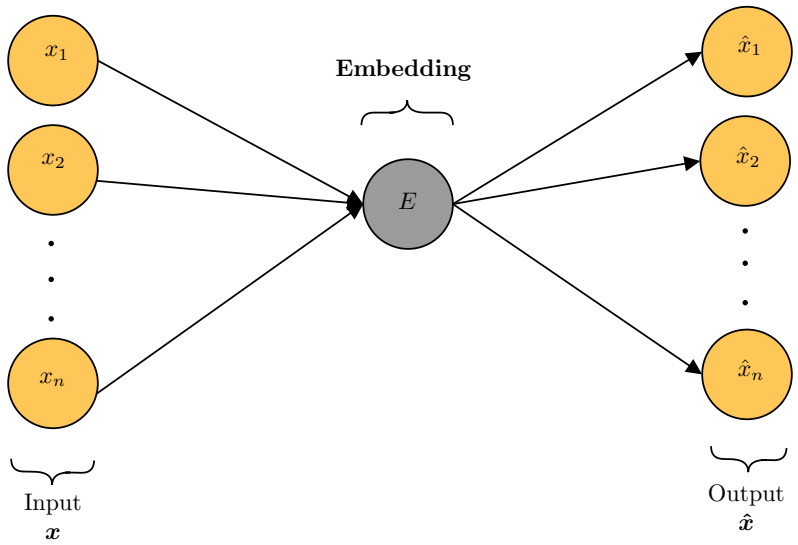


Figure 2.5: Schematic overview of an autoencoder.

The important notion about autoencoders is that they do not want to copy the input $\boldsymbol{x}$ perfectly to the output $\hat{\boldsymbol{x}}$. Instead, the embedding will cause an information bottleneck so the autoencoders need to priorities the data which gets kept for the output. If the autoencoder gets trained ideally, it will only learn to keep the useful properties of the input $\boldsymbol{x}$ to produce the desired output $\hat{\boldsymbol{x}}$. In a way, this can be viewed as a form of *feature extraction* from the data, and therefore, autoencoders are frequently used for unsupervised learning [11].

2.4.1 Skip connections

A special type of ANN is the *Residual Neural Network* which utilizes *skip connections* to *identity map* specific layers outputs to each other [11]. These skip connections can also be applied to an autoencoder. More formally, a skip connection is a connection which adds the input $\tilde{\boldsymbol{x}}$ to its output $\Phi(\tilde{\boldsymbol{x}})$ to yield the total output

$$\boldsymbol{y} = \Phi(\tilde{\boldsymbol{x}}) + \tilde{\boldsymbol{x}}. \tag{2.8}$$

The mapping Φ can consist of a single ANN layer to multiple ANN layers. Further, the mapping Φ together with its skip connection is usually denoted as a *residual block*. A visualization of a residual block with output $\boldsymbol{y}$ can be seen in figure 2.6.

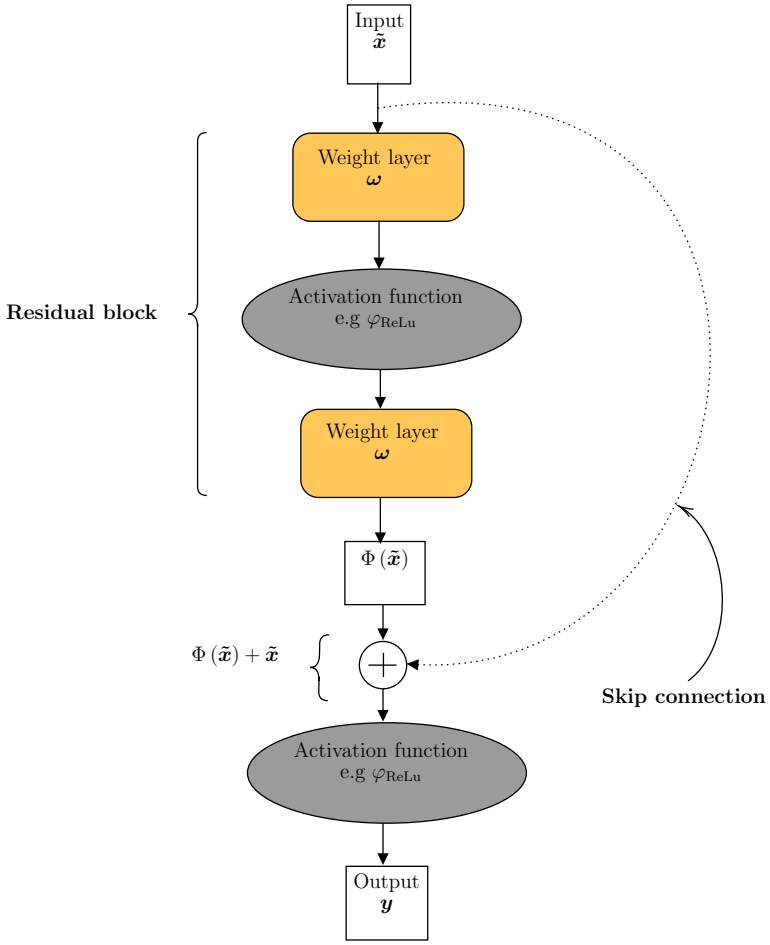


Figure 2.6: Visualization of a simple residual block with a skip connection and input $\tilde{x}$ and output y.

Additionally, skip connection can also be applied to an autoencoder. Although, the input $\tilde{\boldsymbol{x}}$ dimension needs to match the connected mapping $\Phi(\tilde{\boldsymbol{x}})$, which is easily done for an autoencoder by connecting to the corresponding encoder/decoder layer. However, a linear transformation can be used to make the input $\tilde{\boldsymbol{x}}$ match the dimension of the connected mapping $\Phi(\tilde{\boldsymbol{x}})$. Additionally, weights can also be added to the identity mapping of $\tilde{\boldsymbol{x}}$ in (2.8). An example of an autoencoder with skip connections between the encoder and decoder can be seen in figure 2.7.

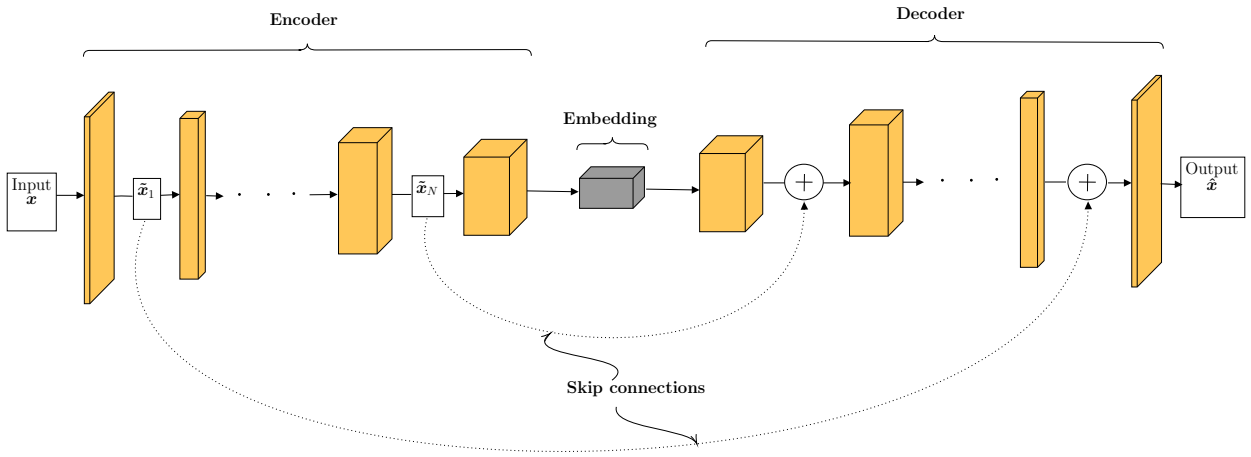


Figure 2.7: An example of skip connections in an autoencoder with N skip connections. The yellow blocks represent a general residual block.

The motivation behind the use of skip connections is that the identity mapping will enable deeper networks by reducing training error. Consequently, these deep networks will presumably enable a more accurate result. Meanwhile, an exact mapping of $\tilde{\boldsymbol{x}}$ will not add any extra parameters or computational complexity [20].

2.5 Audio representation

Speech is an essential part of lip-synchronization since it determines the lip movements of a person. Human speech is produced by the vocal cords in the form of vibrations that propagates as an acoustic wave, which is known as audio. As audio and speech are time-continuous waves, they need to be quantified into numerical values. This process must preserve the contents as well as the perceptual features.

For audio analysis, *mel-frequency cepstral coefficients* (MFCC) and *mel-frequency cepstrum* (MFC), more commonly known as *mel-spectrogram*, have been frequently used to represent speech and audio [21, 22]. These representations have also been widely used in different machine learning projects by, for example, Chen et al. [6] and by Prajwal et al. [7]. Mel frequency cepstral coefficients are simply the coefficients that make up a mel-frequency cepstrum. This method of representing audio was proposed by Mermelstein [23] in the context of speech recognition.

The idea of the mel-frequency cepstrum is to give a compact feature preserving representation of audio by creating a spectrogram with the mel-scale on its frequency axis. The *mel-scale*, or melody scale, is a non-linear scale developed to represent the perceptual scale of pitches. The human audible spectrum ranges from around 20 Hz to 20000 Hz. A specific difference of frequency in the lower side of this spectrum is more clearly audible than a difference in the higher range of the spectrum. Thus, the equal distances between pitches are non-linear and given by the mel-scale [21].

To produce the MFC, and by extension the MFCCs, the audio signal is cropped into T equal spaced time windows of audio that then are transformed to the frequency domain by for example fast Fourier transform (FFT). The frequency spectrum produced is then split into M equally spaced channels, according

to the mel-scale. The MFCCs are obtained by choosing the lowest amplitudes of the spectrum. However, Purwins et al. [21] advises against this due to information and spatial relations being destroyed. Yet, MFCCs does have some merit in creating models when compressed data is required.

CHAPTER 3

Generative Adversarial Network

This chapter is a general introduction to Generative Adversarial Networks. Here we introduce relevant concepts such as generative models followed by the original implementation of GAN. Additionally, problems surrounding GANs, such as convergence, stability, and Helvetica scenario / Mode collapse are discussed. Lastly, the specialized GANs WGAN-GP and LipGAN are introduced.

3.1 Generative Models

Generative models are statistical models that learn a representation of a probability distribution to create data instances from that distribution. This can be seen as a generative model that captures the joint probability of a target data distribution X and a label Y as defined by $P(X, Y)$. The adversary of a generative model is a *discriminative model* which captures the conditional probability $P(X|Y)$ [24]. Examples of discriminative tasks are classification and regression which discern between data. When a discriminative model is not separating data by labels and instead returns a scalar value it is known as a *critic* [25], such as in a *Wasserstein Generative Adversarial Network* (WGAN).

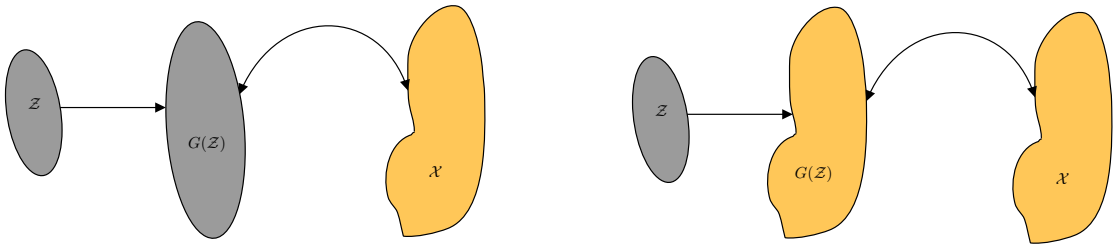
In the context of artificial neural networks, generative models are called *deep generative models* (DGM) and have many applications [26]. In this sense, the model attempts to learn a representation $\mathcal{X}$ defined over $\mathbb{R}^n$ often under the circumstances that n is large and for a complicated distribution. This is denoted as the target distribution and is used as training data for the DGM. The goal of the DGM is to obtain a *generator* G, defined as

$$G : \mathbb{R}^q \to \mathbb{R}^n \tag{3.1}$$

where the prior distribution $\mathcal{Z}$ is in $\mathbb{R}^q$. If the goal is achieved then for each sample $x \sim \mathcal{X}$, there exists one point $z \sim \mathcal{Z}$ such that $G(z) \approx x$. The generator is desired as a component to map a point in the more complicated distribution $\mathcal{X}$ to an easier distribution $\mathcal{Z}$. This problem can be solved by function approximation and thus a feed-forward neural network is suitable for this task [26].

A deep generative model is trained by samples $x \sim \mathcal{X}$ to update the weights $\boldsymbol{\omega}$ of the generator. This is done to render the sample output from the generator, $G(z)$, being indistinguishable from x. This means that the probability distribution $\mathcal{Z}$ is transformed to the probability distribution of $\mathcal{X}$, see figure 3.1

for illustration. The similarity, or distance, between these distributions, is complicated to measure and thus it is hard to determine when satisfying results are obtained [26].



(a) Before training **(b)** After training

Figure 3.1: Illustration of the probability distributions in a deep generative model.

A wide range of implementations of DGMs are used in many fields of machine learning. A type of DGM, *auto-regressive models*, are used for *natural language processing* (NLP). A recent implementation of an auto-regressive model is *GPT-3* that was released in 2020. It uses deep learning to generate text that looks authentic in a way that human evaluators have a hard time distinguishinging from texts generated by humans [27]. Another implementation of DGM is the *variational autoencoder* (VAE) which was introduced by both Kingma et al. [28] and Rezende et al. [29] in 2014. It has been used to perform image generation and automatic image editing [12].

3.2 Generative Adversarial Network

Generative Adversarial Network (GAN) is an implementation of a DGM proposed by Goodfellow et al. [30] in 2014. This was done in the context of utilizing generative models to create data, for example, natural images, audio waveforms, or symbols in natural language corpora. A GAN consists of two ANNs that work in tandem, a generative network, called the generator G, and a discriminative network called the *discriminator* D. The discriminator is tasked with distinguishing a sample created by the generator from a sample from the real probability distribution, this outputs a score based on the discriminator's decision. This process is repeated and consequently, G gets better at creating samples that look like they originate from the real probability distribution. In the meantime, the discriminator also improves at distinguishing between the two distributions. This can be viewed as a *two-player-zero-sum game* that continues, in theory, until the probability distribution of the generated samples is the same as the target probability distribution.

More formally, the progress of a GAN can be expressed as the following. A noise variable $\boldsymbol{z} \sim \mathbb{P}_z$ is used as input into the generator and thus mapped to data space as $G(\boldsymbol{z})$ where G is some feed-forward ANN. $\mathbb{P}_z$ is a random noise distribution, for example, *Gaussian noise*. A sample $\boldsymbol{x}$ is drawn from either the distribution formed by the generator, $\mathbb{P}_g$ or the real distribution $\mathbb{P}_r$. This is used as input into the other network of the GAN, the discriminator, to form $D(\boldsymbol{x})$. Where $D(\boldsymbol{x})$ represents the probability of $\boldsymbol{x}$ originating from either $\mathbb{P}_g$ or $\mathbb{P}_r$. See figure 3.2 for a schematic sketch of a GAN.

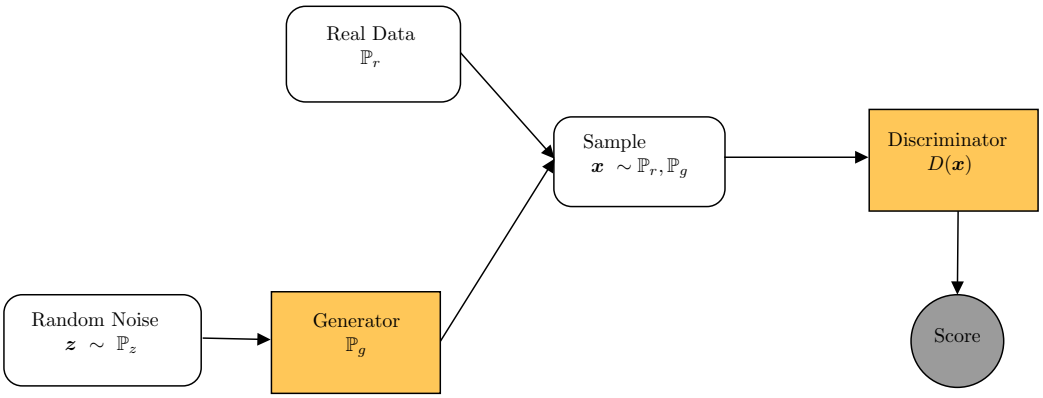


Figure 3.2: Schematic of a GAN with generator G and discriminator D.

3.2.1 Original implementation

In the original version proposed by Goodfellow et al. [30], the discriminator takes shape as a *binary classifier* with the labels $\boldsymbol{l} = \{0, 1\}$,

$$D : \mathbb{R}^n \to [0, 1] \tag{3.2}$$

where n is the dimensions of $\mathbb{P}_g$ and $\mathbb{P}_r$. D is trained to maximize the predictions of correct labels. The discriminator's loss function takes shape as a *cross-entropy* and is expressed as

$$\mathcal{L}_D = \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_g, \mathbb{P}_r}[\log D(\boldsymbol{x})]. \tag{3.3}$$

The generator also employs a cross-entropy loss function. G on the other hand tries to minimize the correct labeling to confuse the discriminator from making a good prediction. The loss for G is defined as

$$\mathcal{L}_G = \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_z}[\log(1 - D(G(\boldsymbol{z})))]. \tag{3.4}$$

The total loss is given by

$$\mathcal{L}(G, D) = \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_g, \mathbb{P}_r}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_z}[\log(1 - D(G(\boldsymbol{z})))] \tag{3.5}$$

where $\mathcal{L}$ is the objective function for the two-player min-max game [30, 31], earlier described as the zero-sum game. The game is defined as

$$\arg \min_G \max_D \mathcal{L}(G, D). \tag{3.6}$$

3.2.2 Training

Training the original GAN is done by performing the min-max game in equation (3.6). This is equivalent to finding a *Nash equilibrium* for the weights of the two players, G and D [26]. A Nash equilibrium is a two-player game where neither player can improve their *cost function* independent of the other player [32]. The cost function is, in this case, equivalent to the loss functions $\mathcal{L}_G$ and $\mathcal{L}_D$. To find the Nash equilibrium is a notoriously difficult problem to solve and one of the main complications with GANs [26, 31, 33]. The complications lies in that optimizing the loss function $\mathcal{L}$ is a *non-convex optimization problem* where the parameters, weights $\boldsymbol{\omega}$ in this case, are extremely high-dimensional.

Two problematic scenarios are described by Ruthotto et al. [26] during the initial phase of training. Early on, the discriminator easily distinguishes between the real data, from $\mathbb{P}_r$, and the generated data, from $\mathbb{P}_g$. If the discriminator is trained to optimality, then it would be impossible for the generator to improve. This is true since the gradient $\nabla_G \mathcal{L}(G, D^*)$ would be close to zero. In the original implementation by Goodfellow et al. [30], this problem is noticed and described as saturation in the generator loss $\mathcal{L}_G$. They suggest a new loss function for the generator,

$$\mathcal{L}_G = \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}[-\log(D(G(\boldsymbol{z}))] \tag{3.7}$$

to amend this problem. This version is called *non-saturating GAN* [34]. Another case could arise when D is not trained sufficiently, then the process of updating the weights of G would become challenging.

The issue when gradients are close to zero is called the *vanishing gradient problem*. The opposite to this problem, when a gradient is growing rapidly, is called *exploding gradient problem*. If one of these problems arises, then the network becomes untrainable and thus worth nothing. This is not only a problem in theory but also in practice when implementing a GAN numerically. Vanishing gradients will cause the updates of weights $\boldsymbol{\omega}$ to become smaller than *machine accuracy* and exploding gradients could cause the gradients to *overflow*, making the value incorrect.

3.2.3 Convergence and stability

In [30], it is proven that the probability distribution $\mathbb{P}_g$ converges to $\mathbb{P}_r$. This is true under some conditions, such that the updates occur in function space, D and G have infinite capacity and, that the resulting optimization problem is modeled as a convex-concave game. This yields well-defined global convergence properties. However, these conditions are not met in real-life implementations [30, 33]. In reality, it is often observed that gradient descent-based optimization does not lead to convergence for GANs [34].

Nagarajan et al. [33] show that GANs are locally stable if $\mathbb{P}_r$ and $\mathbb{P}_g$ are absolute continuous. This is often not the case since both distributions may lie in lower dimension manifolds [34]. An example of this could be walking straight ahead on a line, which is a one-dimensional manifold, where one can not reach the starting point again, without making a discontinuous jump. Whereas when walking on a two-dimensional manifold, like a sphere, walking back to the starting point is possible. Thus, in most cases, GANs are unstable during training, even close to the equilibrium point.

During training, GANs exhibit an oscillating behavior. At the beginning of training, the discriminator pushes $\mathbb{P}_g$ towards the true data distribution $\mathbb{P}_r$. While this happens the discriminator is trained to be more accurate and thus its gradient increases. As $\mathbb{P}_g$ reaches $\mathbb{P}_r$ the discriminator's gradient is the largest and pushes away $\mathbb{P}_g$ from the equilibrium point, until $\mathbb{P}_g$ is back at a point close to where it began but with opposite sign. This oscillating process continues infinitely and thus $\mathbb{P}_g$ never converges to $\mathbb{P}_r$. This can be seen as if the generator is trained to optimality, $\mathbb{P}_g$ is close to $\mathbb{P}_r$, then the discriminator classifies the correct label 50 % of the times. This would make the feedback from D to G be worth nothing since it is completely random.

According to Nagarajan et al. [33], the theory of convergence and stability of GANs are far outpaced by the practical applications. Despite the mentioned flaws above, GANs do work and produce powerful solutions for learning complex real-world distributions [34]. Yet, attempts to remedy the drawbacks of stability and convergence in GANs have been made. Nagarajan et al. [33] show that adding *regularization* to the gradient descent can make it locally stable. Mescheder et al. [34] propose a *gradient penalty* to induce local convergence.

Another way to improve stability in a GAN is to introduce *normalization*. Normalization is a method where

each unit inputs to a unit each have zero mean and unit variance [35]. This can be done in different shapes such as *batch normalization* [31] or *layer normalization* [25]. Normalization facilitates the optimization process of a GAN by inducing the discriminator to produce better quality feedback [36]. Also, normalization can be coupled together with running statistics to yield better performance.

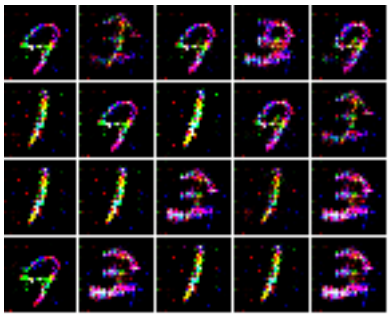
3.2.4 Helvetica scenario / Mode collapse

In the original GAN implementation [30], a problem is mentioned where G collapses too many values z to the same value of x which results in non-diverse samples. This is known as the *Helvetica scenario* [30] or more commonly as *mode collapse*. The consequence of mode collapse is that the generator does not contribute to training because it can not produce diverse samples. The problem arises when the generator optimizes in a greedy way to fool the discriminator. G thinks that it has found an optimal sample and continues to generate the same sample [36].

Mode collapse contributes to the training difficulties mentioned above. Some solutions do exist to remedy mode collapse and one of them is to make the discriminator have high generalization capabilities [36]. Another solution could be to implement gradient penalty or normalization [33, 34]. Mode collapse can be noticed in image generation by ocular inspection of samples during training [26].

An example of mode collapse can be seen in figure 3.3, where the output of the GAN suffers from mode collapse in two ways. Firstly, it fails to generate the desired colors of the real data, and secondly, it only produces 3 out of 9 digits. In this experiment a NS-GAN was used to perform the image generation [37].

(a) Original dataset



(b) Mode collapse

Figure 3.3: Example of mode collapse on the CMNIST dataset [37].

3.2.5 GAN variations

Despite the aforementioned complications with GANs, they have gained prominence as one of the most used methods for image generation that produce state-of-the-art results [33, 36]. Many modifications and extensions to the original GAN implementation by Goodfellow et al. [30] have been made to tackle the convergence and stability issues. These include *Least Squares GAN* [38], *Wasserstein GAN* [39], and *DRAGAN* [40].

Other GAN versions focus more on ad hoc implementations to perform certain tasks instead of improving the convergence properties. One of these implementations is called the *deep convolutional generative*

adversarial network (DCGAN). It was proposed by Radford et al. [35] in the context of using unsupervised learning in GANs to produce image representations. DCGAN implements a GAN structure with CNNs, which are suitable for image processing. This stands in contrast with the original implementation of GANs that used MLPs [30]. The structure of DCGANs includes some properties such as utilizing batch normalization and ReLU as activation function [35]. Even though DCGAN still suffers from some of the stability issues, it still poses a powerful implementation in image generation tasks.

3.3 Wasserstein Generative Adversarial Network

As previously mentioned, GANs are prone to instabilities during training and to mode collapse, and hence, attempts have been made to mitigate this. One of the most notable is the *Wasserstein Generative Adversarial Network* (WGAN), which was introduced by Arjovsky et al. [39] in 2017. Unlike classical GANs which utilize a discriminatory approach of binary classification, WGANs take another path of trying to minimizing the statistical distance between the generated data distribution $\mathbb{P}_g$ and the real data distribution $\mathbb{P}_r$ [26]. Therefore, a WGAN consists of a generator G and a critic D, and not a discriminator as in the original GAN. However, the two-player zero-sum game from the original GANs persists, with slight modifications to the switching intervals between generator and critic during training [39].

The first remedy to the instabilities is to have a GAN loss function that is continuous everywhere and differentiable almost everywhere. One cost function where this hold is the statistical measurement *earth mover's distance* (EMD), also known as the *Wasserstein-1 distance* W_1. Consider the generated data distribution $\mathbb{P}_g$ and the real data distribution $\mathbb{P}_r$ which forms the marginals for the joint distribution $\Gamma(\mathbb{P}_r, \mathbb{P}_g)$. Then the optimal transportation plan to move $\mathbb{P}_g$ from its support to $\mathbb{P}_r$ and its support, is given by the EMD [41]. Further, using the Euclidean norm as distance yields the following

$$W_1(\mathbb{P}_g, \mathbb{P}_r) = \inf_{\gamma \in \Gamma(\mathbb{P}_g, \mathbb{P}_r)} \mathbb{E}_{(a,b)\sim\gamma}\big[\|a - b\|\big] \tag{3.8}$$

for the samples a and b from the joint distribution Γ. Using Kantorovich-Rubinstein duality [42], equation (3.8) can be written to the more practical formulation

$$W_1(G(\mathbb{P}_z), \mathbb{P}_r) = \max_{f \in \mathrm{Lip}(f) \leq 1} \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}[f(G(\boldsymbol{z}))] \tag{3.9}$$

where the maximum is taken over all *1-Lipschitz* continuous functions $f : \mathbb{R}^n \to \mathbb{R}$ [39]. Additionally, it shall be added that W_1 is invariant up to a positive scalar k if the Lipschitz constraint is modified to be k-Lipschitz [43]. Now, the problem becomes to find these functions f, which is done using approximation through a neural network. Further, this network will act as the critic D in the WGAN setup, and W_1 will act as a loss function [26]. This yields the critic loss

$$\mathcal{L}_D = \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}\big[D(G(\boldsymbol{z}))\big] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big]. \tag{3.10}$$

Furthermore, the gradient from the critic D will be used to train the generator G [39]. Likewise, this gives the generator loss

$$\mathcal{L}_G = -\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}\big[D(G(\boldsymbol{z}))\big], \tag{3.11}$$

and lastly, the total loss

$$\mathcal{L}_{\mathrm{WGAN}}(G, D) = \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}\big[D(G(\boldsymbol{z}))\big] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big]. \tag{3.12}$$

Similarly, the two-player zero-sum game will persist, and for the generator G to minimize (3.10) while the approximate 1-Lipschitz continuous critic D will maximize (3.11) [25], i.e

$$\arg\min_{G} \max_{D\in \mathrm{Lip}(D)\leq 1} \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}\big[D(G(\boldsymbol{z}))\big] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big]. \tag{3.13}$$

One advantage of solely using W_1 as a metric is that this can be continuously observed so the critic can be trained until optimally before the training of the generator, unlike classical GANs where this would cause a vanishing gradient. Additionally, this will also help to mitigate mode collapse [39]. Lastly, if G is continuous, differentiable almost everywhere, and locally Lipschitz continuous, then the loss W_1 will be continuous [26].

3.3.1 Enforcing Lipschitz constraint

One practical problem of a WGAN is to enforce the Lipschitz constraint. In the original implementation [39], *weight clipping* was used to keep the weights $\boldsymbol{\omega}$ in an compact interval $[-\xi, \xi]$, after each gradient update. One obvious question from this approach is what ξ should be? If ξ is too large, then it might take a long time for the weights to reach their limit and consequently make it harder to train the critic to optimality. On the contrary, if the weights are too small, then a vanishing gradient can occur [39]. However, it shall be noted that these problems can be mitigated with batch normalization in the critic. However, this will make the critic fail to converge according to [25].

Instead of answering the previous question, another approach that does not involve weight clipping can be used. One of these methods is gradient penalty and was introduced as *Wasserstein Generative Adversarial Network - Gradient Penalty* (WGAN-GP) in [25]. In this method, a regularization by a gradient penalty $\mathcal{R}_{\mathrm{GP}}$ term is introduced. This gradient penalty is defined as

$$\mathcal{R}_{\mathrm{GP}} = \mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\left[\left(\left\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\right\|_2 - 1\right)^2\right] \tag{3.14}$$

where $\hat{\boldsymbol{x}}$ is the output of the generator i.e $G(\boldsymbol{z}) = \hat{\boldsymbol{x}}$, $\boldsymbol{z} \sim \mathbb{P}_z$. Introducing this term to the original WGAN loss (3.12), together with the penalty coefficient λ yields

$$\mathcal{L}_{\text{WGAN-GP}}(G, D) = \underbrace{\overbrace{\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_g}\left[D(\hat{\boldsymbol{x}})\right]}^{\text{Generator loss } -\mathcal{L}_G} - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\left[D(\boldsymbol{x})\right] + \lambda\mathcal{R}_{\mathrm{GP}}}_{\text{Critic loss } \mathcal{L}_D}. \tag{3.15}$$

The motivation behind the gradient penalty term in (3.15), is that a differentiable function is 1-Lipschitz iff it has gradients with the norm at most 1 everywhere [25]. Therefore, the penalty terms penalize gradients with norm separate from 1, and consequently encourages the norm to go towards 1. Unfortunately, this introduction of the gradient penalty term increases the computational complexity and thus the time required for convergence [44]. Additionally, the penalty terms for each critic input $\hat{\boldsymbol{x}}$ are calculated individually. Therefore, batch normalization can not be used since it will input a whole batch instead. However, normalization schemes that introduce no correlation between examples can be used. Notably, layer normalization can be used [25]. Lastly, it shall be added that there are other penalty methods, notably WGAN-LS [45] which modifies the penalty terms in (3.14) slightly. Additionally, there are also methods to circumvent the Lipschitz constraint, such as WGAN-div [43], c-transform/(c, ϵ)-transform WGAN [46].

3.3.2 Remarks about the W_1 approximation

As of recent, the approximation of the Wasserstein distance W_1 using (3.9) has been questioned and been stated to be impossible to approximate in practice [44]. Firstly, they argued that it is impossible to optimize over all 1-Lipschitz functions accurately. Therefore, an exact optimal discriminator will never be obtained. Secondly, they noted that during practical implementations, the network does not have access to the full distributions $\mathbb{P}_g$ and $\mathbb{P}_r$, but only finite subsamples because of the use of mini-batches.

Besides pointing out the flaws of the approximation, there have also been comparisons to presumably better approximations of the Wasserstein distance W_1 [44, 47]. This highlights that the approximation in (3.9) takes longer to converge to the true distance compared to other approximations. However, [44] also looks into the generative performance of the different approximations, specifically between WGAN-GP and the c-transform. There it is noted that while the c-transform is better in approximating the exact W_1, it outputs a worse objective generative performance in the task of face generation. It is noted that the faces become more blurry than what WGAN-GP outputs, see figure 3.4 for samples from their experiments.

Additionally, the authors also argue that the Wasserstein distance W_1 is not a suitable loss for the task of image generation since it utilizes pixel-wise distance, which is not a perceptual distance metric for human vision. Finally, they attribute the initial success of WGANs over classical GANs to the ability to control the Lipschitz constant of the discriminator. Additionally, they emphasize the importance of good initial hyperparameter tuning, which might give misleading results over different GANs.

(a) WGAN-GP **(b)** c-transform WGAN

Figure 3.4: Samples of generated faces using WGAN-GP and c-transform [44].

3.4 LipGAN

In the article Towards Automatic Face-to-Face Translation, Prajwal et al. [7] introduces LipGAN. They use this GAN in a pipeline that inputs a video in the source language L_A and translates it to a target language L_B with correctly lip-synced lips for the target language.

The pipeline starts by inputting the source language L_A audio into an *automatic speech recognition* (ASR) module which converts the speech to text. Further, this text is used as input to a translation software which translates the source language L_A text into target language L_B text. Then, this text gets converted to target language L_B audio through a *text-to-speech* (TTS) module which then also gets processed through a *voice transfer software* to make the output more human-like and diverse. This audio then gets converted into mel-spectrograms, while the video gets chopped up into the corresponding frames of the video.

As usual, the GAN consists of a generator G and a critic D. Importantly, LipGAN is not a conventional GAN in the sense that it does not input random noise into the generator G to produce the generated distribution. Instead, LipGAN inputs frames and audio from an input distribution $\mathbb{P}_z$ and outputs a generated lip-synced frame in the output distribution $\mathbb{P}_g$. Further, the critic D will be fed real frames from the data distribution $\mathbb{P}_r$ and generated frames from the generator's output distribution $\mathbb{P}_g$. The foundation of LipGAN is built on the DCGAN structure in combination with residual blocks and skip-connections [7].

3.4.1 Generator G

As mentioned, the input to the generator consists of two images of dimension $H \times H \times 3$, which are concatenated channel-wise to a tensor of dimension $H \times H \times 6$. During training, these consist of a ground truth frame where the lower part has been masked S_m and a time-unsynced frame S', which is a frameshifted with the time step $\pm\alpha$. The purpose of the masked frame S_m is to give the generator information about the pose of the person. The masked frame S_m is obtained by taking the ground truth frame S and changing the lower half of the image to a black color i.e for an $H \times H$ frame, setting the lower $\frac{H}{2} \times \frac{H}{2}$ pixel values to 0 for an 8-bit image. As for the audio input, it consists of an audio segment A connected to a corresponding frame S and is given as a mel-spectrogram of shape $M \times T$, where M represents the number of frequency channels, and T is the time window for the audio segment.

Moving over to the structure of the generator, it consists of an autoencoder that has three modules; one audio encoder, one face encoder, and a face decoder. The audio encoder consists of a standard CNN with 4 residual blocks and outputs and audio embedding. Likewise, the face encoder consists of a standard CNN with 7 residual blocks and outputs to a face embedding. This face embedding gets concatenated with the audio embedding, which forms one single embedding which gets passed to the face decoder. Similarly, this face decoder consists of a standard CNN with 7 residual blocks. Importantly, there exist six skip connections from a residual block in the face encoder to the corresponding residual block in the face decoder. The purpose of these skip connections is to preserve facial information, which might be lacking in the embedding. Finally, the face decoder outputs the generated output of the generator i.e a generated frame $\hat{S}$. An overview of LipGANs generator structure can be seen in figure 3.5.

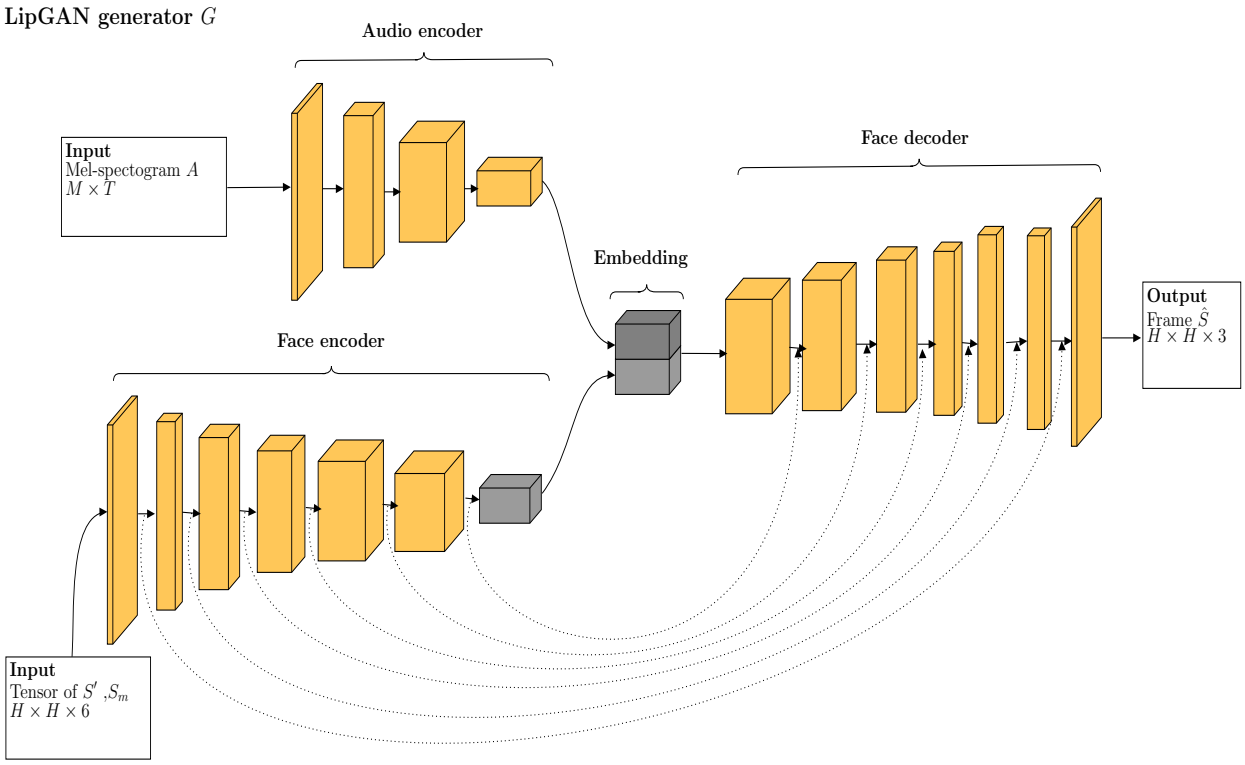


Figure 3.5: Overview of the generator G used in LipGAN.

3.4.2 Critic D

The critic input consists of a true frame S from the data distribution $\mathbb{P}_r$ or a generated frame $\hat{S}$ from the generator G, both as a tensor of size $H \times H \times 3$. Importantly, LipGANs critic also gets fed true frames S with time-unsynced audio A' which shall be considered faked samples. This so it will learn to differentiate unsynced audio as well, and not solely discriminate on the image quality.

As for the structure of the critic, it consists of an audio encoder and a face encoder. The first inputs mel-spectrogram of shape $M \times T$ and consists of a CNN with 4 residual blocks. Furthermore, the face encoder consists of a CNN with 7 residual blocks. The output of the critic is two fixed representations for the input in the audio embedding E^A and in the face embedding E^S. An overview of LipGANs critic D structure can be seen in figure 3.6

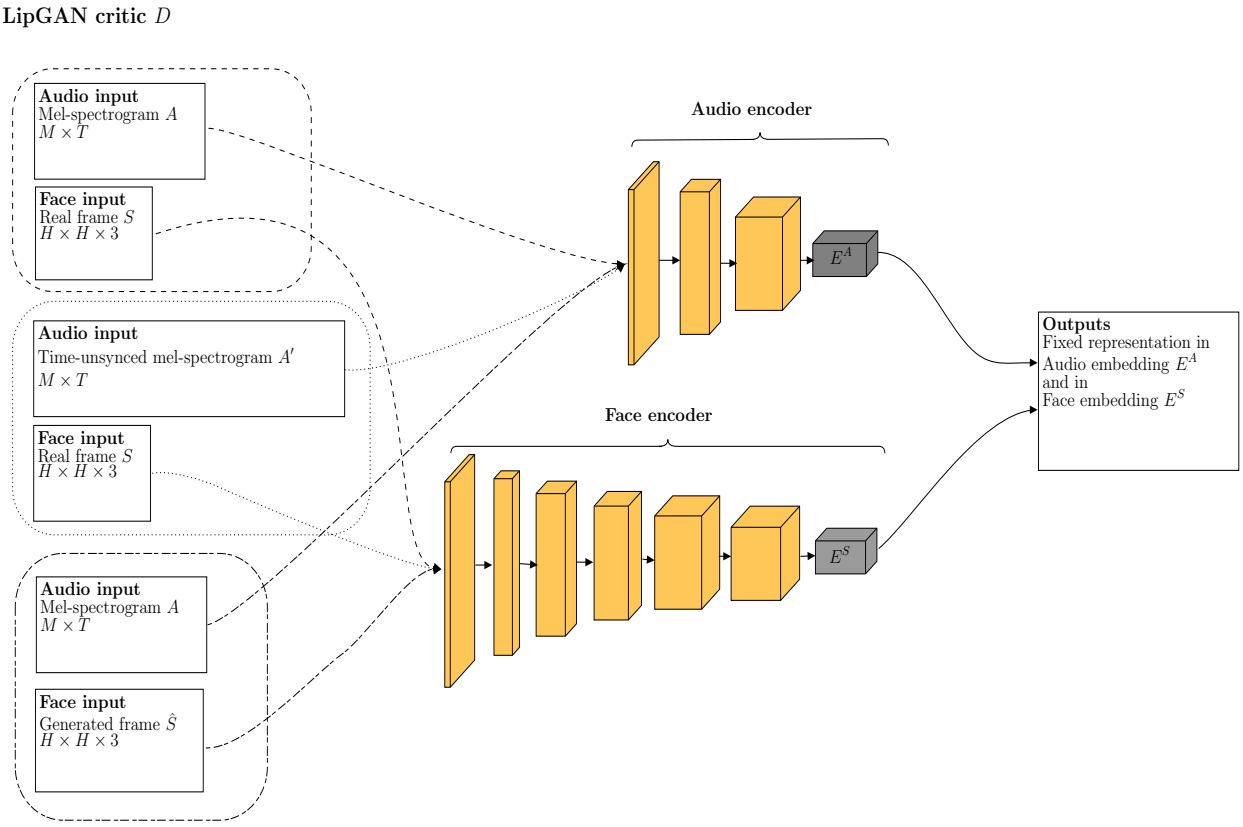


Figure 3.6: Overview of the critic used in LipGAN. Note the different input cases to the critic D.

3.4.3 Losses

Similar to a traditional GAN, LipGAN has a separate loss for the generator G denoted as $\mathcal{L}_G$ and a separate loss for the critic D denoted by $\mathcal{L}_D$. These losses are optimized in an attempt to obtain the optimal generator G^* and the optimal critic D^*.

Firstly, the critic outputs the contrastive loss $\mathcal{L}_c$, which is frequently used for feature mapping, a similar problem to lip-synchronization [4]. The ambition of the contrastive loss is to bring paired data closer together while doing the opposite for unpaired data. The pairs in this case are the video and audio embeddings and their labels determine their coupling. The contrastive loss is given by

$$\mathcal{L}_c(d_i, l_i) = \frac{1}{2N} \sum_{i=1}^{N} \left((1 - l_i) \cdot d_i^2 + l_i \cdot \max(0, m - d_i)^2 \right) \tag{3.16}$$

where N is the number of samples for computing the loss i.e usually the batch size, m the margin which is a user-defined parameter, l_i which is the label for the sample e.g $l_i = 0$ for a fake sample, and $l_i = 1$ for a true sample. Lastly, d_i represents the L^2 distance between the samples fixed representation in the audio embedding E^A and in the face embedding E^S i.e

$$d_i = \left\| E_i^A - E_i^S \right\|_2. \tag{3.17}$$

The contrastive loss $\mathcal{L}_c$ is then used to obtain two other losses, $\mathcal{L}_{\text{real}}$ and $\mathcal{L}_{\text{fake}}$, depending on which sample gets inputted to the critic D. For a real frame S, together with the true audio A, and consequently the label $l = 1$, the real loss is obtained as

$$\mathcal{L}_{\text{real}} = \mathbb{E}_{S,A}[\mathcal{L}_c(D(S, A), l = 1)]. \tag{3.18}$$

However, for the fake loss $\mathcal{L}_{\text{fake}}$, two fake inputs can be used. Namely, a generated input from the generator $G(S', A) = \hat{S}$ with time-synced audio A, called $\mathcal{L}_{\text{face}}$, or a real frame S with time-unsynced audio A', called $\mathcal{L}_{\text{audio}}$. This yield the following combined loss

$$\mathcal{L}_{\text{fake}} = \mathcal{L}_{\text{face}} + \mathcal{L}_{\text{audio}} \tag{3.19}$$

where

$$\begin{aligned} \mathcal{L}_{\text{face}} &= \mathbb{E}_{\hat{S},A}[\mathcal{L}_c(D(\hat{S}, A), l = 0)], \\ \mathcal{L}_{\text{audio}} &= \mathbb{E}_{S,A'}[\mathcal{L}_c(D(S, A'), l = 0)]. \end{aligned} \tag{3.20}$$

These losses, $\mathcal{L}_{\text{real}}$ and $\mathcal{L}_{\text{fake}}$ are then combined as an average as

$$\mathcal{L}_D(G, D) = \frac{\mathcal{L}_{\text{real}} + \mathcal{L}_{\text{fake}}}{2}. \tag{3.21}$$

This is the loss which the critic tries to maximize which yields the optimal discriminator

$$D^* = \arg\max_D \mathcal{L}_D(G, D). \tag{3.22}$$

Moving over to the generator G, it utilizes a reconstruction loss $\mathcal{L}_{\text{re}}$, which is given by

$$\mathcal{L}_{\text{re}}(G) = \frac{1}{N} \sum_{i=1}^{N} \left\| S - G(S', A) \right\|_1. \tag{3.23}$$

The generator tries to minimize this loss, while it at the same time tries to counter the discriminator's loss $\mathcal{L}_D$ by minimizing it. In total, this yields the following optimal generator

$$G^* = \arg\min_G \mathcal{L}_{\text{re}}(G) + \mathcal{L}_D(G, D). \tag{3.24}$$

CHAPTER 4

Technologies & Datasets

In this chapter, the technology and the dataset for all models in this thesis are presented. The software used includes; Python, `numpy`, `Pytorch`, `OpenCV`, `FFmpeg`, `dlib`, and `librosa`. The hardware used for the thesis is described. Lastly, the dataset used for training all of the models is introduced. Additionally, the dataset utilized by the original implementation of LipGAN [7] is also presented and compared to the one used for the models in this thesis.

4.1 Software

4.1.1 Python

Python is an *open-source* general-purpose programming language. It has a big community and a rich selection of packages, especially for machine learning, which it is especially suited for owing to Python's generator function. Therefore, Python was used for all of the implementations in this thesis.

4.1.2 `numpy`

Optimized numerical computations can be enabled in Python using the open-source library `numpy` [48]. It is especially useful for linear algebra operations owing to its `numpy array` data type. For this thesis, `numpy` is primarily used to store vector and matrices as `numpy array` objects, and to perform some basic linear algebra operation on them.

4.1.3 `Pytorch`

`Pytorch` is an open-source machine learning framework for Python, C++, and Java, originally developed by Facebook's AI Research lab [49]. It described itself as being easy to use, while also providing high performance; mainly owing to its use of hardware acceleration. One notable use of hardware acceleration is its use of a *graphical processing unit* (GPU) for tensor operations, which it does using Nvidia's parallel computing platform CUDA [50]. Additionally, it can also utilize Nvidia's cuDNN library, which provides highly tuned implementations for common deep learning routines [51]. Altogether, this will provide a speed boost for the task of deep learning, something which is especially important for GAN training. Therefore, `Pytorch` was used in the thesis for all of the network implementations. Also, `Pytorch` provided all of the

optimization solvers which were used for this thesis. Lastly, `Keras` [52] should also be mentioned as another machine learning framework commonly used in Python, which was used in the original implementation of LipGAN [7].

4.1.4 `OpenCV`

Open-Source Computer Vision Library (`OpenCV`) is an open-source computer vision library written natively in C++ [53]. However, it has a wrapper to Python, and could therefore be utilized for the models in this thesis. The purpose of `OpenCV` in this thesis, is to ease the task of resizing, loading, and saving images to a desired format i.e `numpy arrays` and .jpg images.

4.1.5 `FFmpeg`

`FFmpeg` is an open-source multimedia software, which can decode, encode, transcode, stream, filter, and play digital video or audio [54]. Its purpose for this thesis is to obtain the frames per second (fps) rate of the videos used. Additionally, it is also used to separate or combine the video and audio.

4.1.6 `dlib`

The C++ toolkit `dlib`, is an open-source library containing many machine learning algorithms [55]. Additionally, `dlib` can also be used in Python. For this thesis, `dlib` is solely used for the task of face detection. In the pre-processing step, this is done using its implementation of the histograms of oriented gradients (HOG) + linear support vector machine (SVM) facial detection [56]. While for the inference, a pre-trained max-margin object detection (MMOD) [57] CNN is used. The reason for the use of different facial detection methods between pre-processing and inference is the tradeoff between accuracy and computational time, where the MMOD CNN is more accurate but takes longer than the HOG + linear SVM approach. The latter is especially important since the pre-processing of a 27-hour dataset can take days, even on modern hardware using multithreading.

4.1.7 `librosa`

The open-source Python package `librosa` is a music and analysis tool [58]. For this thesis, it will be used to process the audio. This processing consists of sampling the audio and convert it to mel-spectrograms, which involves the steps described in section 2.5 Audio representation.

4.2 Hardware

As of 2021, working with GANs is a very computationally costly procedure, and consequently, can take a long time to train. As an example, for this thesis, the fastest model to train still took about a day to train with a small dataset and the fastest GPU available for the experiments. However, if this would be done without utilizing the hardware acceleration mentioned in subsection 4.1.3 `Pytorch`, it would take approximately 26 days. Therefore all experiments were conducted using a GPU and the optimization mentioned in subsection 4.1.3 `Pytorch`. To further speed up the training process, experiments ran in parallel on two different systems. The two systems used the GPUs Nvidia RTX 2080 TI and Nvidia RTX 2070 respectively.

4.3 Datasets

To train an artificial neural network, training data is required. This *training data* constitutes the experience from which the machine learning algorithm learns. A dataset is divided into different *data points* and most

often there is a large quantity of these [11]. Since this thesis revolves around creating videos, the dataset consists of these. A dataset can be more diverse if it includes videos in different settings. In general, the performance of a network is determined by how well it achieves its task on data that it has not seen before. Due to this reason a dataset is often split into *training data* and *test data*. The model is trained on the training data and the performance is verified on the test data [11].

In this thesis, a dataset of videos called *GRID* was used. It was introduced by Cooke et al. [59] as a corpus for tasks such as speech perception and speech recognition. GRID contains 33 unique speakers that each utter 1000 sentences in separate videos, that are about 3 seconds long. In total, this results in about 27.5 hours of 25 frames per second video with a resolution of 360×288 and a bitrate 1 kbit/s. Out of the 33 speakers, 16 were female and 17 were male, and all speakers had English as their first language. The videos are filmed in a lab environment with a green screen background rendering a clinic setting for the videos. The speakers are always faced forward and looking into the camera.

In the original implementation of LipGAN, a dataset called *LRS2* [60] was used to train their model. This dataset differs much from the GRID corpus. It is not made up of videos recorded in a lab environment but made up of news recordings from the British Broadcasting Corporation. This makes the LRS2 being more of a *in-the-wild* dataset that captures real conversations and different face poses. It also contains different lighting, different backgrounds, and people with different origins, which GRID lacks. The total length of data used in the implementation of LipGAN was around 29 hours [7].

The two aforementioned datasets differ a lot in terms of properties and thus they constitute the properties of the resulting model after training. A model trained on GRID might perform better on studio-recorded videos where the speaker looks into the camera whereas the LRS2 should perform better on in-the-wild situations. For this thesis, the choice fell on the GRID dataset since it matches the preferences of personalized video messages better, and due to its availability. Nine sample images from the respective datasets can be seen in figure 4.1.

(a) GRID

(b) LRS2

Figure 4.1: Samples of speakers from the datasets GRID [59] and LRS2 [60]. Note that the resolution in the LRS2 samples is not representative since it is a screenshot.

CHAPTER 5

Methodology

Here, the methodology used in this thesis is discussed. The pre-processing, followed by our implementation of LipGAN and WGAN-GP on the LipGAN architecture is presented, including architectural, training, inference, and code details. The experiments conducted for this thesis are explained. Lastly, the novel model L1WGAN-GP, developed in this work, is described in section 5.4 Experiments.

5.1 Data Pre-processing

Before the training data gets passed through any of the presented networks, it needs some pre-processing. The GRID dataset was used for this thesis, and was the only dataset subject to pre-processing. This consisted of the following parts:

- Chop up the video into frames.
- Crop out the face and rescale it to the correct image dimension $H \times H \times 3$. Save it into .jpg format.
- Chop up the audio to corresponding audio segments in mel-spectrograms representation.
- Put the files in a specific file structure.
- Split data into train and test dataset.

This was mainly implemented using the preprocess and audio code[1] from the original LipGAN article [7]. The reason to only use the faces as input data is to reduce visual disturbances such as background or non-speaking persons. Also, it will reduce the input dimension and consequently reduce the computational complexity of training the GANs.

The first step was done using the Python package `OpenCV` and the `VideoCapture` method, which chopped up the frames to `numpy arrays` using the frames per second of the video. Then for the second step, `dlib` was used to detect the faces in the frames using the HOG + SVM face detector. If a face was detected, the face was rescaled into a $H \times H \times 3$ `numpy array` and saved into a .jpg file using `OpenCV`. On the contrary, if no face was detected, the frame was discarded. For the third preprocessing step, `FFmpeg`

[1] `https://github.com/Rudrabha/LipGAN/tree/fully_pythonic` (fully_pythonic branch)

was used to separate the audio from the video and convert it into .wav format. Later, the Python package `librosa` was used to load the .wav data file and convert it into a mel-spectrogram as a `numpy array`, which then got compressed into the .npz format. Utilizing a .txt file with the desired file structure, the later steps would put their output in the specified file structure.

All pre-processing resulted in training data which consisted of the real face inputs S, and the shifted frame S', which had been resized to size $96 \times 96 \times 3$ i.e $H = 96$. Furthermore, the shifted frames S' were obtained by picking a frame using a time step $\pm\alpha$, where α is of random size $\alpha = 1, 2, ..., 6$. As for the audio data, it consisted of mel-spectrograms with $M = 80$ mel-frequency channels, and a time window of $T = 27$. This time window equivalates to about 300 ms of total audio, which is spread out evenly before and after the frame. Lastly, all data attributes can be seen summarized in table 5.1.

Table 5.1: Data attributes for the training data.

Data attributes	
Input image horizontal/vertical dimension H	96
Frameshift time step α	$1, 2, ..., 6$
Mel-frequency channels M	80
Mel-spectrogram time window T	27

Finally, all the pre-processed data resulted in 2202106 frames of faces, together with 33000 mel-spectrograms. This was then subdivided into the three sub-datasets GRIDSmall, GRIDFull, and GRIDTest. The first two subsets, GRIDSmall and GRIDFull, contained 300 and 980 video samples respectively from each of the 33 speakers. As the name suggests, the latter subset, GRIDTest, was used to test the models and therefore had no intersection of data with the two previously mentioned sub-datasets, which are used for training. Further, the test datasets contained 43929 image samples, which was specifically chosen since it matches the sample sizes used to calculate some specific GAN metrics, similar to other GAN comparison articles [61, 62]. All sub-datasets can be seen summarized in table 5.2.

Table 5.2: Information about the data subsets used for all experiments.

Name	**Type**	**Individual samples**	**Videos per speaker**
GRIDSmall	Train	670758	300
GRIDFull	Train	2190517	980
GRIDTest	Test	44589	20

5.2 LipGAN implementation

As a first model for the task of lip-synchronization, an implementation of LipGAN [7], as described in section 3.4 LipGAN, was used. Additionally, the training utilized the different subsets of the GRID dataset, which was pre-processed as mentioned in section 5.1 Data Pre-processing.

5.2.1 Architectural implementation

The overall architecture was nearly identical to the one described in section 3.4 LipGAN. There were three different types of residual blocks used. Namely, the generator's convolutional block, the critic's convolutional block, and the transposed convolutional block.

As the name suggests, the generator's convolutional block was only used for the generator in the face and audio encoder and the face decoder. There, the kernel was sized 4, 3, and 1 with a stride of 3, 2, or 1. This creates the $1 \times 1 \times 512$ output embedding from the face encoder and the audio encoder, which concatenates to the joint $1 \times 1 \times 1024$ embedding which inputs to the face decoder. This decoder also consists of a transposed convolutional block, which does a transposed convolution using solely a kernel size of 3 and a stride of 2. Lastly, both the generator's convolutional blocks and the transposed convolutional block used the ReLU activation function. Additionally, the blocks utilized batch normalization with a momentum of 0.8.

Moving over to the critic, it utilizes solely the critic's convolutional block. In the face encoder, it has convolutional blocks with the kernel sizes 7, 5, and 3 with a stride of 2 and 1. For the audio encoder, it solely uses the kernel size 3 with strides of 3 and 1. For both encoders, the last layer consists of zero padding on both sides of the input. This yielded the output embeddings E^S and E^A of size $1 \times 1 \times 512$ each. Additionally, all residual blocks used the LeakyReLU activation function, with a negative slope of 0.2. Lastly, all blocks in the critic used instance normalization with a momentum of 0.1. In total, all implementations resulted in a model with 47424915 trainable parameters, where 37087763 are for the generator and 10337152 for the critic.

5.2.2 Training implementation

For the training, a random seed for the initial weights $\boldsymbol{\omega}_{\text{Init}}$ was chosen, to be the same for all experiments, unless otherwise mentioned, since the random seed has been shown to have an impact on the performance for different GANs [63]. The random seed which was selected was the `numpy` random seed 10 i.e `numpy.random.seed(10)`.

As for the training, it consisted of 20 epochs with a batch size of 128. In each training step, the critic had an equal probability to be given a real face input S together with time-unsynced audio A' or a faked face $\hat{S}$ from the generator together with time-synced audio A. Additionally, it was always given a real data sample from the training dataset. Then, from the output score, backpropagation was used to train the discriminator using its loss $\mathcal{L}_D$. This loss was then used together with the reconstruction loss $\mathcal{L}_{\text{re}}$ to obtain the generator loss $\mathcal{L}_G$ to train the generator by backpropagation.

For the contrastive loss $\mathcal{L}_c$, a margin of $m = 2$ was used. Further, ADAM was used as optimizer for both the critic and the generator. Additionally, for both networks, an initial learning rate of $\eta = 10^{-3}$ and the decay parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$ was used for ADAM. All training parameters can be seen summarized in table 5.3.

Table 5.3: Training parameters used for the LipGAN model.

LipGAN training parameters	
Random seed for $\boldsymbol{\omega}_{\text{Init}}$	`numpy.random.seed(10)`
Epochs	20
Batch size	128
Contrastive loss margin m	2
Initial learning rate η	10^{-3}
ADAM first decay parameter β_1	0.5
ADAM second decay parameter β_2	0.999

5.2.3 Inference implementation

For inference of the model, the inference data first had to be pre-processed. This was done in a similar matter to the training data, as seen in section 5.1 Data Pre-processing, with frame and audio chopping combined with the facial detection. However, the steps of putting the files in a specific file structure and splitting the datasets were omitted. Lastly, it should be added that the inference data does not necessarily have to be videos, but photos can be used as well. In this case, the photo corresponds to each mel-spectrogram in the input audio.

For the inference, the pre-processed inference data was passed through solely the trained generator to produce the generated face from the input audio and frames. This, together with the audio, was then combined to a video using `FFmpeg`. To obtain a better result, the batch size of the generator was reduced to 64.

5.2.4 Code implementation

The implementations were inspired by the LipGANs author's implementation[1]. However, for this thesis, an up-to-date version of `Pytorch` was used, instead of the outdated `Keras` [52] version used in the original implementation.

The code was structured to have a class for the generator and the critic which used a class for the different residual blocks considers. These were implemented using `Pytorch`'s `nn` framework. The reason for letting the residual blocks be its own class is to implement the skip-connections in the `Pytorch` framework. Further, the contrastive loss was also implemented using the `nn` framework to the class `ContrastiveLoss`. Due to the size of the training datasets, which were bigger than any of the system's video or random access memory, a specific training data handler had to be implemented. This was implemented using `Pytorch`'s `Dataset` framework to the class `TrainDataHandler`. Similarly, this also had to be done for the inference data with the introduction of the `InferenceDataHandler` class. Additionally, to obtain specific metrics, a metric class was used. Also, a utility class was used to save training samples, save metrics, and save the network during or after training. Lastly, the training or inference was initiated from their respective script. A high-level overview of the implementation can be seen in figure 5.1.

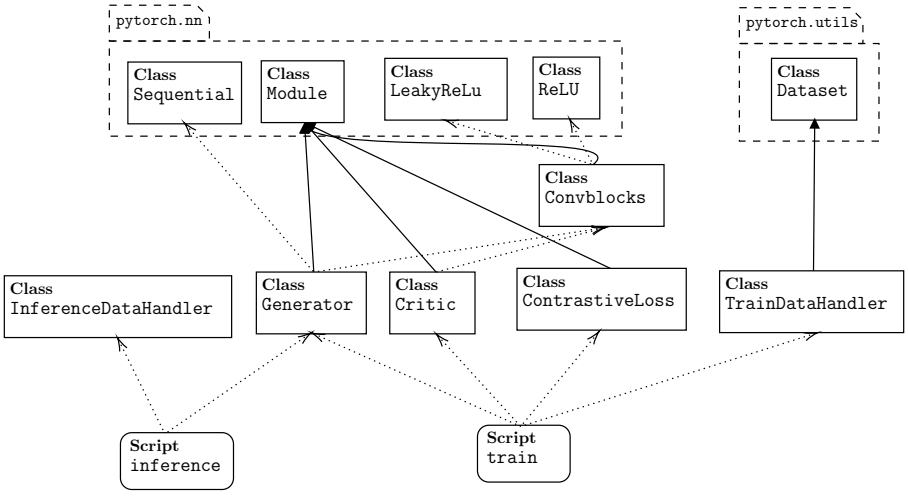


Figure 5.1: High-level overview of the LipGAN implementation. The filled arrows indicate inheritance while the dotted arrows indicate use of a class.

5.3 WGAN-GP implementation

Another model implemented for the task of lip-synchronization was the WGAN model. This model builds on the original WGAN-GP [25] implementation as described in section 3.3 Wasserstein Generative Adversarial Network. The model was trained on the same pre-processed data as all other models, which is presented in section 5.1 Data Pre-processing.

5.3.1 Architectural implementation

The overall architecture mimics the implementation of LipGAN, as seen in subsection 5.2.1 Architectural implementation, with slight modification. Fortunately, the LipGAN architecture used no batch normalization in the critic, which can be a problem for a WGAN as mentioned in subsection 3.3.1 Enforcing Lipschitz constraint. Similarly, the number of trainable parameters stayed the same, at 47424915 where 37087763 are for the generator and 10337152 for the critic.

5.3.2 Training implementation

The major difference between the WGAN model and the LipGAN model is the training process. Firstly, they used the same initial random seed of `numpy.random.seed(10)` for the initializing of the initial weights $\boldsymbol{\omega}_{\text{Init}}$.

Similarly, the WGAN model was trained for 20 epochs with a batch size of 128. However, for each training step, the WGAN model used another approach of synchronizing the generator and critic training. Namely, it trained the critic every step, and the generator every 5th step, using backpropagation. Similar to LipGAN, the critic had an equal probability to be given a real face input S together with time-unsynced audio A' which yields $\mathcal{L}_{\text{audio}}$, or a faked face $\hat{S}$ from the generator together with time-synced audio A which yields $\mathcal{L}_{\text{face}}$.

Since the critic outputs two embeddings E^S and E^A, they were concatenated to one single embedding which could be used as the output of the generator to calculate the loss $\mathcal{L}_{\text{WGAN-GP}}$ as seen in equation (3.15). Therefore, was the gradient penalty $\mathcal{R}_{\text{GP}}$ calculated every step using a penalty coefficient of $\lambda = 10$. Additionally, ADAM was used as optimizer for both networks, using an initial learning rate of $\eta = 10^{-4}$ and the decay parameters $\beta_1 = 0.5$ and $\beta_2 = 0.9$. The inspiration for these ADAM hyperparameters came from multiple WGAN-GP Github repositories. All training parameters can be seen summarised in table 5.4.

Table 5.4: Training parameters used for the WGAN model.

WGAN-model training parameters	
Random seed for $\boldsymbol{\omega}_{\text{Init}}$	`numpy.random.seed(10)`
Epochs	20
Batch size	128
Gradient penalty coefficient λ	10
Initial learning rate η	10^{-4}
ADAM first decay parameter β_1	0.5
ADAM second decay parameter β_2	0.9

5.3.3 Inference implementation

The inference of the WGAN model was done identically to LipGAN, as can be seen in subsection 5.2.3 Inference implementation.

5.3.4 Code implementation

The code implementation for the WGAN-GP model is similar to the LipGAN model, as explained in subsection 5.2.4 Code implementation. However, slight modifications had to be made. Namely, the `ContrastiveLoss` class was no longer needed and could be removed. Additionally, the critic and generator loss had to be changed in the `training` script, and the gradient penalty had to be added. Lastly, the hyperparameters of the optimizer had to be modified.

5.4 Experiments

Experiments in the scope of this thesis were conducted to obtain quantitative results, qualitative assessments, and verify convergence of the models. A brief explanation of each conducted experiment is presented in this section.

1. Firstly, our implementation of LipGAN was analyzed in terms of the loss functions to ensure that the model converges. Inspection of sample images produced during training was also done to assure that the generated samples were convincing faces of satisfying perceptual quality.

2. Our implementation of WGAN-GP was also examined under the same criterion. As a complement to WGAN-GP, a novel model was also implemented and analyzed. This model built on the WGAN-GP but used the L1 reconstruction loss $\mathcal{L}_{\text{re}}$, defined in (3.23), as a loss function in the generator. In this work, we call this model L1WGAN-GP. All hyperparameters from the WGAN-GP model remained the same as in table 5.4.

3. After this, a comparison between the models LipGAN and L1WGAN-GP was done in terms of three different quantitative metrics and qualitative assessment. This was done by using the different models to generate data from the GRIDTest dataset, which was data that the models had not previously seen. The generated data formed a set of samples used as test images while the samples from GRIDTest served as reference images in the metrics.

4. Lastly a qualitative assessment on how different data sets influence the resulting models was done. This featured our implementation of LipGAN, trained on GRID, and the original implementation of LipGAN, trained on LRS2.

The results of these experiments are presented in chapter 7 Results.

CHAPTER 6

Metrics

In this chapter, three quantitative metrics that were used, PSNR, SSIM, and FID-score, are presented. The metrics were used to score the performance of different models and used images as the primary input. The metrics' different properties are presented and evaluated. Lastly, some other considered metrics are discussed.

6.1 PSNR

Peak signal-to-noise ratio (PSNR) is an image processing metric used to measure the difference of power between an image and corrupting noise. The foundation of the metric is built on the *mean squared error* (MSE) between a reference image and a test image [64]. This is defined as

$$\mathrm{MSE}(r, g) = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} \Big(r_{ij} - g_{ij} \Big)^2 \tag{6.1}$$

where r is the reference, g is the test image and both images have dimensions $M \times N$. To calculate the PSNR, a maximum value of an image pixel is required. This may vary depending on how many bits are used in each pixel. In this thesis, all images are coded with 8 bits and thus the maximum value is, $I_{\max} = 255$. The PSNR is defined as

$$\mathrm{PSNR}(r, g) = 10 \cdot \log_{10} \left(\frac{I_{\max}^2}{\mathrm{MSE}(r, g)} \right). \tag{6.2}$$

The PSNR is expressed in a logarithmic quantity using the *decibel* (dB) scale. As the MSE between the images shrinks the PSNR grows larger, thus a high PSNR is desired. This version of PSNR is valid for monochromatic images but is easily extended to include all three pixels in a color image by modifying the MSE to include differences for each of the channels.

PSNR can detect images polluted by noise or other pollutions that affect the values of the pixels [65]. If the PSNR has a low value then the *numerical* difference between the pixel values in r and g is large. This does not however tell much about the image quality itself. Consider the case of a blurry reference and a higher quality test image, then the PSNR would be low yet the image would be of high quality. PSNR also suffers from a low correlation with human evaluation but has the benefit of being simple to calculate [66].

6.2 SSIM

Structural similarity index measure (SSIM) is a metric to measure the similarity between two images. It was introduced by Wang et al. [64] as a metric to determine perceptual image quality that correlates with the human visual system. This is a distinctive difference from PSNR that focuses more on purely the numerical difference between images. SSIM takes three factors into account; luminance distortion $l(r, g)$, contrast distortion $c(r, g)$ and, structure distortion $s(r, g)$. For a reference image r and a test image g these are defined as

$$\begin{aligned} l(r,g) &= \frac{2\mu_r\mu_g + C_1}{\mu_r^2 + \mu_g^2 + C_1} \\ c(r,g) &= \frac{2\sigma_r\sigma_g + C_2}{\sigma_r^2 + \sigma_g^2 + C_2} \\ s(r,g) &= \frac{\sigma_{rg} + C_3}{\sigma_r\sigma_g + C_3} \end{aligned} \tag{6.3}$$

where μ is the expected value, σ is the standard deviation, and σ_{rg} denotes the covariance between the images. Further, C are constants to avoid division by zero. These are defined by

$$\begin{aligned} C_1 &= (K_1 I_{\max})^2 \\ C_2 &= (K_2 I_{\max})^2 \\ C_3 &= C_2/2 \end{aligned} \tag{6.4}$$

where K are small constants and $I_{\max} = 255$ is the maximum value of a 8-bit pixel.

SSIM gives a more perceptual interpretation of differences in images. It captures the distinction of spatially close pixels which are important for the structure of the image. The disadvantage of SSIM compared to PSNR is that it requires more calculations, yet the perceptual performance of SSIM is better.

6.3 FID-score

In general, it is difficult to evaluate the performance of different GAN models in comparison to each other. Salimans et al. [31] proposed a method to compare GANs that uses a pre-trained image classifier network to evaluate the generated distribution $\mathbb{P}_g$ in terms of quality and diversity. This method is called *Inception Score* (IS) and is widely used when comparing GANs [67]. Yet it should be noted that there are some flaws with this method such as that it only works if the evaluated distribution $\mathbb{P}_g$ consists of a class that is known to the classifier. Another problem is that it does not compare the generated distribution $\mathbb{P}_g$ to the desired distribution $\mathbb{P}_r$.

Heusel et al. [68] introduced a new metric called *Fréchet Inception Distance* (FID) that remedies the two mentioned flaws of the Inception score. It builds on the Wasserstein distance, introduced by Fréchet in 1957, that is used to calculate the distance between two Gaussian distributions. To compute this metric an arbitrary feature function, ϕ, is required. The authors suggest using the pre-trained *Inception-v3* network for this task. All images are propagated through the network and on the last pooling layer, the mean μ, and covariance matrices C are calculated. With these components, the FID-score for a reference distribution $\mathbb{P}_r$ and a test distribution $\mathbb{P}_g$ can be expressed as,

$$\mathrm{FID}(\phi(\mathbb{P}_r), \phi(\mathbb{P}_g)) = \left\|\mu_r - \mu_g\right\|_2^2 + \mathrm{Tr}(C_r + C_g - 2(C_r C_g)^{1/2}). \tag{6.5}$$

FID performs well in evaluating distributions generated by GANs in terms of robustness, efficiency, and discriminability [67]. It also poses a more logical choice than the Inception score because it tests both $\mathbb{P}_g$ and $\mathbb{P}_r$. On the other hand, FID lacks consistency due to using an arbitrary feature function, ϕ. Since there is no standard set the measurements are poorly compared to those of other works. It is most common to use the Tensorflow implementation of FID-score when performing tests. In this thesis, `Pytorch` is used and thus a `Pytorch` implementation[2] with the Inception-v3 network was used [69].

6.4 Other considerations

Three quantitative metrics have been presented and were used in the thesis. Two of them focus on image quality and one on network quality. However no metric that measures the actual process of lip-synchronization is presented. Since most of the GANs focus solely on image generation no established metric for lip-synchronization exists. To determine the lip-synchronization performance the most reliable and fair option would be some sort of human evaluation, quantitative or qualitative. A quantitative human evaluation is the *mean opinion score* (MOS) where a reference sample and a test sample are evaluated by humans. These types of evaluations could be used to provide more depth to the analysis of the performance of the different used structures.

Further, it is possible to use the different loss functions as metrics to evaluate the performance of a GAN. However, it is not a good idea to use loss functions, that is used in one of the GANs, as a metric. This is the case since the loss function serves as the objective function and will be minimized by the GAN and thus it would be an unfair match. Yet, the loss functions can be used to prove concepts, such as convergence, and to tune hyperparameters.

[2] `https://github.com/mseitzer/pytorch-fid`

CHAPTER 7

Results

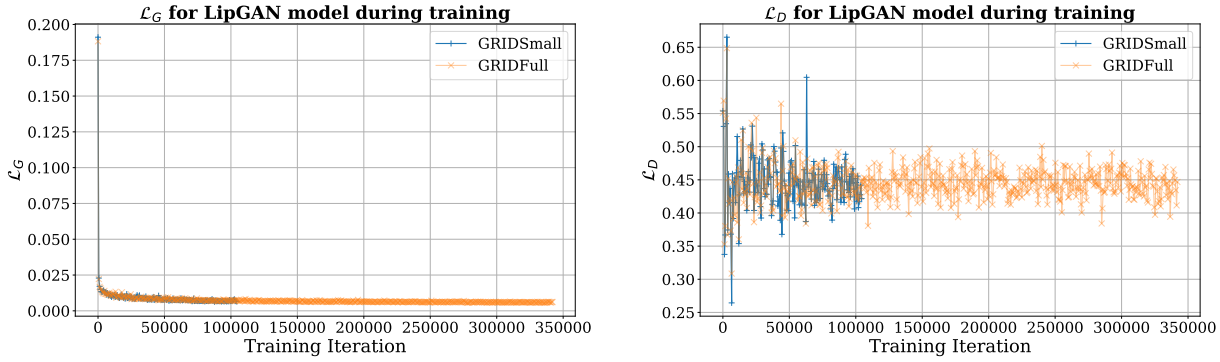
In this chapter, the results of the experiments from section 5.4 Experiments are discussed. Firstly the different models and their individual results and convergences are introduced. Secondly, the models LipGAN and L1WGAN-GP are compared and evaluated using quantitive metrics and qualitative inspection. Lastly, the impact of different datasets on models is examined.

7.1 LipGAN

Firstly, the LipGAN model, implemented as stated in section 5.2 LipGAN implementation, was trained using the GRIDSmall and the GRIDFull datasets for 20 epochs. This took approximately 1 day with 105000 training iterations for GRIDSmall and 3 days with 342400 training iterations for GRIDFull, on both systems.

7.1.1 Losses

During training, the different LipGAN losses were sampled every 600th training iteration for both datasets. The generator loss $\mathcal{L}_G$ together with the critic loss $\mathcal{L}_D$ can be seen in figure 7.1.



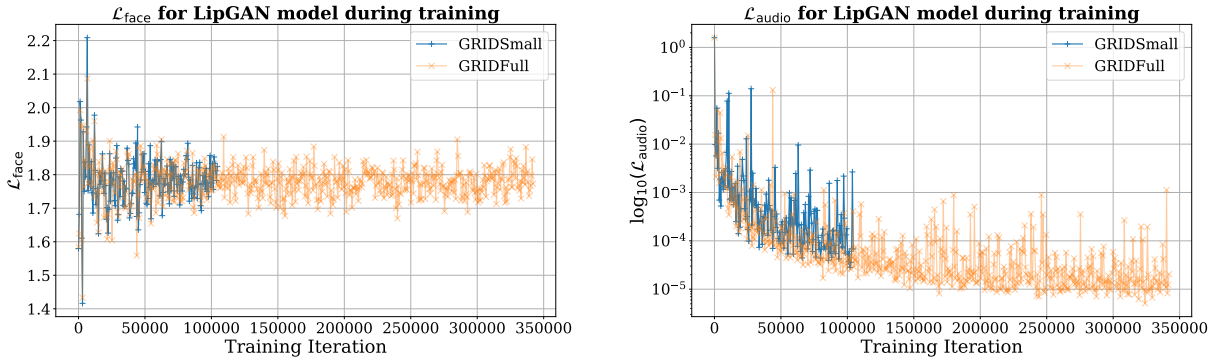
(a) Generator loss $\mathcal{L}_G$

(b) Critic loss $\mathcal{L}_D$

Figure 7.1: Losses during training for the LipGAN model.

As can be seen in figure 7.1, the generator loss converges to around 0, with a minimum loss of $6.0 \cdot 10^{-3}$ for GRIDSmall and $5.7 \cdot 10^{-3}$ for GRIDFull. As for the critic loss, it can be seen to converge around 0.45.

However, some outliers can be seen, which resulted in a search for potential errors in the training samples, although, none were found. Additionally, the loss $\mathcal{L}_{\text{face}}$ with the input of a fake face $\hat{S}$ with real audio A, and the loss $\mathcal{L}_{\text{audio}}$ with the input of a real face S but time-unsynced audio A', were sampled. This was done even if they did not contribute to the critic loss $\mathcal{L}_D$ for that specific iteration. These losses can be seen in figure 7.2.



(a) Faked face $\hat{S}$, synced audio A, loss $\mathcal{L}_{\text{face}}$.

(b) Real face $\hat{S}$, unsynced audio A', loss $\mathcal{L}_{\text{audio}}$.

Figure 7.2: Losses during training for the LipGAN model.

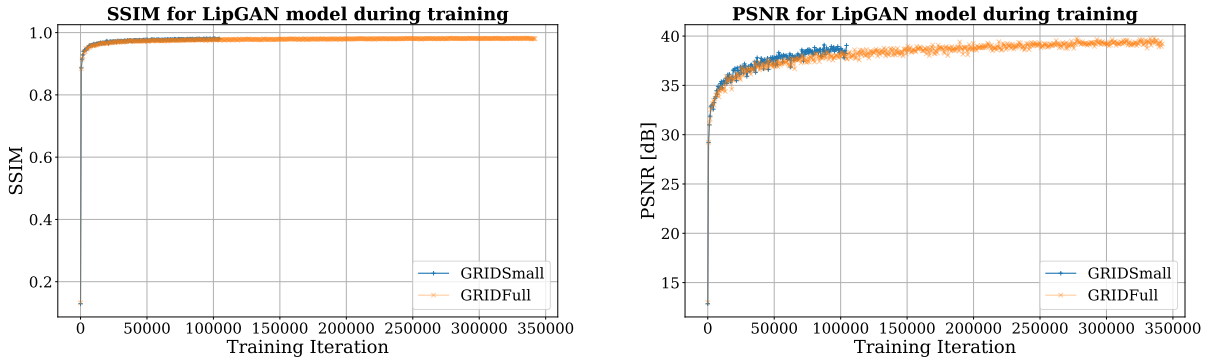
As can be seen, both losses seem to converge, although, this process is slower for $\mathcal{L}_{\text{audio}}$ than $\mathcal{L}_{\text{face}}$. Importantly, it shall be noted that there are some outliers for the loss $\mathcal{L}_{\text{audio}}$.

7.1.2 Sample inspection

Besides saving the losses, some generated sample faces $\hat{S}$ from the generator were saved during the training with GRIDFull. Additionally, the ground truth frame S was also saved. This was done once each epoch, and the speakers were randomly chosen. These samples can be seen in figure 7.3.

Figure 7.3: 20 random generated faces $\hat{S}$ from the generator, together with their corresponding true faces S, from the training of the LipGAN model using GRIDFull. The number denotes the epoch.

As can be seen, most generated faces are of good quality after the first epoch. However, if one looks closely, small differences for some select samples can be seen. For example, in the sample for epoch 7, one can see that the generator produces a face with an open mouth, while the mouth is more closed for the ground truth face. Further, for sample 9, one can notice that the beard has a more blurry appearance than its ground truth counterpart. Lastly, SSIM and PSNR were calculated for the generator's samples, together with the ground truth counterpart, every 600th training iteration. This can be seen in figure 7.4.



(a) SSIM **(b)** PSNR

Figure 7.4: The metrics SSIM and PSNR for the LipGAN model during training. The metrics have been taken for every 600th training iteration.

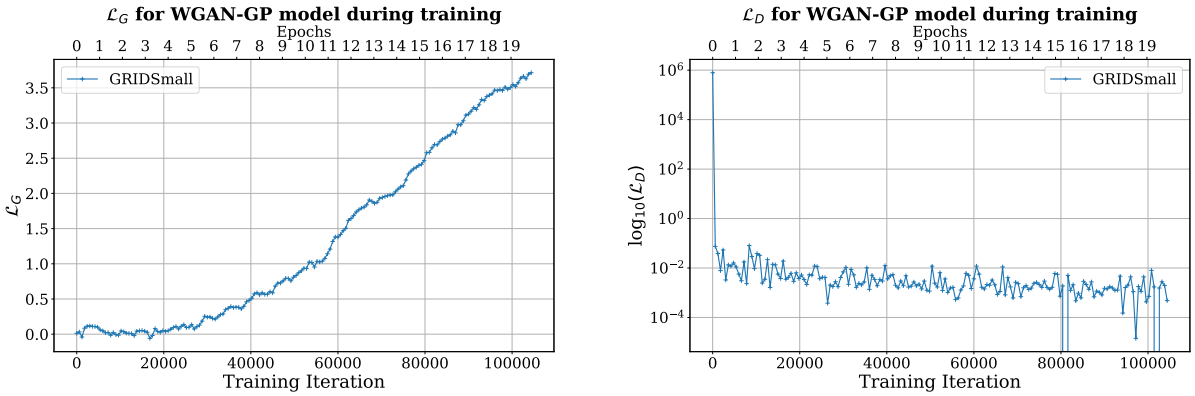
As can be seen for the SSIM, it goes from around 0.13 to 0.98 for both datasets. For the PSNR, it can be seen to start at around 13 dB for both datasets, and end at about 39 dB for GRIDSmall and 40 dB for GRIDFull.

7.2 WGAN-GP

Besides the LipGAN model, the WGAN-GP, as stated in section 5.3 WGAN-GP implementation, was implemented and trained on GRIDSmall for 20 epochs. This model took slightly longer to train, with a time consumption of approximately 1.3 days. Since both models used the same dataset, the number of training iteration stayed intact, at 105000.

7.2.1 Losses

Similar to the LipGAN model, the different losses were sampled every 600th training iteration. The generator loss $\mathcal{L}_G$ and the critic loss $\mathcal{L}_D$ can be seen in figure 7.5.



(a) Generator loss $\mathcal{L}_G$

(b) Critic loss $\mathcal{L}_D$

Figure 7.5: Losses during training for the WGAN-GP model.

As can be seen in figure 7.5, the generator loss $\mathcal{L}_G$ does not seem to converge, while the critic loss $\mathcal{L}_D$ seem to converge quickly. Additionally, the gradient penalty term $\mathcal{R}_{\text{GP}}$, as seen in equation (3.14), was sampled at the same interval. This gradient penalty can be seen in figure 7.6.

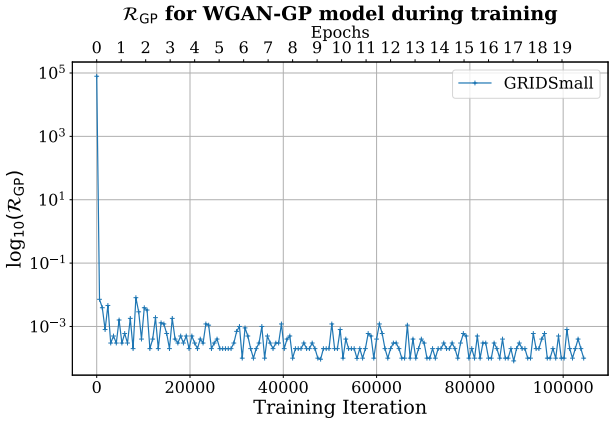


Figure 7.6: Gradient penalty term $\mathcal{R}_{\text{GP}}$ for the WGAN-GP model.

As can be seen, the gradient penalty goes quickly down to approximately 0.

7.2.2 Sample inspection

Lastly, generated sample faces $\hat{S}$ from the generator, together with their ground truth counterpart, were also saved for each epoch. These samples can be seen in figure 7.7.

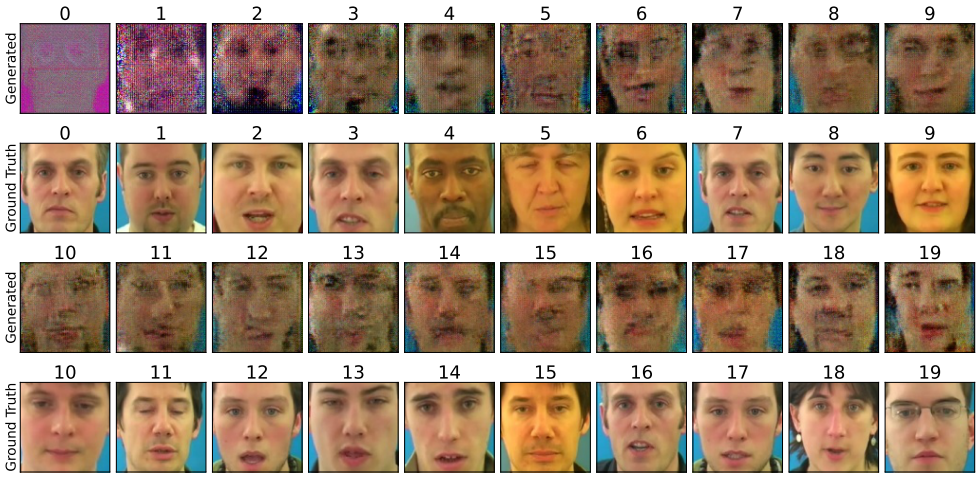
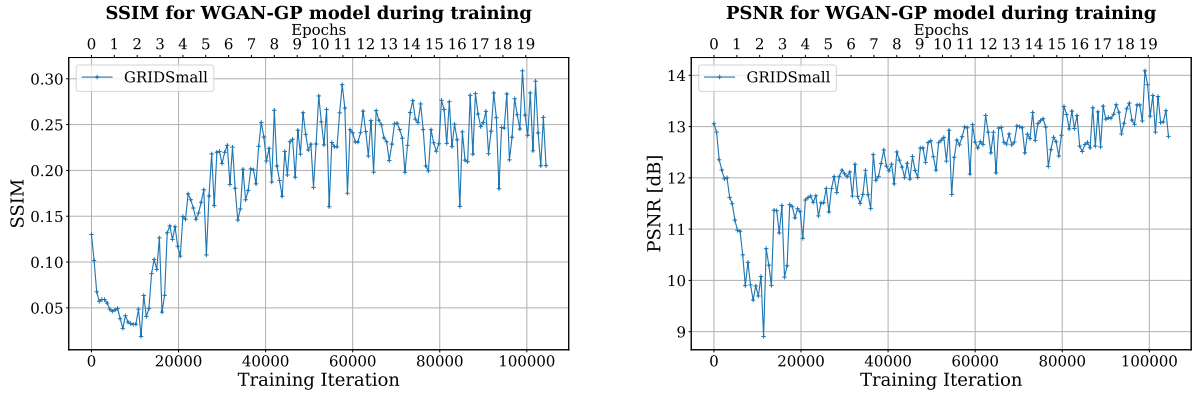


Figure 7.7: 20 random generated faces $\hat{S}$ from the generator, together with their corresponding true faces S, from the training of the WGAN-GP model. The epoch for the samples is denoted by the number above each sample.

Interestingly, these samples have worse perceptual quality and look more like an unrealistic face. Additionally, it shall be noted that the model struggles to produce faces with the correct colors, especially for the first 3 epochs. Similar to LipGAN, the SSIM and PSNR were also inspected for every 600th generated sample. These results can be seen in figure 7.8.



(a) SSIM **(b)** PSNR

Figure 7.8: The metrics SSIM and PSNR for the WGAN-GP model during training. The metrics have been taken for every 600th training iteration.

As can be seen for both metrics, they initially go down, to later go up around epoch 2, and then continue to rise slowly. Yet, it shall be emphasized that they stop increasing rapidly, for both metrics, at around 0.23

for SSIM and 13 dB for PSNR.

Further analysis by inference of the model, showcased that the model seems to produce the same face for each inferences frame. An example of this can be seen in 7.9. This was also experienced on data not originating from the GRID dataset.

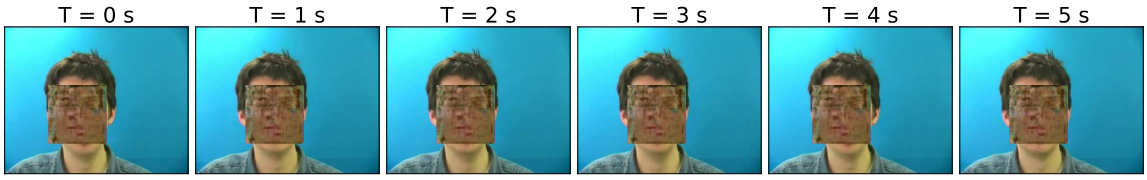


Figure 7.9: Suspected mode failure when doing inference on WGAN-GP. Each image represents a frame at time T, on the inference data.

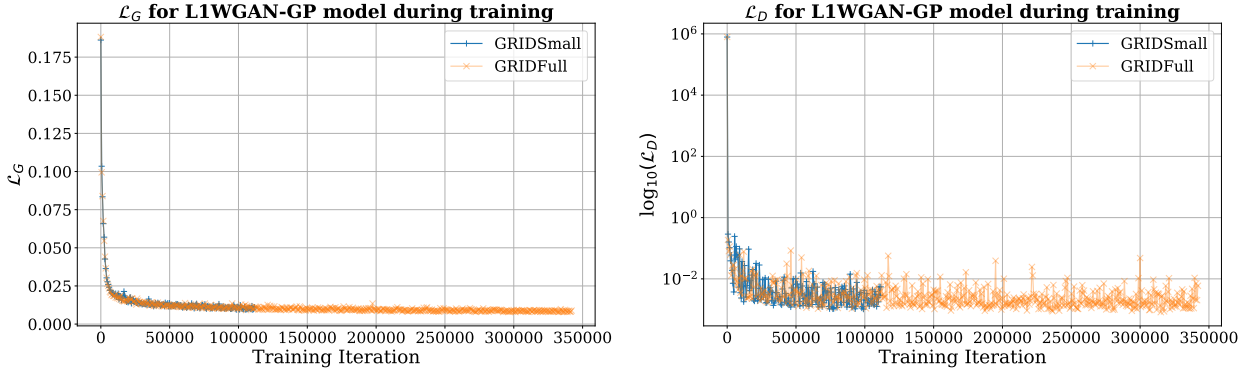
These results suggests mode collapse of the model. Nonetheless, two attempts were made to remedy the suspected mode collapse by retraining the model. Firstly, a similar training rerun was performed since the initial weights of the model could be unfavorable. However, this did not remedy the suspected problems. Secondly, for the other retraining, modifications to the optimizer were made. Namely, by changing the ADAM decay parameter from $\beta_1 = 0.5$ to $\beta_1 = 0$. Unfortunately, did this not remedy the suspected mode collapse. Therefore was the model not trained using the GRIDFull dataset.

7.3 L1WGAN-GP

As mentioned in section 7.2 WGAN-GP, the WGAN-GP model was suspected to experience mode collapse. Also, its generator loss did not seem to converge, as seen in figure 7.5a. This motivated the change of the generator loss, which resulted in the hybrid model L1WGAN-GP, that utilized the L1 reconstruction loss $\mathcal{L}_{\text{re}}$, as explained in section 5.4 Experiments. This model was trained on the GRIDSmall and the GRIDFull dataset for 20 epochs, which resulted in 105000 training iterations and 342400 training iterations respectively.

7.3.1 Losses

Similar to all models, the losses for the L1WGAN-GP was sampled during training for every 600th training iteration. The generator loss $\mathcal{L}_G$ and the critic loss $\mathcal{L}_D$ can be seen in figure 7.10.



(a) Generator loss $\mathcal{L}_G$ **(b)** Critic loss $\mathcal{L}_D$

Figure 7.10: Losses during training for the WGAN-GP model.

As can be seen in figure 7.10, the generator loss seems to converge to a small value, unlike the generator loss in the WGAN-GP model. Also, the critic loss seems to converge. Further, the gradient penalty term $\mathcal{R}_{\text{GP}}$ was also sampled every 600th training iteration and can be seen in figure 7.11.

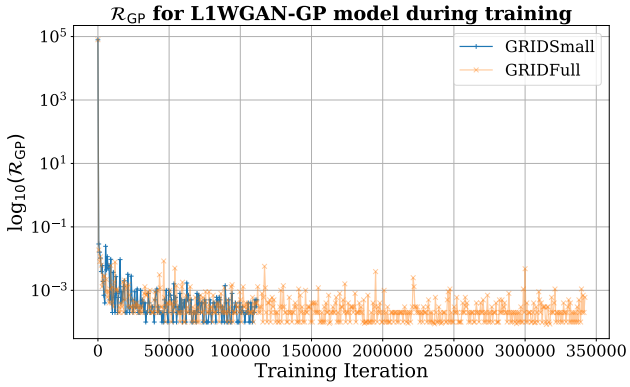


Figure 7.11: Gradient penalty term $\mathcal{R}_{\text{GP}}$ for the L1WGAN-GP model.

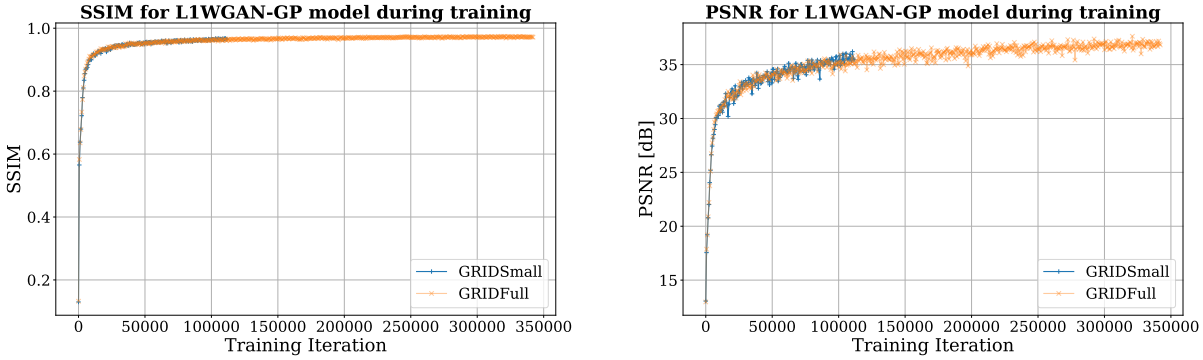
As expected, the gradient penalty term can be seen to go quickly down to around 0.

7.3.2 Sample inspection

Lastly, samples for the generated faces $\hat{S}$ and their corresponding ground truth part S, was saved once per epoch during the training. These samples can be seen in figure 7.12.

Figure 7.12: 20 random generated faces $\hat{S}$ from the generator, together with their corresponding true faces S, from the training of the L1WGAN-GP model using GRIDFull. The epoch for the samples is denoted by the number above each sample.

These samples have a more realistic look than the corresponding samples from the WGAN-GP model. However, compared to the LipGAN model, the L1WGAN-GP samples seem slightly blurry, especially for the early epochs. Luckily, upon inference, the model produced distinct faces for each separate frame, and no sign of suspected mode collapse could be observed. Lastly, the SSIM and PSNR were also calculated for every 600th generated sample from the generator, which can be seen in figure 7.13.



(a) SSIM **(b)** PSNR

Figure 7.13: The metrics SSIM and PSNR for the L1WGAN-GP model during training. The metrics have been taken for every 600th training iteration.

Looking at figure 7.13, it can be seen that the SSIM goes from approximately 0.13 for both models to 0.97 and 0.98 for GRIDSmall and GRIDFUll respectively. Further, the PSNR can be seen to go from around 13

dB for both datasets, to around 36 dB for GRIDSmall, and 37 dB for GRIDFull.

7.4 Comparison Between the Models

In this section, LipGAN and L1WGAN-GP are compared to each other in terms of three quantitative metrics, SSIM, PSNR, and FID, which are explained in chapter 6 Metrics. These were evaluated using unseen test data, in the form of the GRIDTest dataset. Lastly, a qualitative evaluation of the model's performance on test data and inference data was done.

7.4.1 SSIM

The SSIM was evaluated for each of the 44589 data points in GRIDTest and the result of this, in the form of a box plot, can be seen in figure 7.14. Note that the box plot does not include outliers to make the box more visible.

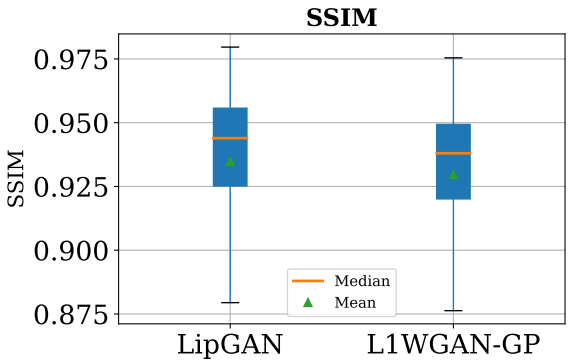


Figure 7.14: SSIM for the models trained on GRIDFull. Outliers have been omitted from the box plot.

As can be seen, LipGAN barely outperformed L1WGAN-GP in terms of both median and mean value. The difference in SSIM was small and both models achieved values close to the maximum possible value of 1.0. A summary of the numeric properties of the acquired SSIM scores can be seen in table 7.1, which confirms the statement above.

Table 7.1: SSIM summary statistics for the models trained on GRIDFull.

Model	Mean ↑	Median ↑	Max ↑	Min ↑
LipGAN	**0.9348**	**0.9439**	**0.9796**	**0.7542**
L1WGAN-GP	0.9296	0.9380	0.9754	0.7052

7.4.2 PSNR

Similar to the SSIM, PSNR was evaluated for all 44589 data points in GRIDTest and the result is presented as a box plot in figure 7.15.

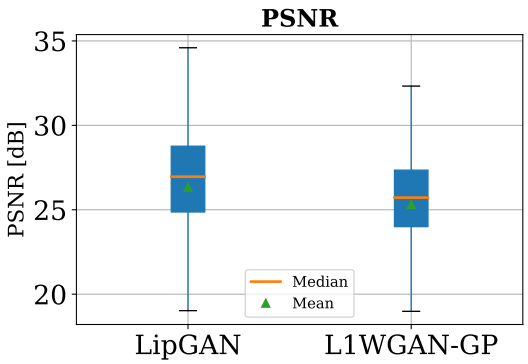


Figure 7.15: PSNR for the models trained using GRIDFull. Outliers have been omitted from the box plot.

In terms of median and mean, LipGAN outperformed L1WGAN-GP. In both cases the spread was large, ranging around 15 dB. As in the comparison of SSIM, the outliers of the box plot were been omitted. A summary of the numerical properties of the box plot can be seen in table 7.2, which again confirms the box plot.

Table 7.2: PSNR summary statistics for the models trained using GRIDFull.

Model	Mean [dB] ↑	Median [dB] ↑	Max [dB] ↑	Min [dB] ↑
LipGAN	**26.34**	**26.96**	**35.35**	**13.67**
L1WGAN-GP	25.32	25.72	34.84	12.81

7.4.3 FID

The FID-score of the 44589 points of reference data in TestGRID and the equal amount of generated test data for each model was compared. The results of the FID score are presented in table 7.3. Interestingly, the L1WGAN-GP model outperforms LipGAN, signifying that, for L1WGAN-GP, the generated data distribution is closer to the reference data distribution.

Table 7.3: FID-score on the models trained using GRIDFull.

Model	FID-Score ↓
LipGAN	15.11
L1WGAN-GP	**14.49**

7.4.4 Qualitative results

Lastly, some qualitative aspects of the two models were examined. This was done by looking at the generated data produced by using GRIDTrain as input. As can be seen by looking at figure 7.16, the model L1WGAN-GP produced images that had visual artifacts. These take shape in many different forms, the most usual

originates around the eyes of the target speaker. The occurrence of artifacts in data produced by L1WGAN-GP was frequent. This problem was not experienced in data produced by LipGAN.

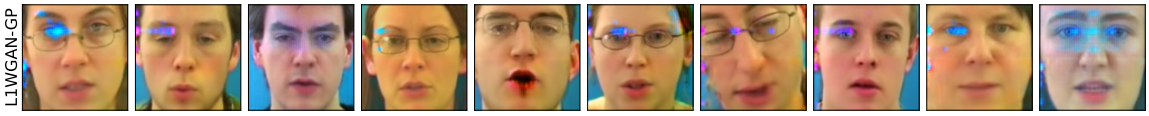


Figure 7.16: Example of visible artifacts from images produced by the L1WGAN-GP model with GRID-Train as input.

Further, upon inspection of data produced during inference, of both models, it was noticed that there was some discrepancy between the generated face and the background. This can be noticed as a visible box surrounding the face. This phenomenon was noticed equally much for both models and is displayed in figure 7.17.

(a) Inference sample from LipGAN.

(b) Inference sample from L1WGAN-GP.

Figure 7.17: Example of a visual box surrounding the face during inference. This phenomenon occurred for both models.

7.5 Impact of dataset

When a model had completed its training, inference was done, with different types of input data in the form of images, video, and audio. The results were compared with the original LipGAN pre-trained model that was trained on the LRS2 dataset. The outcome of inference differs depending on the properties of the used target data. An example of this is showcased in figure 7.18. As can be seen, the LipGAN model trained with GRID did not manage to adapt to the color scheme of this specific image. Additionally, the mouth is misplaced and of incorrect size.

(a) LipGAN trained on GRID. **(b)** LipGAN trained on LRS2.

Figure 7.18: Inference of LipGAN trained on two different datasets. The model's performance is affected by the target data. Image source: [70].

The original LipGAN model performs a good perceptual result whereas our model, trained on GRID, fails to catch the gray colors of the image. It is evident that the performance of model inference is affected by not only its properties but also the target data, and if this is similar to the training data.

CHAPTER 8

Discussion & Further Work

In this chapter, the results from chapter 7 Results are discussed; both in terms of what causes the observation, but also how they can be further explored or mitigated. Secondly, the impact of the dataset used is explored. This is followed by a discussion about the limitations of the model, working with lip-synchronization, flaws in metrics for lip-synchronization, and GANs. Lastly, further work, relating or extending this thesis, is discussed.

8.1 LipGAN

Looking back at the results of the LipGAN model in section 7.1 LipGAN, it can be observed that the model seems to converge and produce satisfactory samples. However, there were some outliers in the loss $\mathcal{L}_{\text{audio}}$, which is produced by a real face S with un-synced audio A' to the critic. As mentioned, there were suspicions that this could be caused by an anomaly in the training data. For example, that `dlib`'s HOG+SVM face detector, used in the pre-processing, could have misidentified a face. However, this seems unlikely since GRID only contains videos of one speaker, faced forward at face level, with the same plain blue background as seen in figure 4.1a. Similarly, there could also be errors in the audio pre-processing caused by `librosa`. However, samples from batches that resulted in outliers in $\mathcal{L}_{\text{audio}}$, were manually inspected, and no anomalies could be observed. Finally, these outliers could be observed for both GRIDSmall and GRIDFull, which had different data batches for the specific iteration since this is done randomly by `Pytorch`'s data handler. This hints that there might be some other unknown phenomena that cause these outliers, or that it can be expected for this specific model and optimizer. Luckily, these outliers did not seem to affect the critic loss $\mathcal{L}_D$ significantly.

As for the samples of generated faces $\hat{S}$ from the training, which can be seen in figure 7.3, it was noted that they are of convincing quality, even in the early epochs. Also, the SSIM and PSNR seem to improve steadily during the training, as highlighted by figure 7.4. Yet, for the sample faces $\hat{S}$, it is noticed that beards tend to be more blurry than their ground truth counterpart, which can highlight that the model might be less convincing for target data of faces with beards. Additionally, it was noticed that for epoch 7, the model failed to generate a closed mouth. This can highlight that the model is not perfect for the mouth-to-sound synchronization, at least in the case of the sample from epoch 7. If this is the case for the model, then the mouth to sound synchronization can potentially be tweaked by experimenting with the time window of the sound corresponding to the frame. This time window can be controlled by the mel-spectrogram width T. Additionally, the frameshift α can also be tweaked to potentially force the

critic to further score on the mouth to sound synchronization. Lastly, there is also the possibility that the number of mel-frequency channels M is too small to provide an accurate representation of the sound, and experiments about this could also be further explored.

8.2 WGAN & L1WGAN-GP

As was shown in section 7.2 WGAN-GP, an attempt to apply a WGAN-GP to the LipGAN architecture was made, but failed to converge, at least for the generator loss $\mathcal{L}_G$. What is interesting, is that the critic loss $\mathcal{L}_D$ converged to a small number, which in theory would suggest that the generated distribution $\mathbb{P}_g$ and the data distribution $\mathbb{P}_r$ would be close in Wasserstein distance W_1 distance. Furthermore, the gradient penalty $\mathcal{R}_{\text{GP}}$ is also close to 0, which would suggest that it enforces the 1-Lipschitz constraint. Yet, this does not seem to result in a good model in practice. However, this might relate to the recent findings that the W_1 distance, using (3.9), is impossible to calculate in practice, as explained in subsection 3.3.2 Remarks about the W_1 approximation. Also, the divergence of the generator loss could be caused by bad parameters, α, β_1, β_2, for the ADAM optimizer. Likewise, bad initial weights $\boldsymbol{\omega}_{\text{init}}$ were also something that could be causing the divergence. However, two reruns were made to experiment with this, but none were successful. Yet, there are endless combinations of these parameters to further try out. Lastly, it shall also be added that it can be the case that it is impossible or almost impossible in practice to adopt a WGAN setup for LipGAN's architecture with its 47 million trainable parameters, and without modifying the generator loss. Especially for the output of the critic, which consists of two separate embeddings E^A and E^S. For our specific approach, we concatenated the two to one single embedding, which we used as the output. Nonetheless, there might be other ways to handle this which can be explored.

As for the samples from the generator during the training, which can be seen in figure 7.8, they are of a more unrealistic perceptual quality. Yet, the same can be said about samples presented by Stanczuk et al. [44], which can be seen in figure 3.4a, and raises the question if this can be expected? The worse perceptual quality can also be seen for the SSIM and PSNR during the training, which is in the range of 0.05-0.30 and 9-14 dB respectively and can be seen in figure 7.8. This can be compared to the other models which during just around $5 \cdot 10^4$ training iterations quickly goes from an SSIM of 0.13 to 0.98 for both models, and a PSNR from 13 dB to 37 dB for LipGAN and to 34 dB for L1WGAN-GP, using GRIDSmall. Also, the spread and variance of these are lower. However, this might also highlight that a WGAN can produce more varied samples, which can be desired. Further, it is also noted that the model seems to experience mode collapse since it produces the same frame for all tested inference data, as seen in figure 7.9. This was also seen for data that is not from GRID. What is interesting is that this can not be seen during the training. It can be that this behavior came late in the training, after the last sample. Additionally, could it also be that our test data have something similar, which the model mode collapses on which it does not for the training data. However, this seems highly unlikely since suspected mode collapse happened for all 3 tested WGAN-GP models. Lastly, all of the mentioned problems could further be analyzed by training it on more data. Unfortunately, due to the time of training GANs, and the fact that we found one remedy by the L1WGAN-GP model, was this not prioritized.

About the L1WGAN-GP model, it was seen to converge for both losses, $\mathcal{L}_G$ and $\mathcal{L}_D$, as seen in figure 7.10. Similarly, the gradient penalty $\mathcal{R}_{\text{GP}}$, which can be seen in figure 7.11, could also be seen to converge to something small, as it is designed to do. Further, the samples taken from the generator during the training seem to have a much more realistic perceptual quality than their WGAN-GP counterpart. However, they could be seen to be slightly more blurry than their LipGAN counterpart. However, some visual artifacts could be seen, see figure 7.16 for examples.

In total, the L1WGAN-GP results show that the LipGAN architecture can employ a WGAN-GP de-

sign, if the generator loss is changed to an L1 reconstruction loss $\mathcal{L}_{\text{re}}$. What is interesting is if it is possible to use another generator loss? This was not tested, but would be interesting to further research. Perhaps, this would remove some blurriness of the samples, and make the samples equal or better than the LipGAN samples.

8.3 Comparison Between the Models

The quantitative metrics used to compare LipGAN and L1WGAN-GP did not give a decisive advantage to either model. LipGAN reigned supreme in the two traditional image metrics, SSIM and PSNR, whereas L1WGAN-GP had a better FID-Score. In all three cases, the results were numerically close to each other, deciding to determine what model was the best a hard task.

In the qualitative assessment, a large number of artifacts were noticed in the data produced by the L1WGAN-GP model. These mostly occurred around the eyes of the target face and took shape as discolored pixels, in many cases matching the surrounding background, see figure 7.16. The artifacts most likely originate from the fact that the L1WGAN-GP model fails to differentiate the background from certain areas of the face. It is difficult to determine why these artifacts appear, an educated guess could be that the generator in L1WGAN-GP only begins updated every fifth iteration. This could result in the model requiring more training and thus our model in its current state would be undertrained. A model that requires more training than what is already used, might be an unfeasible solution due to extensive training time. This of course presumes that the same hardware is used.

Artifacts as they appeared in L1WGAN-GP, are unacceptable for our task since it is of paramount importance that the model delivers a perceptual convincing output. This fact, in combination with a better result in two out of three quantitative metrics, renders LipGAN as the best and preferred model. It should be highlighted that both models solved the task of lip-synchronization adequately good in a subjective manner.

It is troubling that the quantitative metrics did not give a larger discrepancy between the models considering the artifacts produced by L1WGAN-GP. A reason for this fact is that the metrics compare entire pictures and thus a small artifact would not render a large difference in the metrics, even though this would render the perceptual experience ruined. In general, the three metrics used are ill-suited to measure the actual perceptual experience of the videos created by the models. Additionally, their sole focus is image quality. The process of lip-synchronization lacks established metrics and thus it is impossible to measure its performance quantitatively. It should also be mentioned that there is no consensus in what way to quantitatively measure the performance of a GAN. The predominant evaluation of GANs is made qualitatively [67].

A suitable quantitative metric that originates from qualitative assessment, such as mean-opinion-score, should be considered to determine the perceptual results. Yet, it should be highlighted that these types of metrics have some flaws of their own. All things mentioned above contribute to making this problem difficult to measure, which is scientifically troublesome.

8.4 Dataset

When experimenting with the models of LipGAN and L1WGAN-GP, some interesting properties were found. It turned out that the generalized performance of our models, trained on GRID, was very low. As seen in figure 7.18, our model does not work at all on a monochromatic image. The failure of the model is likely occurring due to the dataset's lack of diversity. The GRID dataset was chosen knowingly of this fea-

ture since the product that we are aspiring to contribute to most likely would feature lab environment videos.

It is well known in deep learning that the training data have to be of the same distribution as the data that is supposed to be used for inference. The failure of our model on the Marie Currie image, 7.18, highlights this feature. Yet, there exist solutions to make a dataset more diverse. This could for example be image augmentation, which could include adding noise, turning the photos into monochrome, or zooming in at the mouths. Some of these augmentations could, however, be hard to implement in the current architecture since the bottom half of the image is cropped. Another possible improvement is to increase the resolution of the training images. This could render higher quality images but it would come at the cost of more parameters in the network. Lastly, it shall be mentioned that the Marie Currie example was done using a model trained on LRS2, which has around 29 hours of data compared to GRID's 27.5 hours.

8.5 Limitations

In general, the results of the models were satisfying. However, there were also some limitations to the models and the techniques as a whole. Firstly the pipeline used for creating lip-synchronized videos had a data-driven approach to the problem. This rendered a powerful structure, though that was slow to train and required vast amounts of data. This was amplified by the complexity of the pipeline, which was necessary to produce satisfying results. There were some disadvantages to having a complex model with around 47 million trainable parameters, and the most prominent throughout the thesis was the wall-clock time needed to train, which made hyperparameter tuning a tedious task. Due to the lack of proper hyperparameter tuning, it is hard to draw definite conclusions of the models as their results could be improved or worsened by hyperparameters. A less complex pipeline could include text as audio representation, which could also be used in a TTS solution in the pipeline.

Another limitation of the project is that it is dependant on third-party Python packages, for example for face detection. Since there was little insight on how this works and when it could potentially fail it could lead to problems. Sometimes, the face detection would not catch the entire face which makes samples flawed. Additionally, the models could have problems handling several people in one frame since it is hard to know which person is talking when.

It should also be mentioned that bad generalized performance, in general, is a bad thing for deep learning models. However, in this case, it might be suitable since the models should work well on one or few individuals as were intended in the proposed service.

In general, the lack of quantitative measurements for lip-synchronization is the most prominent limitation to this work. It is hard to quantitatively determine what good lip-synchronization is. If there is no metric, then who decides on what is a good model? This could be matter of opinion. In this thesis, ocular inspection had to be used to verify that lip-synchronization is sufficiently good. It is also hard to display lip-synchronization in anything else than video, making it hard to motivate results in a report. Additionally, the FID-score, which is the prominent metric for GANs, lacks in generalized performance, since the feature extractor ϕ is arbitrary, as explained in section 6.3 FID-score. The Inception-v3 is mostly used, but other feature extractors could also be used. This makes the metric impossible to compare to other results, which have not passed the same feature extractor. Additionally, there is no set amount of samples or any default dataset that should be used to calculate the metric, which also renders different results.

8.6 Further Work

As previously mentioned, there are some further paths to research around the task of lip-synchronization. Since there is not a tremendous amount of research around it, yet, the possibilities of applications are vast. Additionally, there also seems to be a big interest around it among machine learning enthusiasts. Therefore, we have summarized some further points to research. Since working with GANs is a time-consuming endeavor, each one of these points could be a thesis of its own. Some of the points this thesis opens up are the following:

- Adapt further GAN classes to a lip-synchronization baseline. For example, adapt an LSGAN [38], a WGAN-LS [45], and a WGAN-div [43] to the LipGAN architecture and see if those have any advantages. Additionally, do we also feel that it would be interesting to adapt a c-transform WGAN to LipGAN, since contrary to what Stanczuk et al. [44] states about its performance seen in figure 3.4b, we believe that a similar result could be beneficial for the specific task of lip-synchronization for a generic face.

- As mentioned about the impact of the dataset in section 8.4 Dataset, an interesting extension would be to extend our work by training it using image augmentation. Additionally, it would also be interesting to see which image augmentation is beneficial for the task of lip-synchronization. For example, having monochrome augmentation of some training sample would perhaps mitigate the problem seen in figure 7.18, or it could introduce the opposite problem for target data from GRID. Also, it is believed that augmentation of turning the face 180° would have a bad impact since it will invert the mouth movements. Furthermore, augmentation could also be done for the sound, e.g by passing it through some voice synthesizer. In total, it would be interesting to have a conclusion of what augmentation can advantageous, and which can be disadvantageous.

- A related path to the previous point would be to try out other datasets for our model. Perhaps in another language to see how the model adapts to other languages or dialects. Additionally, the impact of the resolution of the data could also be researched, together with the dimension of the input face H, which perhaps could be a remedy to the observed problem for all models of a box around the face. Lastly, it could also be researched if it is possible to use less or more data, and which models are more susceptible to this.

- For our whole intended pipeline, we use text as the initial inputs, which gets passed to a TTS which then is the input to the model. An interesting approach would be to skip the whole TTS part, and only input text to our model. Interestingly, the GRID dataset is transcribed, with timings, and could potentially be used for this. However, this would require some modifications to the pre-processing, since each word does not correspond to a constant amount of frames, which the mel-spectrograms do.

- One important step in our pipeline is the pre-processing. There, facial detection is crucially used to obtain just the face which reduces the input data and eases the training. However, this makes our model very dependent on the facial detection used, and how it crops out the face. Therefore, more research could be used to use other facial detection or bounding boxes than `dlib`'s, which is solely used in this thesis.

- As previously mentioned in section 8.5 Limitations, to our knowledge, there is no established metric for the task of lip-synchronization. Additionally, there is no established unified metric for GANs either; but attempts have been made [71]. It would be interesting to research if one of the many GAN metrics is especially suitable for the task of lip-synchronization. Additionally, a novel metric could be introduced, perhaps using a similar approach of FID by utilizing a pre-trained network.

- Lastly, the task of lip-synchronization, is related to deep fakes, which can bring negative associations as mentioned in section 1.4 Ethical Considerations. However, we believe that it has lots of potential for good purposes, such as personalized text messages, reduce bandwidth in telecommunication, ease

work in the movie industry, and enable automatic video dubbing which in combination with educational videos can enable good and accessible education worldwide. Therefore, security approaches to sort out the malicious uses from the non-malicious uses of deepfakes could be further researched or implemented. For example, a standard organization, with guidelines and security tools, could be set up. This organization could provide certain watermarks to certify certain deepfakes with non-malicious intent. This watermark could be a specific watermark stamp in the video, or maybe some encoded certificate key in the audio spectrum, preferably outside of the hearing spectrum. Additionally, detailed knowledge about the deepfake approaches used could be utilized to improve ways to detect deepfakes, for example by looking for "blurry boxes" around the facial region.

We hope that this will inspire future scientists and help future research in this field.

CHAPTER 9

Conclusion

To conclude, this thesis has showcased that it is indeed possible to create lip-synchronized videos by using a data-driven deep learning pipeline that contains GANs, autoencoders, and residual blocks with skip connections. This was done in the implementation of LipGAN and L1WGAN-GP. Additionally, it is concluded that our implementation of WGAN-GP did not produce satisfying lip-synchronized videos, due to suspected mode collapse. Why this is the case is uncertain. It was remedied by changing the generator loss to the L1-reconstruction loss to form a novel model, L1WGAN-GP.

It was evident that out of the three proposed models, LipGAN outperformed the other models in two of the three quantitative metrics and qualitative assessment. However, the fact that L1WGAN-GP had the best FID-score, despite producing visual artifacts, showcased that there was a lack of proper metrics to evaluate our problem. Additionally, there were no established quantitative metrics to score the lip-synchronization and thus this was solely done by qualitative inspection.

It can be concluded that our models require vast amounts of data. In our tests, the results with GRIDFull outperformed the tests carried out with GRIDSmall in terms of quantitative metrics and qualitative assessment. However, it can not be settled that the amount of data in GRIDFull is in any way optimal to train the model. It might require even more data or training time.

Lastly, the chosen dataset affects the model's inference since it had a bad generalized performance. Thus the dataset chosen to train the model should be chosen with care and consideration. It is of importance that the training data reflects upon the intended use of the model.

Bibliography

[1] S. Suwajanakorn, S. M. Seitz, and I. Kemelmacher-Shlizerman, "Synthesizing Obama: learning lip sync from audio," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, pp. 1–13, 2017.

[2] R. Kumar, J. Sotelo, K. Kumar, A. de Brebisson, and Y. Bengio, *ObamaNet: Photo-realistic lip-sync from text*, 2017. arXiv: 1801.01442 [cs.CV].

[3] X. Yao, O. Fried, K. Fatahalian, and M. Agrawala, *Iterative Text-based Editing of Talking-heads Using Neural Retargeting*, 2020. arXiv: 2011.10688 [cs.CV].

[4] H. Zhou, Y. Liu, Z. Liu, P. Luo, and X. Wang, *Talking Face Generation by Adversarially Disentangled Audio-Visual Representation*, 2019. arXiv: 1807.07860 [cs.CV].

[5] J. S. Chung, A. Jamaludin, and A. Zisserman, *You said that?* 2017. arXiv: 1705.02966 [cs.CV].

[6] L. Chen, Z. Li, R. K. Maddox, Z. Duan, and C. Xu, *Lip Movements Generation at a Glance*, 2018. arXiv: 1803.10404 [cs.CV].

[7] K. R. Prajwal, R. Mukhopadhyay, P. Jerin, J. Abhishek, V. Namboodiri, and C. V. Jawahar, *Towards Automatic Face-to-Face Translation*, Nice, France, 2019. DOI: 10.1145/3343031.3351066. [Online]. Available: http://doi.acm.org/10.1145/3343031.3351066.

[8] O. Fried, A. Tewari, M. Zollhöfer, A. Finkelstein, E. Shechtman, D. B. Goldman, K. Genova, Z. Jin, C. Theobalt, and M. Agrawala, "Text-based editing of talking-head video," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–14, 2019.

[9] T. M. Mitchell, *Machine learning*, 1st. McGraw-Hill New York, 1997, ISBN: 0070428077.

[10] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd. USA: Prentice Hall Press, 2009, ISBN: 0136042597.

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

[12] F. Chollet, *Deep Learning with Python*, 1st. USA: Manning Publications Co., 2017, ISBN: 1617294438.

[13] A. L. Maas, A. Y. Hannun, and A. Y. Ng, *Rectifier nonlinearities improve neural network acoustic models*, Citeseer, 2013.

[14] A. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées," *Comp. Rend. Sci. Paris*, vol. 25, no. 1847, pp. 536–538, 1847.

[15] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[16] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[17] T. Karras, T. Aila, S. Laine, and J. Lehtinen, *Progressive growing of gans for improved quality, stability, and variation*, 2018. arXiv: 1710.10196 [cs.NE].

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[19] Y. LeCun, "Generalization and network design strategies," *Connectionism in perspective*, vol. 19, pp. 143–155, 1989.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[21] H. Purwins, B. Li, T. Virtanen, J. Schluter, S.-Y. Chang, and T. Sainath, "Deep learning for audio signal processing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, May 2019, ISSN: 1941-0484. DOI: 10.1109/jstsp.2019.2908700. [Online]. Available: http://dx.doi.org/10.1109/JSTSP.2019.2908700.

[22] H. K. Elminir, M. A. El-Soud, and L. Abou El-Maged, "Evaluation of different feature extraction techniques for continuous speech recognition," *International Journal of Science and Technology*, vol. 2, no. 10, 2012.

[23] P. Mermelstein, "Distance measures for speech recognition, psychological and instrumental," *Pattern recognition and artificial intelligence*, vol. 116, pp. 374–388, 1976.

[24] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in neural information processing systems*, 2002, pp. 841–848.

[25] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[26] L. Ruthotto and E. Haber, *An introduction to deep generative modeling*, 2021. arXiv: 2103.05180 [cs.LG].

[27] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[28] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2014. arXiv: 1312.6114 [stat.ML].

[29] D. J. Rezende, S. Mohamed, and D. Wierstra, *Stochastic backpropagation and approximate inference in deep generative models*, 2014. arXiv: 1401.4082 [stat.ML].

[30] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, 2014. arXiv: 1406.2661 [stat.ML].

[31] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, *Improved techniques for training gans*, 2016. arXiv: 1606.03498 [cs.LG].

[32] J. Nash, "Non-cooperative games," *Annals of mathematics*, pp. 286–295, 1951.

[33] V. Nagarajan and J. Z. Kolter, *Gradient descent gan optimization is locally stable*, 2018. arXiv: 1706.04156 [cs.LG].

[34] L. Mescheder, A. Geiger, and S. Nowozin, *Which training methods for gans do actually converge?* 2018. arXiv: 1801.04406 [cs.LG].

[35] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2016. arXiv: 1511.06434 [cs.LG].

[36] M. Wiatrak, S. V. Albrecht, and A. Nystrom, *Stabilizing generative adversarial networks: A survey*, 2020. arXiv: 1910.00927 [cs.LG].

[37] W. Fedus, M. Rosca, B. Lakshminarayanan, A. M. Dai, S. Mohamed, and I. Goodfellow, *Many paths to equilibrium: Gans do not need to decrease a divergence at every step*, 2018. arXiv: 1710.08446 [stat.ML].

[38] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, *Least squares generative adversarial networks*, 2017. arXiv: 1611.04076 [cs.CV].

[39] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein gan*, 2017. arXiv: 1701.07875 [stat.ML].

[40] N. Kodali, J. Abernethy, J. Hays, and Z. Kira, *On convergence and stability of gans*, 2017. arXiv: 1705.07215 [cs.AI].

[41] M. Arjovsky and L. Bottou, *Towards principled methods for training generative adversarial networks*, 2017. arXiv: 1701.04862 [stat.ML].

[42] C. Villani, *Optimal transport: old and new*. Springer Science & Business Media, 2008, vol. 338.

[43] J. Wu, Z. Huang, J. Thoma, D. Acharya, and L. V. Gool, *Wasserstein divergence for gans*, 2018. arXiv: 1712.01026 [cs.CV].

[44] J. Stanczuk, C. Etmann, L. M. Kreusser, and C.-B. Schönlieb, *Wasserstein gans work because they fail (to approximate the wasserstein distance)*, 2021. arXiv: 2103.01678 [stat.ML].

[45] H. Petzka, A. Fischer, and D. Lukovnicov, *On the regularization of wasserstein gans*, 2018. arXiv: 1709.08894 [stat.ML].

[46] A. Mallasto, G. Montúfar, and A. Gerolin, *How well do wgans estimate the wasserstein metric?* 2019. arXiv: 1910.03875 [cs.LG].

[47] T. Pinetz, D. Soukup, and T. Pock, *On the estimation of the wasserstein distance in generative models*, 2019. arXiv: 1910.00888 [cs.LG].

[48] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[50] D. Luebke, "Cuda: Scalable parallel programming for high-performance scientific computing," in *2008 5th IEEE international symposium on biomedical imaging: from nano to macro*, IEEE, 2008, pp. 836–838.

[51] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, *Cudnn: Efficient primitives for deep learning*, 2014. arXiv: 1410.0759 [cs.NE].

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

[53] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[54] S. Tomar, "Converting video formats with ffmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

[55] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[56] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Ieee, vol. 1, 2005, pp. 886–893.

[57] D. E. King, *Max-margin object detection*, 2015. arXiv: 1502.00046 [cs.CV].

[58] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "Librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, Citeseer, vol. 8, 2015, pp. 18–25.

[59] M. Cooke, J. Barker, S. Cunningham, and X. Shao, "An audio-visual corpus for speech perception and automatic speech recognition," *The Journal of the Acoustical Society of America*, vol. 120, no. 5, pp. 2421–2424, 2006.

[60] T. Afouras, J. S. Chung, A. Senior, O. Vinyals, and A. Zisserman, "Deep audio-visual speech recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019, ISSN: 1939-3539. DOI: 10.1109/tpami.2018.2889052. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2018.2889052.

[61] K. Kurach, M. Lucic, X. Zhai, M. Michalski, and S. Gelly, *A large-scale study on regularization and normalization in gans*, 2019. arXiv: 1807.04720 [cs.LG].

[62] M. J. Chong and D. Forsyth, *Effectively unbiased fid and inception score and where to find them*, 2020. arXiv: 1911.07023 [cs.CV].

[63] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, *Are gans created equal? a large-scale study*, 2018. arXiv: 1711.10337 [stat.ML].

[64] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[65] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th international conference on pattern recognition*, IEEE, 2010, pp. 2366–2369.

[66] J. Korhonen and J. You, "Peak signal-to-noise ratio revisited: Is simple beautiful?" In *2012 Fourth International Workshop on Quality of Multimedia Experience*, IEEE, 2012, pp. 37–38.

[67] Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, and K. Weinberger, *An empirical study on evaluation metrics of generative adversarial networks*, 2018. arXiv: 1806.07755 [cs.LG].

[68] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, *Gans trained by a two time-scale update rule converge to a local nash equilibrium*, 2018. arXiv: 1706.08500 [cs.LG].

[69] M. Seitzer, *pytorch-fid: FID Score for PyTorch*, https://github.com/mseitzer/pytorch-fid, Version 0.1.1, Aug. 2020.

[70] Wikimedia Commons contributors, *File:maria skłodowska-curie 1903.jpg — Wikimedia Commons, the free media repository*, [Online; accessed 3-June-2021], 2020. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Maria_Sk%C5%82odowska-Curie_1903.jpg&oldid=437189076.

[71] A. A. Alemi and I. Fischer, *Gilbo: One metric to measure them all*, 2019. arXiv: 1802.04874 [stat.ML].

Master's Theses in Mathematical Sciences 2021:E33
ISSN 1404-6342

LUTFMA-3450-2021

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/