Automatic speech recognition of poor quality audio using generative adversarial networks

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering* in Computer and Electronic Engineering at the Potchefstroom Campus of the North-West University

Walter Heymans

27011194


0000-0003-2375-2371

Supervisor: Prof. M.H. Davel

Co-Supervisor: Dr. C.J. Van Heerden

Graduation: May, 2022

Declaration

I, Walter Heymans, hereby declare that the dissertation entitled "Automatic speech recognition of poor quality audio using generative adversarial networks" is my own original work and has not already been submitted to any other university or institution for examination.

Walter Heymans

Student number: 27011194

Signed on the 22^{nd} day of November 2021 at Potchefstroom.

Acknowledgements

This research was performed within the MUST Deep Learning research group of the North-West University, which is a member of the Centre for Artificial Intelligence Research (CAIR) of the Department of Science and Innovation. It was supervised by Professor Marelie Davel and Doctor Charl van Heerden in association with Saigen. It has been a privilege to work with an incredible group of experienced researchers, both at MUST and Saigen.

I would like to thank:

- Ulrike Janke, who managed most of the administrative duties during my studies. She always made sure that we were comfortable and that our workload was balanced.
- My supervisors, Prof. Marelie and Dr. Charl, who guided me through this research and provided invaluable feedback when necessary. Their inputs have been remarkable and are greatly appreciated.
- The Saigen team, who provided the call centre data and access to their server. They assisted whenever a problem occured.
- South African Centre for High Performance Computing (CHPC), who made their facilities available for many of the experiments performed.
- Fellow students at MUST, for their contribution to making the group a success, participating in social activities and providing new perspectives on deep learning.

Lastly, I would like to thank my family for their patience and support. A special mention to Ivaniza, who was always there when I needed her. She stood by my side throughout and ensured that I completed this research with a positive attitude and smile on my face.

Abstract

In this study, we investigate the use of generative adversarial networks (GANs) to improve speech recognition performance of poor quality audio obtained from a real-world source. A GAN is developed to transform acoustic features of noisy audio prior to downstream acoustic modelling. The system utilises a baseline acoustic model trained on good quality data to improve the performance on mismatched data. This is achieved without requiring manual creation of parallel datasets. The practical relevance of the GAN is realised when a strong commercial-grade speech recognition system – which has already been optimised for a given set of conditions – is required to decode new mismatched data. The GAN can then act as a front-end to the existing system.

We compare the GAN-based front-end to multi-style training (MTR) on three datasets in a controlled environment. The GAN system is much faster to train than a comparable MTR system with similar performance. The developed GAN is applied to a South African call centre dataset and achieves consistent improvements over a baseline model. Therefore, this provides a practical approach to improve ASR systems in mismatched environments.

Keywords: *automatic speech recognition, generative adversarial networks, multi-style training, call centre audio, WAV49 encoding*

Contents

List of Figures

List of Tables

List of Acronyms

ASR Automatic speech recognition

CD context-dependent

CNN convolutional neural network

CTC connectionist temporal classification

DNN deep neural network

FFT fast Fourier transform

GAN generative adversarial network

GMM Gaussian mixture model

GP gradient penalty

GPU graphical processing unit

HMM hidden Markov model

FBanks filter banks

fMLLR feature-space maximum likelihood linear regression

LDA linear discriminant analysis

MFCC Mel-frequency cepstral coefficients

MLLT maximum likelihood linear transform

MLP multilayer perceptron

MTR multi-style training

NLL negative log-likelihood

NS-GAN non-saturating GAN

PLP perceptual linear prediction

ReLU rectified linear unit

RNN recurrent neural network

SAT speaker adaptive training

SeER senone error rate

SEGAN speech enhancement GAN

SGD stochastic gradient descent

SNR signal-to-noise ratio

SN-GAN spectral normalisation GAN

TDNN time-delay neural network

WER word error rate

WGAN Wasserstein GAN

Chapter 1

Introduction

We provide background on the study and motivate how it can benefit speech recognition systems. We define the scope as well as the research questions and objectives.

1.1 Background

Automatic speech recognition (ASR) is a technology that enables computer systems to convert spoken language into text using an automated process. ASR has been an active field of research since the 1970s and is still being developed and improved today [1]–[3]. The word error rate (WER), a typical measure of performance for ASR systems, has been significantly reduced over the last few decades. Main factors that contributed to this improvement were recent developments in deep learning, increased computational power of modern computers, specifically graphical processing units (GPUs), and large amounts of collected data [4].

Speech recognition is important for several reasons. Applications include human-to-machine communication, real-time language translation, voice-search, digital assistants,

smart home control and call centre-based speech analytics. Despite many years of research with significant progress in the field, there are still shortcomings that prevent ASR systems from achieving acceptable performance on poor quality audio. Factors related to poor quality audio include various forms of background noise, microphone distortion and encoding. Another issue in ASR is domain mismatch. When a system is tested on data that is different from the training data, the performance is usually poor. This can be caused by a number of factors including different recording environments, different sampling rates, people that speak in different styles and accents, etc. All these factors must be addressed to make ASR as robust as possible to prevent unwanted behaviour.

Most state-of-the-art ASR systems use some form of deep neural network (DNN) to predict word sequences using features extracted from an audio waveform [5]–[8]. Many different techniques have been developed to handle mismatch between training and testing data, a typical issue with poor quality audio. Popular solutions to improve ASR generalisation include improving the acoustic model [5], [6], multi-style training (MTR) [9]–[13], feature enhancement [14], [15], unsupervised learning and self-training [8].

In this work, we investigate the use of generative adversarial networks (GANs) to improve DNN-based ASR systems that use hidden Markov models (HMMs) for sequential modelling on poor quality audio. A GAN is a framework where two neural networks improve by learning from each other [16]. A generator network is tasked to produce new samples from some distribution by using random noise as input. The discriminator network then estimates if the sample came from the true distribution or the generator. The generator learns by using the output from the discriminator to improve the samples it generates. The discriminator is trained by maximising its output for real samples and minimising it for generated samples. The original GAN was developed to generate new images, but has since been applied to a variety of other tasks including speech enhancement [17] and improving ASR robustness in noisy conditions [14].

This research is aimed at poor quality audio obtained from South African call centres. Call centres handle very large amounts of data on a daily basis. Typically, all calls are recorded and stored for future reference, legal purposes and call centre speech analytics. Due to

the large number of calls, the recordings are often compressed for longer term storage. This can decrease the required storage space by up to 20 times. Although compression is beneficial for storage requirements, it is a challenging problem for ASR systems.

1.2 Problem statement

Compression methods, which introduce various forms of noise and artefacts, are used by call centres to save storage space due to large volumes of audio recordings. ASR systems are required to transcribe these compressed audio signals, but fail to give accurate results, since the audio quality is significantly degraded due to background noise, low sampling rates and encoding. Recent developments in deep learning, especially the advent of GANs, suggest that a generative model can be used to improve speech recognition accuracy on poor quality audio. The goal of this study is to investigate GANs to improve ASR systems on poor quality audio obtained from a real-world source.

1.3 Project scope

The initial scope of the research, given the problem statement above, is restricted to a limited number of datasets and architectures.

- **Datasets:** The LibriSpeech corpus contains 1 000 hours of audiobook recordings and is publicly available [18]. The corpus includes labeled subsets of clean and noisy audio. Techniques will primarily be developed on this corpus because a controlled environment can be created to isolate different effects. Public results are widely available, which allows for benchmarking of initial systems.

  An affiliate of the MUST Deep Learning research group, Saigen, has granted secure access to datasets of audio recordings from various call centres in South Africa. Transcribed training and test data are available for nine call centres. Our final goal is to improve ASR performance on these datasets.

- **Architectures:** Baseline systems will be developed using multilayer perceptron (MLP) and time-delay neural network (TDNN) architectures. The effectiveness of GANs will be evaluated against the baseline acoustic models that use MTR. There are two GAN architectures that will primarily be considered. The first is based on the encoder-decoder architecture of a speech enhancement GAN (SEGAN) [17], but instead of operating on raw audio, it will be used on acoustic features. The second architecture is a convolutional neural network (CNN) that retains the dimension of the input features by using padding.

- **Toolkits:** The Kaldi speech recognition toolkit will be used as a basis for all ASR systems [19]. This includes the preparation of data, language models, lexicons, Gaussian mixture models (GMMs), HMMs, decoding and scoring.

  DNN acoustic models will be trained using the Pytorch-Kaldi toolkit [20], an interface between Kaldi and Pytorch, which is an open-source machine learning library for Python. This toolkit allows all the standard features of Kaldi to be used but with the efficiency and flexibility of Pytorch for DNNs.

  A toolkit will be developed that allows efficient training of GANs. The toolkit must interface with Pytorch-Kaldi to integrate developed GANs into an ASR pipeline.

1.4 Research questions

To investigate the capabilities of GANs for improving speech recognition systems, the following research questions were formulated:

- What are the characteristics of the specific real-world audio that is the focus of this study?
- How much can a baseline system be improved for mismatched audio by using MTR?
- What are promising GAN-based approaches that can be applied to this task?
- How much can GANs improve speech recognition systems on poor quality audio?

- How well do GAN-based approaches compare with state-of-the-art MTR in a controlled environment?
- How well do the most promising techniques studied in the controlled environment perform in the call centre environment?

1.5 Objectives

The objective of this study is to investigate the feasibility of using GANs to reduce the WER on poor quality audio when using an ASR system trained on good quality data. The objectives include the following:

- Determine the most important audio characteristics that are relevant for this task. These characteristics can include signal-to-noise ratio (SNR) of background noise, coding effects (WAV49 encoding), dual channel audio compressed to a single channel and sampling rate differences.
- Develop a baseline ASR system and verify it against public results in order to ensure that existing tools are correctly applied.
- Create a controlled environment and isolate the effects that contribute to decreased ASR performance. This environment will act as the main test bed for the GAN development that is to follow.
- Develop an MTR system in the controlled environment in order to have a stronger baseline to compare the GAN system with.
- Determine which approach is best suited for the problem.
  - Review alternative approaches to using GANs in related tasks, including MTR.
  - Determine whether and how well a GAN can be applied to this task, specifically to create preprocessed features prior to downstream processing.
  - Determine whether the current design and/or training process can be improved.

  - If relevant, compare with alternative approaches.

- Apply and evaluate the techniques developed in the controlled environment to the call centre data.

1.6 Research methodology

This section briefly outlines the different parts of the study and how each action was approached and executed.

The study consists of:

- **Literature review:** An in-depth study of DNNs and ASR is performed, followed by more specific architectures for acoustic modelling such as MLPs and TDNNs. The literature review studies the current approaches used to handle mismatched data, including MTR. Finally, GANs are studied, specifically the applications in ASR.

- **Experimental development:** A protocol that ensures consistency in results will be developed to train and optimise ASR systems and GANs. A series of datasets will be created for experimentation in the controlled environment.

- **Software development:** A modular toolkit will be developed that can train, evaluate and compare different types of GANs based on the methods chosen in the experimental development. Baseline models will be integrated into the toolkit for fast and efficient comparison.

- **Experimentation:** First, a baseline ASR system will be developed and verified against public results. The baseline will be improved on mismatched data using MTR in the controlled environment. After the MTR system is evaluated, a GAN will be developed and compared to the baseline and MTR systems. Finally, the developed techniques will be evaluated on the call centre data.

- **Assessment of experimental findings:** After the experimentation of each system is complete (baseline, MTR and GAN), the results will be assessed and discussed. WER will be the main performance metric used to evaluate and compare ASR models. Finally, we will assess how well the results of the controlled environment extended to the call centre data.

1.7 Dissertation overview

This dissertation consists of eight chapters. Each chapter focuses on a section of the research done. A brief outline of each chapter is next.

- Chapter 2 provides background on the main techniques used in this study, namely DNN-HMM ASR systems, acoustic modelling, MTR and GANs. Specific attention is given to the use of GANs for improving robustness in ASR systems.
- Chapter 3 introduces the experimental setup used to train, optimise and evaluate ASR systems. The chapter also provides an overview of the software toolkits and datasets used for experimentation.
- Chapter 4 describes the development of the baseline MLP and TDNN acoustic models. The models are trained on the LibriSpeech corpus and verified against public results that use a similar setup.
- Chapter 5 describes the MTR systems that are developed to improve the baseline on mismatched audio. The effects that different MTR styles have on ASR performance in noisy and encoded environments are investigated.
- Chapter 6 describes the development and training procedure of a new architecture that we refer to as a Guided-GAN, a model that operates on acoustic input features before it is passed to a downstream acoustic model. This GAN is evaluated and compared to the baseline MLP and MTR models.

- Chapter 7 applies the developed Guided-GAN to a real-world task. A proprietary South African call centre dataset is used.
- Chapter 8 concludes the research and summarises the key findings. Possible directions for future work are also discussed.

1.8 Publications

The research performed has provided opportunities for publication. There are two publications that stem from this research:

- "Multi-style training for South African call centre audio" [21], published at the Southern African Conference for Artificial Intelligence Research (SACAIR) 2021. This paper contains selected sections from Chapters 4, 5 and 7.
- "Efficient acoustic feature transformation in mismatched environments using a Guided-GAN" [22] is pending review. It contains sections from Chapter 6 and 7.

Chapter 2

Background

We review literature that is relevant to this study. We provide an overview of ASR systems, domain mismatch, multi-style training and GANs. We investigate related work and identify how this is different from ours.

2.1 Overview

In this chapter, we provide background on the main elements of this study: (a) ASR systems, (b) dealing with domain mismatch in ASR and (c) GANs. In Section 2.2, we provide a brief overview of the two most commonly used types of ASR systems: DNN-HMM and end-to-end ASR systems. We describe the components of an HMM-based ASR system, followed by DNN acoustic modelling architectures. Later in the study, a baseline ASR system will be developed that uses these architectures. In Section 2.3, we review popular approaches used to address mismatched training and evaluation data, a typical issue with call centre audio. Finally, in Section 2.4, an overview of how a GAN works is provided, followed by applications using GANs to create robust ASR systems.

2.2 Automatic speech recognition

There are two main types of widely used ASR systems today. The first is based on HMMs and can be used with GMMs or DNNs [23]. Section 2.2.1 describes the functionality and core components of an HMM-based ASR system. The second type is the much newer end-to-end ASR system that uses one or more DNNs to replace several components in a DNN-HMM ASR system [24]–[26]. Section 2.2.3 provides an overview of end-to-end ASR systems. The research in this study will focus on HMM-based ASR systems, although many of the contributions can be applied to end-to-end systems as well.

2.2.1 DNN-HMM ASR systems

Traditional speech recognition systems were based on HMMs and used GMMs to model the distribution of speech features. DNNs later replaced GMMs because they significantly outperformed the latter [23], [27], [28]. In a DNN-HMM ASR system, an HMM models the sequential dynamics of the speech signal and a DNN estimates the posterior probabilities of the HMM states. There are three main components in an HMM-based ASR system: an acoustic model, a lexicon and a language model. A diagram of such a system is shown in Figure 2.1.

The input to the ASR system is a raw audio waveform. The waveform is split into small frames, usually 25 milliseconds, with some overlap, usually 10 milliseconds. Frames are converted to the frequency domain using the fast Fourier transform (FFT) over time, also called a spectrogram. Audio features are calculated using the spectrogram representation of the speech signal. Many different types of features can be used, for example, Mel-frequency cepstral coefficients (MFCC), perceptual linear prediction (PLP), filter banks (FBanks) or feature-space maximum likelihood linear regression (fMLLR). A popular choice for DNN acoustic models is MFCC or fMLLR features. To provide the acoustic model with more temporal context, several frames of features are spliced together (usually, between 9 and 23 frames).

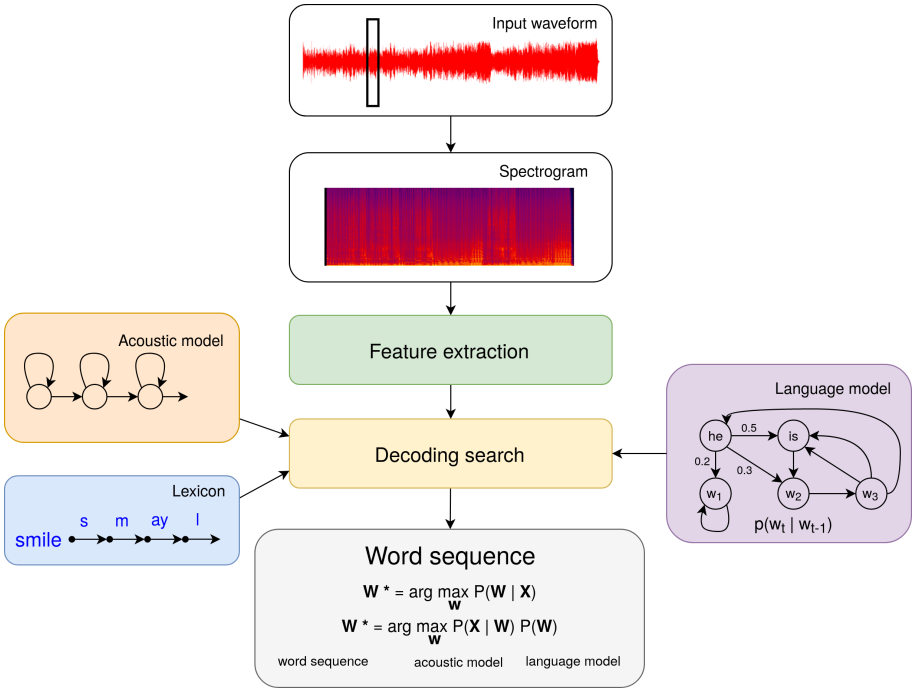

Figure 2.1: Diagram of an HMM-based ASR system.

Speaker-independent systems can be improved by using a speaker-dependent ASR system that captures speaker-specific information [29]. One approach to implement a speaker-dependent system is to use fMLLR input features for speaker adaptive training (SAT). First, a monophone system is trained using 13 MFCC features. The monophone system is used to create frame alignments for the first triphone system. The triphone system is then trained using MFCCs with first (delta) and second order (delta-delta) derivatives to provide information about the rate and acceleration of the speech. Improved alignments are created using the new system. New features are created by using linear discriminant analysis (LDA) to reduce the dimension of the spliced MFCC features to 40. The LDA features are decorrelated using a maximum likelihood linear transform (MLLT) [30]. A speaker-independent triphone system is trained using the MLLT features to create alignments for a SAT model. fMLLR features are then computed to train a speaker-dependent triphone model. Alignments are created for the DNN acoustic model using the speaker-dependent triphone model. After the fMLLR features are computed and aligned, a DNN is trained to estimate the observation probabilities for HMM states when given the features as input. The acoustic model is implemented using a combination of a DNN and

an HMM.

The acoustic model outputs a probability that a phone or senone is observed [23]. A phone is a basic distinctive unit of speech that forms the building blocks to pronounce words. Phones can be used individually (monophones) or modelled in the context of two or more phones. Monophone systems have been mostly replaced by triphone systems that make use of tied context-dependent (CD) states called senones.

Phones or senones are mapped to words using a lexicon. It is a type of dictionary that defines the phonetic pronunciation of words (see the phones for the word 'smile' in the lexicon block in Figure 2.1). Lexicons are usually created by language professionals and will differ from one language or dialect to the next.

At this stage, the system can output a sequence of words, but without context of the rest of the sentence. A language model is used to determine the probability of a word given other words in the same sequence. This improves the decoding accuracy because an acoustic model may give very similar probabilities to homophones, while the language model can distinguish between them based on the context of the sentence. Finally, a decoding search finds the most likely word sequence by using the combined probabilities from the acoustic and language models.

2.2.2 DNN acoustic modelling

The simplest form of DNN used in acoustic modelling is MLP neural networks. An MLP has an input layer, hidden layers and an output layer [31]. Figure 2.2 shows a diagram of an MLP neural network. Each layer is made up of many units, also called neurons. A nonlinear activation function is applied to the units to allow nonlinear functions to be modelled [32]. Popular choices for activation functions include sigmoid, tanh, rectified linear unit (ReLU) and leaky ReLU. All units from one layer are connected to all units in the next layer. Each connection has a weight that is multiplied by its input value before passing it to the next layer. A bias is added to the layer to allow an offset to be learnt to fit the data. The parameters are trained using some form of stochastic gradient

descent (SGD) algorithm (Adam, RMSprop, Adagrad, etc.) to predict the correct output given an input [33]. The loss between the predicted output and ground truth labels are calculated with a loss function. Cross-entropy and mean squared error are popular choices for loss functions. The gradients of the loss with respect to the parameters are estimated using an efficient implementation of the chain rule, called backpropagation [34]. Regularisation is used in DNNs to improve the generalisation by making modifications to the learning algorithm [31]. Many different techniques can be used to regularise a network, including weight decay [35], batch normalisation [36], dropout [37] and early stopping [38].

One problem with MLPs is that they do not have a mechanism to model sequential inputs. TDNNs were introduced to acoustic modelling to handle long-term temporal dependencies in speech [5]. Figure 2.3 shows an example of a TDNN architecture with three layers. Each input frame (labelled with an 'x') has 140 features - 40 MFCCs with a 100-dimensional i-vector appended [5]. A number of narrow context windows are spliced and processed together. This is referred to as sub-sampling because not all inputs are processed together, which decreases the computational cost significantly. The following hidden layer also

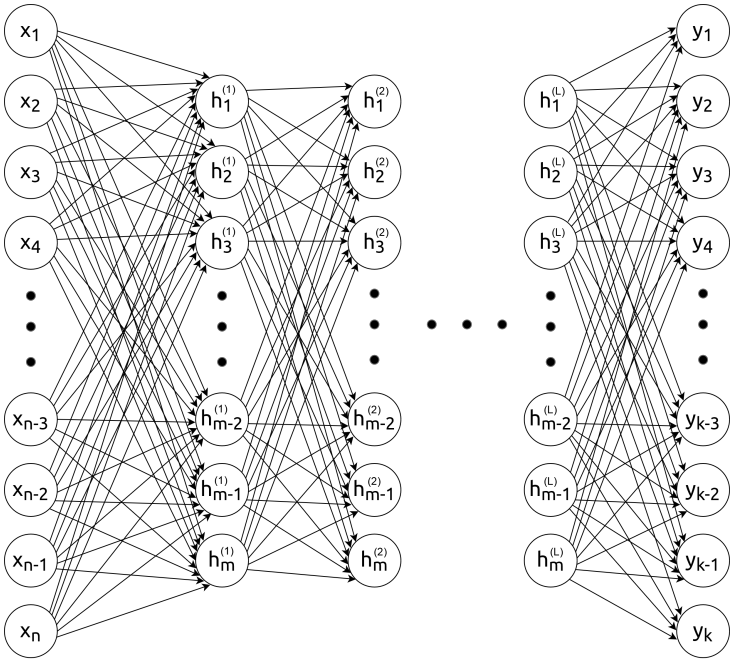

Figure 2.2: Diagram of an MLP neural network. The input layer is labeled with 'x', hidden layers with 'h' and the output layer with 'y'.

preserves the sequence but has a wider context than the previous layer. This continues through all the layers until the output layer is reached where the information from the hidden layers are mapped to senone probabilities. TDNNs are computationally more expensive to train than MLPs, but their performance is usually better [5].

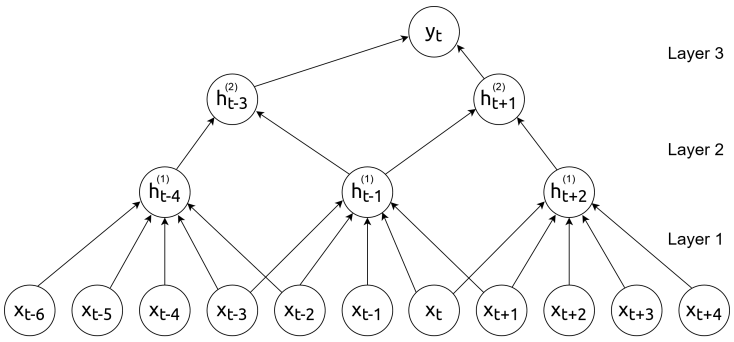

Figure 2.3: Diagram of a TDNN with sub-sampling. The input layer is labeled with 'x', hidden layers with 'h' and the output layer with 'y'.

2.2.3 End-to-end ASR systems

A newer form of ASR, that does not require HMMs, is the end-to-end ASR system. In an end-to-end system, acoustic signals are directly mapped to phones or characters by replacing several components of a DNN-HMM system with one or more DNN [24]–[26]. This drastically reduces the complexity of HMM-based systems. Feature extraction is replaced by an encoder network that learns feature representations by using either an audio waveform or spectrogram as input. Frame alignments are not required, since a recurrent neural network (RNN) with connectionist temporal classification (CTC) [39] loss or an attention mechanism [25] is used to learn alignments between sequences of input frames and output labels. Decoding in an HMM-based system is replaced by a decoder network that maps feature representations to output phone or character sequences. The entire network architecture is trained end-to-end by optimising the probability of the output sequence using SGD [26]. A language model can also be integrated into end-to-end ASR systems using a beam search [25], [26].

2.3 Domain mismatch

ASR systems tend to perform poorer when there is a large mismatch between training and testing data. Factors that contribute to this mismatch include various forms of background noise, microphone distortion, different recording environments, encoding noise, people that speak in different styles and accents, etc. It is difficult for an ASR system to generalise to new audio with different conditions if no attempt is made during training to handle such variability in the data.

In this section, we review approaches that are used to handle mismatched data and increase ASR robustness in noisy environments. We review approaches for domain adaptation and noise robustness in ASR systems (Section 2.3.1) and give an overview of one of the most common methods used to handle mismatch, namely MTR (Section 2.3.2).

2.3.1 Domain adaptation and noise robustness

Domain adaptation is used to adapt a classifier trained in one domain to be effective in another [40]. A large number of methods have been proposed to improve ASR robustness. Most approaches need some prior knowledge about the mismatch to be effective [41]–[43]. A study by Tang et al. investigated different techniques for domain adaptation in distant speech recognition using close-talk data [44]. They compared speech enhancement, MTR and using unsupervised domain adaptation with autoencoders. MTR was the most successful approach, followed by speech enhancement. Another study showed that feature enhancement, noise aware training, MTR and dropout can all improve noise robustness in DNN ASR systems [45]. Unsupervised deep domain adaptation attempts to embed information about the target domain in the classification network by using multi-task learning [46]. The acoustic model is trained to predict senone probabilities and also determine the domain from which the features come. Since the same layers are used in the first part of the network, the network is much more robust to data from different domains.

2.3.2 Multi-style training

MTR is one of the most popular and well-established techniques used to address mismatch in audio for ASR [7], [9]–[13]. This is a data augmentation strategy that aims to transform the training data to be more representative of the testing data and to learn robust representations of the training data. New training datasets are created from an existing set by adding a series of MTR styles. These can include changing the speed and volume [11], speech style [9] or sampling rate [10], adding time and frequency distortions [7] or background noise, and simulating reverberation [13]. The styles are typically chosen without knowledge of the testing conditions and must still be able to handle a wide variety of mismatch. In addition, the number of styles that are added must be considered because the computational cost of training an ASR system increases significantly with each style that is added.

2.4 Generative adversarial networks

Originally, GANs were used for image generation (Section 2.4.1), but have also been applied to many other tasks including speech enhancement [17] (Section 2.4.2) and improving robustness in speech recognition [14], [15], [47] (Section 2.4.3).

2.4.1 Image generation

A GAN framework can generate new samples of a distribution by learning from an existing dataset [16]. Two neural networks, a generator and a discriminator, compete in a min-max game. The generator is trained to produce new samples of a target distribution by using random noise as input (also called a latent vector). Instead of using the target distribution directly, the generator is trained using the output of a discriminator network. The discriminator is trained to determine if a sample came from the true distribution or the generator. The output is maximised for real samples and minimised for generated

samples. After training, in the ideal situation, the generator can accurately produce new samples from the target distribution. Figure 2.4 shows a diagram of a GAN architecture for image generation.

The loss functions for the generator and discriminator of the original GAN, also referred to as a non-saturating GAN (NS-GAN), are given in Eq. 6.4 and 6.5. It is called non-saturating because the generator's loss function was changed to improve the gradients in the early stages of training. Instead of training the generator to minimise the likelihood that the discriminator assigns to generated samples, the generator is trained to maximise the likelihood of the generated samples being real.

GANs have seen many improvements in different ways. The focus has mostly been on changes to the architecture (DCGAN [48], StyleGAN [49], StarGAN [50], U-Net [51], etc.) and loss functions (Wasserstein GAN (WGAN) [52], spectral normalisation GAN [53], least squares GAN [54], etc.) with good performance improvement observed over the NS-GAN first proposed [16]. WGANs were introduced to improve training stability of GANs by leveraging the Wasserstein distance to create a loss function with better theoretical properties [52]. A WGAN requires that the discriminator lie in a space of 1-Lipschitz functions [52]. The original WGAN clipped the weights of the discriminator if they were larger than a certain threshold. This sometimes caused convergence issues that led to undesired

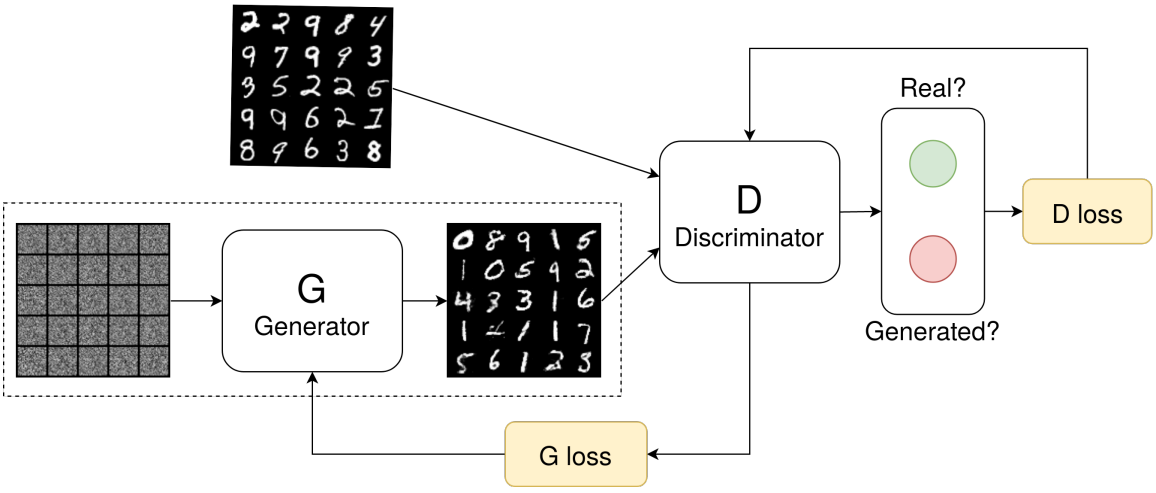

Figure 2.4: Diagram of a GAN architecture for image generation.

behaviour. An improved technique was proposed to satisfy the Lipschitz constraint which utilises a gradient penalty term in the discriminator's loss function [55]. This approach produced better results and resolved the convergence problems of the original WGAN. Spectral normalisation was introduced to rival gradient penalty in WGANs. Spectral normalisation is much cheaper to compute than calculating the gradient penalty, and produces similar results. Despite many of the claimed improvements, different large-scale studies revealed that with good hyperparameter optimisation, most GANs can achieve similar results [56], [57]. Other improvements made to GANs include regularisation, feature matching, mini-batch discrimination, one-sided label smoothing, semi-supervised learning and using different learning rates for the generator and discriminator [58].

Image generation GANs have achieved remarkable results on natural images and human faces [48], [59]. Realistic scenes can be generated to visualise places that do not exist. High-definition human faces generated by a GAN can be difficult to distinguish from real pictures. GANs can also be used to upsample low-quality images much better than traditional methods [60]. Instead of using random noise as input, a super-resolution GAN uses a low-quality image as input and produces an upscaled, high-quality image as output. The discriminator is trained using the original high-resolution images and the images upsampled by the GAN as input.

Conditional GANs provide a mechanism to control the output that the GAN generates [61]. Additional information is provided to both networks during training and is used in the generation process. The output of the generator is conditioned on the information provided. For example, the generator can be used to generate samples of a specific class by providing the class label as an additional input to the generator.

A style-GAN uses a combination of a conditional GAN and a super-resolution GAN to transform images of faces to contain features from another set of images [49]. The input to the GAN is a source image that must be transformed and another target image that is used to condition the input. For example, a picture of a man can be transformed to look like a woman wearing glasses or a child with a hat. GANs are still a relatively new technology and many practical applications are still maturing. Despite its novelty, a GAN

is an extremely powerful tool that can do things that were considered impossible only a decade ago.

2.4.2 Speech enhancement

SEGAN was the first GAN applied to speech enhancement [17]. A noisy speech signal, created by artificially adding noise to clean speech, is passed to a generator network. The generator uses an encoder-decoder architecture with convolutional layers. The encoder downsamples the speech signal to a low dimensional embedding after which a latent vector is added. The decoder then uses transposed convolutional layers with skip connections between corresponding parts in the encoder to upsample the embedding back to the original dimension. The output of the generator is an enhanced speech signal. The discriminator is trained by passing both the clean speech signal and the noisy version as two input channels. The loss is calculated to maximise the output of the discriminator for clean samples and minimise the output for enhanced samples.

SEGAN performed better than a classical enhancement method (Wiener filtering) on almost all metrics evaluating audio quality [17]. The audio quality improvement was measured by a range of objective metrics and perceptual quality determined by human listeners. Although SEGAN improved the audio quality, the improvements did not extend to a reduction in WER for ASR systems [15], [47].

2.4.3 Speech recognition

GANs can improve both DNN-HMM [47] and end-to-end ASR systems [14], [15] by creating indistinguishable representations between clean and noisy audio samples. A clean corpus is augmented by adding noise or room impulse responses to create a new noisy corpus. GANs used for DNN-HMM ASR systems normally operate on acoustic features (log-power spectra, MFCCs) [47] and on log-Mel filterbank spectra or embeddings when used for end-to-end systems [14], [15]. The generator maps the noisy features to the same

representation as the corresponding clean features. The discriminator provides feedback to the generator using the clean and noisy features as input. An L1 or mean squared error loss between clean and noisy samples is minimised which assists the generator with the mapping [14], [15]. This only works when the noisy set is created from the clean set because the loss requires both samples to be of the same frames in a recording. All these approaches rely on the assumption that the test conditions are known. In a mismatched environment, where you have a small training set with matched conditions, none of these GANs can be applied directly. They are only as effective as the conditions that can be created using a clean dataset. Still, GANs such as these that are used to improve features or embeddings have been shown to outperform data augmentation and DNN-based enhancement techniques [14], [47].

A different approach is to use a GAN to fine-tune an end-to-end ASR system [62]. The entire ASR model is treated as a generator network. The ASR system is trained normally until convergence occurs. After training, a discriminator network is tasked to determine if the transcriptions came from the ASR system or are ground truth. Since the ASR model is already trained, the output is similar to the ground truth, which makes the task very difficult for the discriminator. The GAN can improve output transcriptions of an ASR model that has already converged. Adding the GAN-based fine-tuning technique yields consistent WER improvements [62].

2.5 Discussion

In this chapter, we provided an overview of HMM-based and end-to-end ASR systems. We described two network architectures, the MLP and TDNN networks, that will be used in this study. We reviewed approaches to handle domain mismatch and selected MTR as a strong technique that is easy to implement. Finally, we explained how a GAN works and provided an overview of existing applications in the speech recognition field.

Current approaches that use GANs for feature enhancement rely on artificially creating a noisy set from a clean set using some form of perturbation. An L1 or mean squared

error loss is used to minimise the error between noisy samples and the ground truth. There are a few problems with this approach. The mean squared error objective function makes strong implicit assumptions about (1) the independence of temporal relationships (reordering samples does not change the loss), (2) the equal importance of all signal samples and (3) the inaccuracy of describing the degree of signal fidelity (how accurate the target samples are matched) [63]. This means that minimising the loss does not accurately recreate the original signal. Another problem is that artificially creating a dataset does not fully address the data mismatch; it only reduces its effect. None of the GAN-based enhancers or other forms of speech or feature enhancement techniques can operate on a noisy training set that does not have a parallel version to train the enhancer on.

All these approaches try to address a specific condition (reverberation, noise, etc.) rather than mismatched data in general. Our goal is to develop a GAN that does *not* require parallel training data and addresses any form of mismatch between two datasets, not just known conditions that are added to a clean training set.

Chapter 3

Experimental setup

We provide an overview of the tools, protocol and datasets used for system development and analysis.

3.1 Overview

In this chapter, we describe the experimental setup that is used to train, optimise and compare different ASR systems. First, we provide a brief overview of the software toolkits we used and the reasons why they were chosen (Section 3.2). We then introduce the datasets used for experimentation and give a short description of how the data was collected (Section 3.3). Finally, we describe the protocol for DNN acoustic model training and optimisation (Section 3.4).

3.2 Software

We used two public speech recognition toolkits for different stages in our ASR pipeline and another toolkit that was developed to train GANs. All three toolkits were used together and were designed to allow fast and efficient training of DNN-HMM ASR systems.

We first give an overview of the Kaldi speech recognition toolkit used for feature extraction, GMM training, HMM modelling and decoding (Section 3.2.1). We then explain how DNN acoustic models are trained using the Pytorch-Kaldi toolkit (Section 3.2.2). Lastly, we describe a modular GAN training toolkit that was developed to train and optimise GANs for acoustic feature transformation (Section 3.2.3).

3.2.1 Kaldi speech recognition toolkit

Kaldi[1] is an open-source speech recognition toolkit based on finite state transducers [19]. It was written in C++ with the goal to be modular, flexible and easy to understand. Kaldi is well documented and has baseline recipes available for a large number of popular public datasets. This ensures consistency because the code was already properly tested by a large community. The entire speech recognition pipeline can be built in Kaldi, although we use the Pytorch-Kaldi toolkit for DNN training (see Section 3.2.2).

Before an ASR system can be built, the dataset must be preprocessed to be in the correct format for Kaldi. There are a few important files that need to be created. Each utterance needs to get an identifier and must be tied to an audio recording. The person who spoke the utterance must also be identified and linked to the recording. After these mandatory files are created, the features are extracted using the provided scripts. Different types of features can be used separately or together. Monophone and triphone systems are trained using speaker-independent features first and then a triphone SAT model is trained afterwards. Alignments are created using each model before the next one is trained. The last SAT triphone model is used to create alignments for DNN training. All DNNs and

[1] `https://github.com/kaldi-asr/kaldi`

GANs are trained using their respective toolkits.

After the DNN is trained, decoding is done using Kaldi. The outputs of the DNN are stored in a Kaldi ARK file and used by the decoder together with the language model. Lattices are generated and scored using a provided scoring script. The transcriptions are available to view and the WER is calculated using the ground truth labels.

3.2.2 Pytorch-Kaldi

Pytorch-Kaldi[2] is a toolkit for DNN training in a Kaldi ASR pipeline [20]. This allows flexible implementation of DNN acoustic models using Pytorch, a popular open-source machine learning library for Python. Kaldi is used for its efficiency in feature extraction, HMM modelling and decoding. The toolkit is well documented and has support from the authors available if required.

Configuration files are used to setup experiments. The architecture, hyperparameters, datasets and decoding settings can all be set manually. Custom DNN architectures can be defined and used for an acoustic model using Pytorch classes. Example configuration files are provided for some popular public datasets, including the LibriSpeech corpus.

A branch of the toolkit was created to integrate more features. The two most prominent integrations are allowing a GAN to be used on the input features and Weights and Biases[3], an online tool used to track experiments and log data.

3.2.3 GAN training toolkit

A GAN training toolkit was written in Python using the Pytorch library[4]. The goal of this toolkit is to allow efficient training and optimisation of GANs and to make design choices modular by using configuration files.

[2] `https://github.com/mravanelli/pytorch-kaldi`
[3] `https://wandb.ai`
[4] Toolkit will be made available on request.

A number of different GAN loss functions can be specified, or new ones can be added to the toolkit. A list of optimisers is available to choose from. The generator and discriminator networks each have their own adjustable learning rates and update schedules. Weights and Biases are also integrated into the toolkit to allow live tracking of experiments using the online platform. Many different parameters and values are logged to be viewed and compared. This is also very useful for debugging purposes.

The toolkit allows the creation of custom network architectures using Pytorch classes. Pre-trained acoustic models can be imported and used in a number of ways. Different metrics are calculated and displayed to evaluate the GAN using a validation dataset.

The toolkit has GPU support which drastically decreases the computational time required to run experiments. Overall, the toolkit is flexible, easy to use and integrates well with the other two toolkits.

3.3 Datasets

A controlled environment for experimentation is created using the LibriSpeech corpus [18] because it contains a lot of good quality data that is publicly available, and it is one of the most popular benchmarks for new ASR systems (Section 3.3.1). Additive noise is artificially added to the LibriSpeech corpus using two popular noise corpora, QUT-NOISE [64] (Section 3.3.2) and Musan [65] (Section 3.3.3), to simulate a mismatch in noise conditions. A proprietary South African call centre dataset is used to evaluate developed techniques in a real-world scenario (Section 3.3.4). Another corpus containing mixed quality recordings from eight different South African call centres are used for final evaluation (Section 3.3.5).

3.3.1 LibriSpeech corpus

The LibriSpeech corpus contains 1 000 hours of English speech derived from audiobooks that are part of the LibriVox project [18]. The speech is sampled at 16kHz and includes both male and female speakers. Table 3.1 shows the subsets into which the corpus is divided. The corpus was ranked from the best to the lowest quality by sorting speakers based on the average WER for their recordings. The best part of the corpus was selected to create the "clean" subsets and the remaining utterances were used for the "other" subsets. There are two development sets and two test sets, matching the quality of both types of training data.

Four language models are included in the corpus to make it easier for fair comparison of models. The language models were trained using audiobooks after filtering out all texts that were included in the development and test sets. A tri-gram (tg-large) and four-gram (fg-large) language model were trained using all texts. Two smaller language models were created by pruning the tri-gram model with different thresholds (tg-small and tg-med). Usually, the smallest tri-gram model is used for decoding, and then larger models are used to rescore the lattices. A lexicon was created using the 200 000 most common words in the training set, which accounts for about 97.5% of the words in the evaluation sets.

Table 3.1: Data subsets of the LibriSpeech corpus [18].

Subset	**Hours**	**Minutes per-Speaker**	**Female Speakers**	**Male Speakers**	**Total Speakers**
dev-clean	5.4	8	20	20	40
test-clean	5.4	8	20	20	40
dev-other	5.3	10	16	17	33
test-other	5.1	10	17	16	33
train-clean-100	100.6	25	125	126	251
train-clean-360	363.6	25	439	482	921
train-other-500	496.7	30	564	602	1 166

3.3.2 QUT-NOISE corpus

The QUT-NOISE corpus was collected to simulate noisy speech in a wide range of common noise scenarios [64]. It was primarily intended to test voice activity detection algorithms, but it can also be used for data augmentation in speech recognition.

The corpus includes 20 high quality recordings (48kHz, 16 bit) of different noise types across 10 different physical locations. Two separate recordings, each at least 30 minutes in duration, were taken at each location with at least one day between recordings. The locations included a cafe, food court, kitchen, street, car with open and closed windows, indoor pool and an indoor car park. A wide range of noise conditions allow for the simulation of a noisy dataset using clean audio.

3.3.3 Musan corpus

The Musan corpus consists of 109 hours of music, speech and noise recordings [65]. It was developed for voice activity detection and speech/music discrimination algorithms. The corpus is freely available on OpenSLR[5].

The corpus consists of three subsets: music (42.5 hours), speech (60 hours) and noise (6 hours). All recordings are 16kHz WAV files. The noise subset is useful for simulating noisy speech data. There are 929 files in the noise subset with different noise types including dial tones, fax machines, footsteps, crowds of people, cars idling, animal noises, paper crackling, rain, thunder, wind, etc. The noise files in the corpus were downloaded from Free Sound[6] and Sound Bible[7].

[5] `http://www.openslr.org/17`
[6] `https://freesound.org`
[7] `https://soundbible.com`

3.3.4 South African Call Centre corpus

To evaluate ASR systems on real-world data, a proprietary dataset is used, referred to from here as the South African Call Centre (SACC) corpus. The specific call centre compresses calls in three steps: (1) the sampling rate is reduced to 8kHz, (2) dual channel audio are combined to a single channel, and (3) the audio is encoded with WAV49 encoding.

Most high quality ASR systems work with wide-band audio that is sampled at 16kHz. All data in the SACC corpus is narrow-band (8kHz). Narrow-band audio has less frequency information available than wide-band, which tends to hurt ASR systems slightly.

Telephone calls usually have two channels, one for the call centre agent and one for the client (person who called or is being called). Two audio channels use twice as much storage space as a single channel, which is why they are combined to form only one channel. This creates three problems for an ASR system: (1) noise from both channels are present in the new signal, (2) overlapping speech, and (3) speaker confusion. All of these factors contribute to decreased speech recognition performance.

Compressing audio with a codec can reduce the storage space significantly, but also keep most of the original audio quality. There are many different compression methods, such as Free Lossless Audio Codec (FLAC), MPEG Audio Layer III (MP3), Advanced Audio Coding (AAC), Ogg Vorbis, Speex and Opus [66]. The SACC corpus is WAV49-encoded. A full-rate GSM 06.10 codec with a compression ratio of 10:1 is applied to the audio file [67]. It is then saved in a WAV file format resulting in a WAV49-encoded file [68].

The SACC corpus is mostly South African English, but there are occasional non-English words from other official South African languages. Utterances with mostly non-English speech have been removed from the corpus. Table 3.2 shows the datasets in the SACC corpus after undesired utterances were removed. There are 48.8 hours of training data that were originally not encoded, with an encoded version created using Sox[8]. The training, development and test sets were created from the same set by dividing the corpus into three parts. A 1.2 hour held-out test set that was recorded and processed by the call

[8] `http://sox.sourceforge.net`

centre at a later stage is also available for final testing.

Table 3.2: SACC corpus subsets with sampling rate, encoding and total duration.

Dataset	**Sampling Rate**	**Encoding**	**Hours**
train	8kHz	-	48.8
train-e	8kHz	WAV49	48.8
dev	8kHz	-	7.1
dev-e	8kHz	WAV49	7.1
test	8kHz	-	6.2
test-e	8kHz	WAV49	6.2
held-out test	8kHz	WAV49	1.2

3.3.5 Mixed Small Call Centre corpus

The data from one call centre to the next can be significantly different from one another. Not only does the vocabulary change, the recording quality and compression methods are also different. To evaluate our ASR systems on real-world mismatched data, we use a corpus containing data from eight different South African call centres. The corpus is referred to from here as the Mixed Small Call Centre (MSCC) corpus.

Table 3.3 shows the subsets in the MSCC corpus. Three call centres that have different conditions and use different compression techniques were used in isolation. Call centre 1 has single channel recordings sampled at 16kHz, 256kbps. The audio is downsampled to work with 8kHz ASR models. Call centre 2 is encoded with an A-Law codec sampled at 8kHz, 64kbps. Call centre 3 only has four recordings with a total duration of 11.4 minutes. The calls are stereo channel sampled at 8kHz, 256kbps. Because we do not use speech activity detection in our ASR systems, we combined the channels to form a single channel. The subset labeled as "All" contained calls from eight call centres, including the first three.

Table 3.3: Characteristics of the subsets in the MSCC corpus including the total time in minutes.

Dataset	**Sampling Rate**	**Bitrate**	**Encoding**	**Calls**	**Minutes**
Call centre 1 train	16kHz	256kbps	-	8	38.8
Call centre 1 dev	16kHz	256kbps	-	6	20.7
Call centre 1 test	16kHz	256kbps	-	5	15.5
Call centre 2 train	8kHz	64kbps	A-Law	12	43.2
Call centre 2 dev	8kHz	64kbps	A-Law	3	12.8
Call centre 2 test	8kHz	64kbps	A-Law	3	15.3
Call centre 3 train	8kHz	256kbps	-	2	6.6
Call centre 3 dev	8kHz	256kbps	-	1	2.9
Call centre 3 test	8kHz	256kbps	-	1	1.9
All centres train	8-16kHz	64-256kbps	mixed	50	351.2
All centres dev	8-16kHz	64-256kbps	mixed	19	145.7
All centres test	8-16kHz	64-256kbps	mixed	26	188.9

3.4 Experimental procedure

A strict experimental procedure is followed throughout this study to ensure consistency and reproducibility. All experiments are performed using the three toolkits described in Section 3.2. All DNN acoustic models are trained using a specific protocol (Section 3.4.1) and are optimised by searching the same hyperparameters (Section 3.4.2). WER is used to evaluate all ASR systems using public scoring scripts (Section 3.4.3).

3.4.1 DNN acoustic model training

DNN acoustic models are trained using the Pytorch-Kaldi toolkit (see Section 3.2.2). A set of default baseline models are included in the toolkit. We use the toolkit's default training setup with the standard scoring scripts for the LibriSpeech corpus. Everything

in our setup is the same as in the default setup, except that we additionally optimise four selected hyperparameters (batch size, learning rate, language model weight and word insertion penalty).

All networks are trained with the SGD optimiser and a learning rate scheduler that halves the learning rate when the relative improvement[9] on the development set is less than 0.001. The acoustic model is trained with the negative log-likelihood (NLL) loss function to predict HMM state probabilities. All networks are trained for 24 epochs unless specified otherwise; at this point all networks have converged (see Section 4.2.3). Each network is trained with three different random initialisation seeds, unless specified otherwise. We report on the average WER and standard error across seeds. A summary of the hyperparameters is shown in Table 3.4.

Table 3.4: Summary of hyperparameters for DNN acoustic model training.

Hyperparameter	**Value**
Dropout probability	0.15
Loss function	NLL
Optimiser	SGD
Initial learning rate	0.04-0.1
Improvement threshold	0.001
Batch size	128-1024
Epochs	24
Context window	± 5
Acoustic weight	0.1
Language model weight	4 - 23
Word insertion penalty	0.0 - 1.0

[9]Senone error rate is used to measure performance after each training epoch. See Eq. 3.2

3.4.2 DNN acoustic model optimisation

Optimising DNNs can be very time-consuming if too many hyperparameters are used in a search. We found that batch size and learning rate have the largest impact on final system performance, which is why we focus on optimising these two parameters. The batch size and learning rate are optimised on the development set using a grid search. The learning rate is stepped in increments of 0.02 and the batch size is doubled starting from 128. The language model weight and word insertion penalty, also found with a grid search, that gave the lowest WER on the development set are used for the final systems. The language model weight is incremented in steps of 1 and the word insertion penalty in steps of 0.1. The range covered by the searches is wide enough to contain the optimal value. In rare cases where the best hyperparameters are on the edge of the grid, the search is expanded to include another set of values. All other hyperparameters are kept fixed.

The goal of the hyperparameter search is to minimise the WER on a development set. Only the best combination of hyperparameters is used to decode the test set. The range of the parameters used during the grid search is shown in Table 3.4.

3.4.3 ASR evaluation

All ASR systems are evaluated based on WER and use the standard scoring scripts from the corpus where applicable. WER is the sum of all insertion, substitution and deletion errors divided by the total number of words in the corpus:

$$WER = \frac{insertions + substitutions + deletions}{total} \times 100, \tag{3.1}$$

expressed as a percentage.

Another metric that is used during training is senone error rate (SeER). After each epoch, the SeER is calculated to determine if the learning rate should be halved. It is also used for hyperparameter optimisation of GAN networks. The SeER is calculated as follows:

$$SeER = \frac{senones_{correct}}{senones_{total}} \times 100, \tag{3.2}$$

expressed as a percentage. SeER is useful because it can estimate the performance of an ASR system without having to calculate the WER, which is computationally much more expensive.

3.5 Discussion

In this chapter, we reviewed the software toolkits used in this study, the datasets we use to evaluate ASR systems and our training and optimisation protocol for DNN training. The three toolkits are integrated to form a pipeline that allows isolation of different ASR sub-systems. DNN acoustic models and GANs can efficiently be trained using the Pytorch library. It is also possible to integrate a GAN into an existing ASR pipeline. The datasets are chosen to allow comparison with public results, and a controlled environment can be created to isolate different effects that contribute to mismatched audio. We introduced the SACC and MSCC corpora that will be used to evaluate our methods in a real-world scenario. We explained the procedure we follow to train and optimise DNNs to ensure proper convergence.

Chapter 4

Baseline ASR system

The baseline MLP acoustic model is described and compared to public results that use the same setup. We confirm that the networks are trained until convergence occurs and compare three input feature types. A TDNN model is trained and compared to the MLP.

4.1 Introduction

In this chapter, we develop an initial MLP acoustic model that will be used later in Chapters 5 and 6 as the basis for more complex systems. The system is developed using the 100-hour LibriSpeech corpus and compared to public results using the same setup. This is done to confirm that the training and optimisation process is sufficient, as described in Section 4.2. We then evaluate the baseline model on WAV49-encoded audio by manually encoding the LibriSpeech corpus, and develop a new baseline using encoded training data in Section 4.3. This is done to determine how much sampling rate differences and encoding contribute to decreased ASR performance. Finally, in Section 4.4, we train a TDNN acoustic model on the unencoded LibriSpeech corpus using the same setup as our first baseline model.

4.2 System overview

We use the Pytorch-Kaldi ASR toolkit to train a CD-DNN-HMM ASR system on the LibriSpeech corpus. For our baseline system, we use an MLP network architecture for the acoustic model (Section 4.2.1). The network is trained and optimised using the protocol described in Sections 3.4.1 and 3.4.2. The baseline system is compared to public results to confirm that our experimental setup is up to standard (Section 4.2.2). Lastly, we confirm that the network is trained until convergence occurs (Section 4.2.3).

4.2.1 MLP network architecture

We use the default MLP acoustic model of the Pytorch-Kaldi toolkit for the LibriSpeech corpus [20]. The acoustic model is a five-hidden-layer network with 1 024 hidden units per layer. All hidden layers use ReLU activation functions with batch normalisation and dropout with a probability of 0.15. The output layer does not use batch normalisation or dropout and has a softmax activation function. We use fMLLR input features with a temporal context window of 11 frames [69]. The hyperparameters are summarised in Table 3.4.

4.2.2 Baseline results

The baseline MLP ASR system is trained on the LibriSpeech 100-hour training set (*train-clean-100*). The small tri-gram language model (tg-small) is used during decoding and is rescored afterwards using the large four-gram language model (fg-large). The system is tested on all development and test sets of the LibriSpeech corpus. Our baseline MLP model is compared to public results, that use the same network architecture and language models, in Table 4.1. The models in the top section use the tg-small language model and the models at the bottom use the fg-large language model. The WER and standard error of our model is shown across three different random initialisation seeds. Our model achieved similar WERs to both public results on all four sets, slightly outperforming them

(possibly because we optimised the hyperparameters more).

Table 4.1: MLP baseline compared to public results using LibriSpeech 100-hour training set (*train-clean-100*). Average WER and standard error shown over three seeds.

Model	**Evaluation Sets**			
	dev-clean	**test-clean**	**dev-other**	**test-other**
Our MLP, tg-small	**8.88 ± 0.10**	**9.18 ± 0.02**	**23.56 ± 0.01**	**25.16 ± 0.06**
Ravanelli et al. [20]	-	9.60	-	-
Povey et al. [19]	9.19	9.66	26.68	28.64
Our MLP, fg-large	**5.85 ± 0.02**	**6.29 ± 0.02**	**18.63 ± 0.04**	**20.27 ± 0.11**
Ravanelli et al. [20]	-	6.50	-	-
Panayotov et al. [18]	5.93	6.59	20.42	22.52

4.2.3 Convergence

Convergence is an important property to verify when comparing different DNNs. If a model has not yet converged, comparing it with a fully-trained model will influence the results. The default setup for the LibriSpeech corpus in the Pytorch-Kaldi toolkit uses 24 training epochs for the MLP acoustic model. To confirm convergence at this point, we train a model for 30 epochs and measure the SeER on the training and development sets. We also compare the WER of the two models trained for 24 and 30 epochs, respectively.

Figure 4.1 shows the SeER measured on the *train-clean-100* set using the MLP acoustic model during training. The SeER is expressed as a fraction and is measured after every training chunk. The training data is split into 100 chunks to reduce loading time and memory usage. Figure 4.2 shows the SeER on the development set (*dev-clean*). In both figures, the SeER does not consistently decrease after epoch 24. The WER at epoch 24 and epoch 30 differs by 0.01% absolute, which suggests that the network has converged.

In the rest of this study, all MLP networks will be trained for 24 epochs, since we expect the networks to have converged at this point. If it is clear that the validation SeER is still decreasing, we will continue training until the error converges.

Figure 4.1: SeER measured on *train-clean-100* subset of the LibriSpeech corpus using the MLP baseline acoustic model.

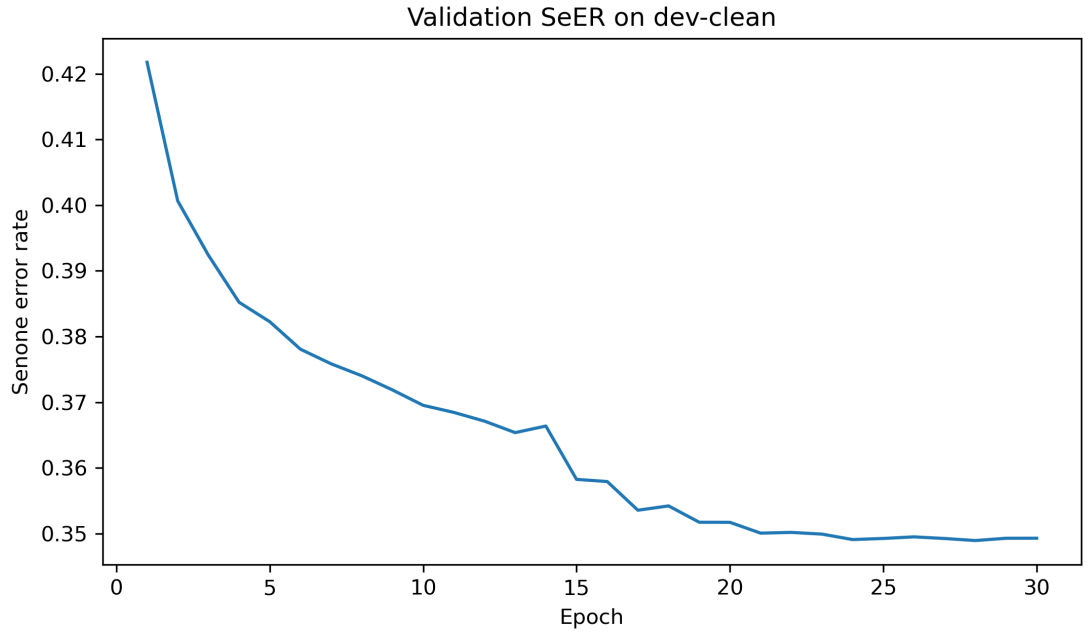

Figure 4.2: SeER measured on *dev-clean* subset of the LibriSpeech corpus using the MLP baseline acoustic model.

4.3 Baseline on WAV49-encoded LibriSpeech corpus

In this section, we create a controlled environment using the LibriSpeech corpus to measure the effect of WAV49 encoding on ASR performance. We first describe how the datasets are encoded (Section 4.3.1). Then, we compare the results of networks using data with different sampling rates (Section 4.3.2). Lastly, we train three networks using different input feature types on WAV49-encoded audio and determine which type performs the best in our setup (Section 4.3.3).

4.3.1 WAV49-encoded LibriSpeech corpus

To measure the effect of WAV49 encoding, we create three encoded datasets using subsets of the LibriSpeech corpus. All audio processing is done using Sox[1]. First, the audio is downsampled to 8kHz. The GSM 06.10 codec is applied to the 8kHz audio, and it is then saved in a WAV file format. After encoding, the file is upsampled back to 16kHz to train a wide-band model. This is done because we want to use a GAN later as a front-end to an existing 16kHz system. The upsampling process used does not attempt to interpolate values to add high-frequency information that was lost during downsampling. Only the existing lower frequency components are retained. Table 4.2 shows the training, development and test sets that are created.

Table 4.2: WAV49-encoded training, development and test sets created using the LibriSpeech *train-clean-100*, *dev-clean* and *test-clean* sets.

Dataset Name	**Source Dataset**	**Encoding**	**Hours**
train-clean-e	train-clean-100	WAV49	100.6
dev-clean-e	dev-clean	WAV49	5.4
test-clean-e	test-clean	WAV49	5.4

[1] `http://sox.sourceforge.net/`

4.3.2 Sampling rate differences on the LibriSpeech corpus

Two sets of networks are trained on the *train-clean-100* and *train-clean-e* datasets using 16kHz and 8kHz audio, respectively. We also downsampled the clean training and development sets to 8kHz to measure the performance difference caused by WAV49 encoding when unencoded, narrow-band audio is used for training.

The WER results are shown in Table 4.3. All data in the top section is used at 16kHz (training and development sets as well), while the data in the bottom section is used at 8kHz. The *train-clean-100* model performed the best on the *dev-clean* set for both frequencies. When this model (*train-clean-100, 16kHz*), trained using 16kHz unencoded audio, is evaluated on the encoded development set that was upsampled to 16kHz, the WER drastically increased to 19.32%. By downsampling the unencoded training data (*train-clean-100, 8kHz*), the result improved to 11.44% WER. This network, trained by using only unencoded data, performed only slightly worse than the *train-clean-e* model (11.44% vs 11.21% WER). The large difference in WER between the 16kHz *train-clean-100* and 8/16kHz *train-clean-e* models can be reduced significantly by downsampling unencoded training data. Encoding the training data only provided a slight improvement over the 8kHz *train-clean-100* model of 2.0% relative WER.

We observed that most of the mismatch is a result of the difference in sampling rate and not due to encoding. The improvement achieved when downsampling unencoded training data was 40.1% relative WER and only 2.0% when encoding the train set. While we expected the encoding (after downsampling) to have a significant effect on performance, the results here indicate that this is not the case.

4.3.3 Input feature selection

Input features can play a significant role in ASR systems. Some features are fast to compute, while others are more robust to noise [70]. We evaluate three different feature types on the WAV49-encoded LibriSpeech development set (*dev-clean-e*). All features we

Table 4.3: WER results of models with different sampling rates on *dev-clean* and *dev-clean-e*. Average WER and standard error is shown over three seeds.

Train Set	**Sampling Rate**	**dev-clean**	**dev-clean-e**
Wide-band			
train-clean-100	16kHz	**8.88 ± 0.10**	19.32 ± 0.11
train-clean-e	16kHz (upsampled)	10.71 ± 0.05	**11.02 ± 0.03**
Narrow-band			
train-clean-100	8kHz	10.29 ± 0.03	11.44 ± 0.04
train-clean-e	8kHz	10.76 ± 0.02	11.21 ± 0.03

compare use a temporal context window of 11 frames (5 left, current and 5 right).

The first feature type we use is the default fMLLR features in the Pytorch-Kaldi toolkit. We compare them to MFCCs with delta and delta-delta coefficients (39 features) and a concatenation of 40 dimensional MFCCs with a 100-dimensional i-vector, because popular TDNN networks use such features [5]. Saon et al. used the features in a similar way and they found that a 100-dimensional i-vector performed the best [71]. The 40 dimensional MFCCs are spliced together to form a 440-dimensional feature vector. The i-vector is extracted over 7 frames and is appended to the MFCCs, resulting in a feature vector with 540 features. This is then used as input to the MLP acoustic model for senone prediction.

To compare the different input feature types, we train networks using the *train-clean-100* and *train-clean-e* sets. We evaluate the networks on the *dev-clean* and *dev-clean-e* sets. This is done for all three feature types. The WER results are shown in Table 4.4. The networks that used fMLLR features performed the best on both development sets for each training set. When using WAV49-encoded training data, the performance on the unencoded development set was still good compared to the other feature types. Both MFCC networks performed very poorly on mismatched datasets[2]. Based on this, we only use fMLLR features for the rest of the experiments.

[2]Opposite type of dataset that was trained on.

Table 4.4: WER results of models with different input features on *dev-clean* and *dev-clean-e*. Average WER and standard error is shown over three seeds.

Train Set	**Input Features**	**dev-clean**	**dev-clean-e**
train-clean-100	MFCC + Δ + ΔΔ	10.00 ± 0.04	26.96 ± 0.03
train-clean-100	MFCC + i-vector	8.96 ± 0.03	27.03 ± 0.08
train-clean-100	fMLLR	**8.88 ± 0.10**	**19.32 ± 0.11**
train-clean-e	MFCC + Δ + ΔΔ	16.48 ± 0.08	12.58 ± 0.03
train-clean-e	MFCC + i-vector	17.05 ± 0.34	12.05 ± 0.04
train-clean-e	fMLLR	**10.71 ± 0.05**	**11.02 ± 0.03**

4.3.4 Test set results of baseline MLP systems

The two MLP baseline models that achieved the best WER on the development sets were used to decode the test sets. Both models used the *tg-small* language model and one set of 100-hour training data. Table 4.5 shows the results for the two models on the *test-clean* and *test-clean-e* sets. Similar to the results on the development sets, the *train-clean-100* model performed the best on the unencoded test set (*test-clean*), and the *train-clean-e* model performed much better than the *train-clean-100* model on the encoded test set (*test-clean-e*).

Table 4.5: WER results of best MLP baseline models on *test-clean* and *test-clean-e*. Average WER and standard error is shown over three seeds.

Train set	**test-clean**	**test-clean-e**
train-clean-100	**9.18 ± 0.02**	19.15 ± 0.11
train-clean-e	10.95 ± 0.05	**11.10 ± 0.02**

4.4 TDNN baseline

TDNNs can model long-term temporal dependencies much better than MLP acoustic models, but they are computationally more expensive. In this section, we train a TDNN

using the *train-clean-100* subset of the LibriSpeech corpus to compare with our baseline MLP model. The network architecture is described in Section 4.4.1 and the results are presented in Section 4.4.2.

4.4.1 TDNN network architecture

Based on the network used by Peddinti et al. [5], we train a TDNN with sub-sampling. TDNN networks generally have four to five layers with a number of hidden units between 512 and 1 536 [72], [73]. We use a four-layer TDNN with 1 024 hidden units per layer. All layers used batch normalisation, ReLU activation functions and dropout with a probability of 0.2. When the probability is lower than 0.2, the network does not generalise as well - the training SeER is much lower than the development SeER. A linear layer with softmax activation function produces the final output. The network is trained and optimised as described in Sections 3.4.1 and 3.4.2, except that the context window contains 21 frames instead of 11. We use fMLLR input features instead of MFCCs with i-vectors because we confirmed that fMLLR features perform better on this dataset for an MLP network and we do not use the model in an online fashion. Originally, MFCCs with i-vectors were used for TDNN networks because fMLLR features require two passes to calculate, which means you need future information to calculate the values for the current frame. Table 4.6 shows a summary of the hyperparameters for our best TDNN network.

4.4.2 TDNN baseline results

The TDNN model was trained using the *train-clean-100* subset and evaluated on all four evaluation sets of the LibriSpeech corpus. The results are shown in Table 4.7. The TDNN performed better than the MLP baseline on all sets when using the large language model and better on the two clean sets when using the small language model.

The training time of the TDNN is approximately 19 times longer than that of the MLP baseline when using the same training data. The main reason is that the context window

of the TDNN is much wider and the network has more parameters. Since the performance gains with a TDNN in the current experimental setup is fairly small, and the computational cost is high, we will continue using the MLP baseline model in further experiments.

Table 4.6: Hyperparameters for TDNN network with the lowest WER on the development set.

Hyperparameter	**Value**
Hidden layers	4
Units per layer	1024
Dropout probability	0.2
Loss function	NLL
Optimiser	SGD
Initial learning rate	0.08
Batch size	512
Epochs	24
Context window	± 10

Table 4.7: WER results of TDNN compared to MLP baseline using the LibriSpeech 100-hour corpus (*train-clean-100*).

Model	**Evaluation Sets**			
	dev-clean	**test-clean**	**dev-other**	**test-other**
MLP, tg-small	8.88 ± 0.10	9.18 ± 0.02	**23.56 ± 0.01**	**25.16 ± 0.06**
TDNN, tg-small	**8.52**	**8.98**	24.10	25.89
MLP, fg-large	5.85 ± 0.02	6.29 ± 0.02	18.63 ± 0.04	20.27 ± 0.11
TDNN, fg-large	**5.69**	**6.15**	**18.11**	**19.80**

4.5 Discussion

In this chapter, we described the development of a baseline MLP ASR system and verified its performance against public results. We confirmed that our model is trained for long enough to ensure proper convergence. Another baseline model was developed on a WAV49-encoded LibriSpeech corpus to improve the clean baseline model on an encoded test set. Both MLP systems were evaluated on the WAV49-encoded test sets which set the benchmark for the models developed in Chapters 5 and 6. We also developed a TDNN model that outperformed the MLP baseline on most sets, but were computationally too expensive to adopt for continued experiments.

We confirmed that sampling rate differences are responsible for a much larger mismatch than WAV49 encoding. We expected that encoding would have a larger effect, but the results indicate differently. Three different feature types were compared on WAV49-encoded audio. fMLLR features performed the best for both unencoded and encoded data.

Chapter 5

Multi-style training

A multi-style training model is developed to improve the standard baseline results on mismatched data. Different conditions are isolated and their effect on ASR performance is measured.

5.1 Introduction

MTR is a data augmentation technique that aims to address mismatch in data and learn robust feature representations. MTR systems usually use multiple versions of the same training dataset by changing the audio in some way. The method that is used to change the data is referred to as a style. MTR is when you use multiple styles to train a DNN. Most state-of-the-art ASR systems use some form of MTR [5], [7], [72].

In this chapter, we develop an MTR ASR system to improve the baseline from Chapter 4 as a stronger competitor for the GANs developed in Chapter 6. In Section 5.2, we provide an overview of our MTR system and how we perturb clean training data. To investigate the effects that different MTR styles have on ASR system performance, we

create a controlled environment using the LibriSpeech corpus in Section 5.3.

5.2 System overview

The primary objective of our MTR system is to reduce the WER on a noisy WAV49-encoded test set. The characteristics of this set is very similar to call centre conditions. We use the baseline MLP acoustic model architecture developed in Chapter 4 for our MTR system. The networks are trained and optimised using the same protocol as our baseline, as described in Sections 3.4.1 and 3.4.2. We investigate three perturbation types - noise, speed and volume - to make our system more robust to noisy WAV49-encoded audio.

A noisy set is created by adding additive noise to clean data. Mathematically, a noisy speech signal is the sum of the original speech signal and a noise signal at each time step:

$$y(t) = x(t) + n(t) \tag{5.1}$$

where $x(t)$ is the original clean speech signal, $n(t)$ is the noise signal and $y(t)$ is the output noisy speech signal. Noise is added for the entire duration of the clean speech signal. If a noise file has a longer duration than the speech signal, a random portion of the noise file is selected with the same duration as the speech signal. If a noise file is shorter than the speech signal, the noise is repeated until the duration of the speech signal is matched. This ensures that the entire output signal has noise present.

Speed and volume perturbation are applied to a signal with a change of 10% and 20%, respectively. The speed is either increased or decreased with equal probability - approximately half of the utterances have a slower speed compared to the original set and the other half have a faster speed. Volume is handled in a similar way. Both speed and volume perturbation were done using Sox, the same tool used by Ko et al. [11].

The MTR models we train use combinations of noise-, speed- and volume-perturbed datasets. During training, the datasets are combined without shuffling the utterances.

All MTR models are trained for 24 epochs, similar to the baseline, despite more training data being used. Convergence normally occurs at an earlier epoch (around epoch 16) compared to models using only one training dataset.

5.3 Multi-style training in a controlled environment

We use the LibriSpeech corpus to create a controlled environment for MTR experiments (Section 5.3.1). The corpus is well suited for this, since a large portion of the corpus has been labeled as "clean", meaning that the recordings do not have much noise. When using these clean audio recordings, noise and encoding can easily be added to simulate call centre audio conditions.

We trained many MTR systems by using different combinations of datasets to see which combination improves the baseline the most (Section 5.3.2). The results are further confirmed by evaluating the best MTR models on a noisy WAV49-encoded test set (Section 5.3.3). In a final experiment, we increase the network capacity to determine if the performance of MTR systems can be improved in this way (Section 5.3.4).

5.3.1 MTR datasets for controlled experiment

We use the 100-hour subset of the LibriSpeech corpus for training data (*train-clean-100*) and the small tri-gram (tg-small) language model for faster decoding and to make the comparison of different acoustic models more efficient. To simulate call centre conditions, we add background noise to the clean development (*dev-clean*) and test (*test-clean*) sets using the QUT-NOISE corpus [64] with a SNR of 5 dB. For our training data, we add noise using the Musan noise corpus [65] to create an artificial mismatch in noise conditions. An artificial mismatch is created between training and test data because the noise corpus used to create the test set has different types of noise than the corpus used to perturb the training data. For both noise corpora, we randomly add a noise file to each utterance for the total duration of the utterance. We also encode these sets with WAV49 encoding and

reduce their sampling rate to 8kHz.

Table 5.1 shows the training datasets we use for MTR. Different combinations are used to see the combined performance impact on the development set. Table 5.2 shows the development and test sets that are created from the *dev-clean* and *test-clean* subsets, in a similar manner as the training set but with fewer conditions.

Table 5.1: Multi-style training datasets created using the 100-hour clean LibriSpeech subset (*train-clean-100*).

Dataset Name	**Encoding**	**Noise Corpus**	**SNR**	**Speed**	**Volume**
train-clean-100	-	-	-	-	-
train-clean-8k	-	-	-	-	-
train-clean-e	WAV49	-	-	-	-
train-noisy-e-5	WAV49	QUT	5	-	-
train-clean-e-s	WAV49	-	-	10%	-
train-clean-e-v	WAV49	-	-	-	20%
train-clean-e-sv	WAV49	-	-	10%	20%
train-musan-e-5	WAV49	Musan	5	-	-
train-musan-e-10	WAV49	Musan	10	-	-
train-musan-e-15	WAV49	Musan	15	-	-
train-musan-e-20	WAV49	Musan	20	-	-
train-musan-e-15-s	WAV49	Musan	15	10%	-
train-musan-e-15-v	WAV49	Musan	15	-	20%
train-musan-e-15-sv	WAV49	Musan	15	10%	20%

Table 5.2: Development and test datasets created using the LibriSpeech *dev-clean* and *test-clean* sets.

Dataset Name	**Source Dataset**	**Encoding**	**Noise Corpus**	**SNR**	**Hours**
dev-noisy-e-5	dev-clean	WAV49	QUT	5	5.4
test-noisy-e-5	test-clean	WAV49	QUT	5	5.4

5.3.2 MTR results in controlled experiment

In this experiment, we analyse the effect of different MTR styles on a set with mismatched noise conditions. Different combinations of noise, speed and volume perturbation are used for training data. All data, except for the 8kHz *train-clean-8k* set, is upsampled to 16kHz using Sox. The upsampling process is the same as in Section 4.3.2. Table 5.3 shows the WER results on the development set (*dev-noisy-e-5*).

Similar to the results in Section 4.3.2, downsampling the clean training data improved the WER, although the improvement is much less than before. Adding WAV49 encoding to the training data improved the relative WER of the *train-clean-8k* model by 15.6%, which is much more than observed in Section 4.3.2.

Speed and volume perturbation were applied to the clean encoded training data. Different combinations of datasets were evaluated. A small improvement in WER (0.8% relative) was observed when using only speed perturbation. None of the other combinations resulted in a notable improvement - the *train-clean-e-v* and *train-clean-e-sv* networks performed worse than without perturbation. This may be because the training data already captures a large range of speed and volumes, or that the development set does not vary much in terms of speed and volume.

Noise perturbation was applied to clean training data using four different SNR values and the sets were encoded afterwards. The performance with the 15dB network was much better than the rest, despite the fact that the development set used an SNR of 5dB. The different noise corpora, QUT-NOISE versus Musan, can explain the difference. Energy in the noise files are distributed differently, so the SNR values are not directly comparable. The *train-musan-e-15* model showed a 17.0% relative improvement over the *train-clean-e* model. This emphasises how important matched training and test conditions are. It is crucial to use the correct SNR for noise perturbation because it has a large influence on system performance.

We used the best noise-perturbed training dataset and added speed and volume perturbation. When using speed and volume perturbation in any combination, the WER did

Table 5.3: WER results on *dev-noisy-e-5* using training datasets with different styles. Average WER and standard error is shown over three seeds.

Model	Datasets	Size	Dev WER
Variations of clean set			
train-clean-100	train-clean-100	1	36.46 ± 0.14
train-clean-8k	train-clean-100 @ 8kHz	1	33.23 ± 0.21
train-clean-e	train-clean-e	1	**28.06 ± 0.03**
Speed and volume			
train-clean-e-s	train-clean-e + s	2	**27.83 ± 0.02**
train-clean-e-v	train-clean-e + v	2	28.21 ± 0.07
train-clean-e-sv	train-clean-e + sv	2	28.25 ± 0.10
train-clean-e-s-v	train-clean-e + s + v	3	28.04 ± 0.13
Noise			
train-musan-e-5	train-musan-e-5	1	29.29 ± 0.34
train-musan-e-10	train-musan-e-10	1	27.29 ± 0.12
train-musan-e-15	train-musan-e-15	1	**23.30 ± 0.06**
train-musan-e-20	train-musan-e-20	1	26.64 ± 0.09
Speed, volume and noise			
train-musan-e-15-s	train-musan-e-15 + s	2	24.09 ± 0.05
train-musan-e-15-v	train-musan-e-15 + v	2	23.97 ± 0.11
train-musan-e-15-sv	train-musan-e-15 + sv	2	24.34 ± 0.02
train-musan-e-15-s-v	train-musan-e-15 + s + v	3	**23.80 ± 0.09**
train-musan-e-15-s-v-sv	train-musan-e-15 + s + v + sv	4	23.89 ± 0.08
Matched noise			
train-noisy-e-5	train-noisy-e-5	1	**19.75 ± 0.04**

not improve the result compared to using only additive noise. A similar phenomenon was observed by Ko et al. [11] on the ASpIRE corpus, where they only observed an absolute WER improvement of 0.1%. However, their result still improved and did not get worse like ours. We further investigate this in Section 5.3.4.

Finally, we trained a network using a noise-matched training dataset, *train-noisy-e-5* that

also uses the QUT-NOISE corpus for perturbation. Compared to the best MTR model, a relative improvement of 15.2% WER was achieved using this model. This shows that MTR has many shortcomings when test conditions are significantly different from the training set. When encountering unseen environments on a new test set, most systems will probably struggle to do well.

5.3.3 MTR results on test set

Up to this point, all results were reported on the development set. Table 5.4 shows the WERs on the test set (*test-noisy-e-5*) using six selected models that performed the best in each category on the development set. The results on the test set are similar to those on the development set. The best MTR model of all the combinations tested is the one that used only noise perturbation with the SNR value that performed the best on the development set. The relative difference in WER between the best MTR model and the *train-clean-100* model is 33.5%. MTR can clearly reduce the WER significantly if the conditions are properly chosen, but MTR still performed 16.5% worse than the *train-noisy-e-5* model with matched conditions.

5.3.4 MTR using larger networks

By adding speed and volume perturbation, you also add more training data. It is possible that the network with only 1 024 hidden units is too small to capture the larger data distribution. Increasing the network capacity should help the models that are using more training datasets to generalise better and possibly give an advantage to the speed and volume perturbation networks.

Using the same training and optimisation protocol described in Sections 3.4.1 and 3.4.2, we train three networks with 2 048 hidden units per layer instead of 1 024. The average WERs are shown in Table 5.5 over three seeds for both network sizes and the dimensions of the hidden layers are shown in brackets. The performance of all models improved, but

Table 5.4: WER results on *test-noisy-e-5* using training datasets with different styles. Average WER and standard error is shown over three seeds.

Model	Datasets	Size	Test WER
Variations of clean set			
train-clean-100	train-clean-100	1	36.92 ± 0.18
train-clean-e	train-clean-e	1	29.16 ± 0.12
Speed and volume			
train-clean-e-s	train-clean-e + s	3	28.61 ± 0.11
Noise			
train-musan-e-15	train-musan-e-15	1	**24.54 ± 0.22**
Speed, volume and noise			
train-musan-e-15-s-v	train-musan-e-15 + s + v	3	24.87 ± 0.08
Matched noise			
train-noisy-e-5	train-noisy-e-5	1	**20.48 ± 0.03**

the model using speed and volume perturbation improved more than the model using only noise perturbation. The larger capacity benefits the model with more training data, but also the model using only encoded training data. The difference between the small and large networks for the *train-musan-e-15* model is almost negligible.

This experiment confirmed the hypothesis that the MTR models required more capacity to outperform the noise-perturbed network. There is, however, an increased computational cost when doubling the number of hidden units on top of the three times more training data. This becomes an important trade-off to consider if computational resources are limited, which is why we continue to use the MLP models with 1 024 hidden units going forward.

5.4 Discussion

In this chapter, we developed an MTR system that significantly improves the baseline model on noisy WAV49-encoded audio. We showed that MTR can have a very positive

Table 5.5: WER results on *dev/test-noisy-e-5* using MLP acoustic models with 1 024/ 2 048 hidden units per layer. Average WER and standard error is shown over three seeds.

Model	Size	Dev WER	Test WER
Encoded			
train-clean-e (1 024x5)	1	28.06 ± 0.03	29.16 ± 0.12
train-clean-e (2 048x5)	1	26.82 ± 0.08	27.74 ± 0.14
Noise			
train-musan-e-15 (1 024x5)	1	23.30 ± 0.06	24.54 ± 0.22
train-musan-e-15 (2 048x5)	1	23.15 ± 0.10	24.55 ± 0.04
Speed, volume and noise			
train-musan-e-15-s-v (1 024x5)	3	23.80 ± 0.09	24.87 ± 0.08
train-musan-e-15-s-v (2 048x5)	3	**22.81 ± 0.06**	**24.34 ± 0.08**

effect on ASR performance, given that the styles are chosen appropriately. Speed and volume perturbation can slightly reduce WER in some scenarios, but are computationally much more expensive than using only noise perturbation.

The two styles that provided the best improvement on the WAV49-encoded LibriSpeech corpus were (1) encoding training data and (2) noise perturbation, if there is no noise in the training set. However, if the SNR of the added noise is completely different to the test conditions, it can hurt the system. The best MTR setup significantly outperformed the clean baseline on the test set, but still performed worse than the noise-matched model. With proper network capacity, MTR does not hurt system performance, although it may be computationally very expensive.

Chapter 6

Generative adversarial networks

A generative adversarial network-based method is developed to transform acoustic features of poor quality audio. The new transformed features are used to improve speech recognition performance of a baseline system on mismatched data.

6.1 Introduction

In this chapter, we develop a GAN to improve an ASR system on poor quality and/or mismatched audio. Our approach is implemented on an ASR that uses a CD-DNN-HMM, but is applicable to a wider variety of systems. The approach is modular because the GAN operates on acoustic features rather than any specific part of the DNN-HMM system. Our goal is to support an existing commercial-grade ASR system to be able to process audio from different conditions by preprocessing the audio, rather than changing the core system itself.

An overview of the developed GAN system is given in Section 6.2. We evaluate the GAN on three different datasets in a controlled environment in Section 6.3. Finally, we analyse

the improvement that the GAN makes in more detail in Section 6.4.

6.2 System overview

The goal of our GAN is to improve acoustic features of noisy audio to be better classified by an acoustic model. To achieve this, we train a GAN using two datasets, one using clean audio and the other noisy audio. We use unpaired audio samples[1] because artificially creating a dataset has certain limitations. It is not easy to create a diverse training set when only using clean audio and samples of known noise conditions. There are many variables, and recreating them all can be very difficult. This was clearly shown in Section 5.3.3, where the model using a training dataset with matched noise conditions significantly outperformed one that was manually created using unmatched noise conditions. When you have two different datasets, one clean and the other noisy, you can use samples from both datasets to train the GAN. Using a noisy training set with better matched conditions to the test set will improve the output of the GAN significantly.

Noisy audio features are used as input for a generator network. The generator is tasked to create transformed features that are better classified by an acoustic model. To train the generator, a discriminator network is used to evaluate the quality of the generated features. Clean and noisy features are separately provided to the discriminator as input. The discriminator is trained to maximise its output for clean samples and minimise its output for noisy samples. The ground truth labels (clean or noisy sample) are used to train the discriminator to better distinguish between them. The generator tries to maximise the output that the discriminator assigns to generated features. The training process can be viewed as a min-max game that approaches a Nash equilibrium between the two networks.

GANs typically only use two networks for training - a generator and a discriminator. When a generator is tasked to transform or enhance the input, some mechanism is necessary to force the generated sample to approach the desired values. For example, if a generator is

[1] Clean and noisy samples are not of the same utterance.

tasked to remove noise from an audio file, the original clean audio file must also be used to tell the generator what the ground truth must be. This is normally achieved by adding the L1 or mean squared error loss between the clean and noisy sample to the generator's loss function. Minimising this loss will then cause the generated output to approach the desired values.

This approach is only valid when using paired clean and noisy samples, meaning the noisy dataset must be created by adding noise to the clean dataset because the samples need to match for the loss to be useful. Another possibility is to use a conditional GAN that uses senone labels as input to the generator and discriminator. This assists the generator to create improved features that are of the same class as the noisy features. There are two factors that limit this approach. During evaluation, you do not have the true label of the features. Without the label, the generator cannot be conditioned on the label. If the senone labels are only used in the discriminator, a lot of examples of all classes are required during training. This is only possible when a large enough training dataset is used since small sets might not include enough examples of all classes.

Our approach is to use the output of a trained acoustic model to guide the GAN during training. The NLL loss is calculated using the output of the acoustic model and the ground truth senone labels. Using this loss, the parameters of the generator are adjusted to improve the classification accuracy of the acoustic model. This forces the generator to produce features that are classified better. Figure 6.1 shows a diagram of the training process for a Guided-GAN. When the system is used as front-end for an ASR system, only the generator network is used. The noisy features are given to the generator which then transforms them and passes it on to the acoustic model for classification.

The Guided-GAN is intended to be used when a baseline system trained on good quality data already exists. The system is then improved on a new mismatched dataset by training a GAN that transforms the noisy features. The baseline system can still be used on clean data like it normally would, but it can additionally perform well on data that would have resulted in very poor performance. The GAN is a modular component which allows it to be replaced by another GAN when a new dataset needs to be evaluated in the future.

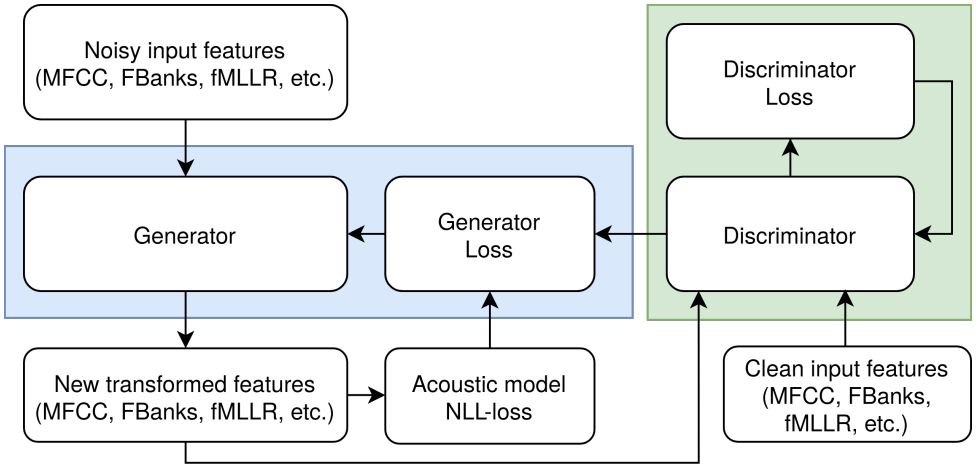

Figure 6.1: Diagram of the training process for a Guided-GAN that is used to transform noisy audio features to improve speech recognition accuracy.

6.2.1 GAN training

An MLP acoustic model is trained first as described in Section 3.4.1. This acoustic model is used to guide the GAN during training and to perform decoding afterwards. The GAN can use any type of feature or embedding as input, but we chose fMLLR features following the results in Section 4.3.3.

We use three different loss functions and select the one that is best suited for our approach (see Section 6.2.5 for details about the other two loss functions). The first is an adapted version of a spectral normalisation GAN (SN-GAN) [53], [55] in which we extend the standard generator loss function with an additional term. Specifically, we use the labels of the noisy sample to calculate the NLL loss of the cleaned sample. This value is added to the loss function to guide the generator to create features that are more likely to be classified correctly by the target acoustic model. The standard SN-GAN loss for the generator is given by:

$$\mathcal{L}_G = -\mathbb{E}_{\hat{\mathbf{x}}\sim p_g}[D(\hat{\mathbf{x}})], \tag{6.1}$$

where p_g is the model distribution implicitly defined by $\hat{\mathbf{x}} = G(\tilde{\mathbf{x}})$ with $\tilde{\mathbf{x}}$ the noisy input features, and $D(\hat{\mathbf{x}})$ is the output of the discriminator that can be a positive or negative value. We add the NLL loss to (6.1):

$$\mathcal{L}_G = -\mathbb{E}_{\hat{\mathbf{x}}\sim p_g}[D(\hat{\mathbf{x}})] - \lambda \cdot \mathbb{E}_{\hat{\mathbf{x}},\tilde{y}\sim p_{data}} \log p_{am}(\tilde{y}|\hat{\mathbf{x}}), \tag{6.2}$$

where λ is a hyperparameter, p_{am} is the probability defined by the trained acoustic model, and $\tilde{y}$ is the ground-truth senone class label of the noisy features. We use a sigmoid activation function to bound the output of the discriminator between zero and one. This is done to ensure the magnitude of the SN-GAN loss is comparable to the NLL loss term. The λ hyperparameter controls this ratio. The discriminator is trained with the standard SN-GAN loss:

$$\mathcal{L}_D = -\mathbb{E}_{\mathbf{x}\sim p_d}[D(\mathbf{x})] + \mathbb{E}_{\hat{\mathbf{x}}\sim p_g}[D(\hat{\mathbf{x}})] \tag{6.3}$$

where p_d is the real distribution defined by the clean samples $\mathbf{x}$. Spectral normalisation is used in the discriminator to ensure that we satisfy the Lipschitz constraint [53]. The generator and discriminator are trained jointly by updating the discriminator once for every generator update. The generator loss is calculated after the discriminator has been updated. The discriminator has then already been trained on the batch that the generator will be updated on. This allows for a stronger discriminator, which in turn improves the generated output. Each network has an independent learning rate, which allows better control over the balance between the networks during training.

Algorithm 1 describes the training process of a Guided-GAN that is applied to the input features of an ASR system using an SN-GAN loss function. A Guided-GAN can also be applied to many other tasks. Any classification task dealing with mismatched data can possibly be improved by setting up a Guided-GAN in a similar way as presented here.

6.2.2 GAN evaluation

The most accurate and objective metric to select the best hyperparameters for a GAN is the WER on a development set. Measuring the WER requires decoding, which takes significant time and resources. An approach that is computationally more cost-effective and correlates well with WER (see Figures 6.2 and 6.3) is to measure the SeER of the baseline MLP acoustic model when given the generated features as input. The hyperparameters of the GAN are optimised to find the lowest SeER measured by a trained acoustic model on a development set.

Algorithm 1 Guided-GAN training algorithm.

Require: α_d, discriminator learning rate. α_g, generator learning rate. m, batch size. N, number of training epochs. k, number of data chunks. λ, weight of acoustic model NLL loss term.

Require: w_0, initial discriminator parameters. θ_0, initial generator parameters. ϕ_t, trained acoustic model parameters.

for N epochs **do**
  for k chunks **do**
    for Number of batches in chunk **do**
      Sample $\{\mathbf{x}^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_{clean}$ a batch from the clean data.
      Sample $\{\tilde{\mathbf{x}}^{(i)}; \tilde{y}^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_{noisy}$ a batch and labels from the noisy data.
      $\hat{\mathbf{x}} \leftarrow G_\theta(\tilde{\mathbf{x}})$
      $\mathcal{L}_D \leftarrow \frac{1}{m}\sum_{i=1}^{m}[D_w(\hat{\mathbf{x}}^{(i)}) - D_w(\mathbf{x}^{(i)})]$
      $w \leftarrow \text{Adam}(\nabla_w \mathcal{L}_D, w, \alpha_d)$
      $\mathcal{L}_G \leftarrow -\frac{1}{m}\sum_{i=1}^{m}[D_w(\hat{\mathbf{x}}^{(i)})] - \lambda \cdot \mathbb{E}_{\hat{\mathbf{x}},\tilde{y}\sim p_{data}} \log p_{am}(\tilde{y}|\hat{\mathbf{x}})$,
      $\theta \leftarrow \text{Adam}(\nabla_\theta \mathcal{L}_G, \theta, \alpha_g)$
    end for
  end for
  for Number of batches in development set **do**
    Sample $\{\tilde{\mathbf{x}}^{(i)}; \tilde{y}^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_{dev}$ a batch and labels from the development set.
    $\hat{\mathbf{x}} \leftarrow G_\theta(\tilde{\mathbf{x}})$
    SeER $\leftarrow f_{\phi_t}(\hat{\mathbf{x}})/\tilde{y}$
  end for
  Validate GAN results using SeER on entire development set. Save θ for early stopping if SeER is lowest yet.
end for

To confirm the correlation between WER and SeER, we measure both metrics during GAN training on the LibriSpeech corpus. Ten points are selected where we measure the SeER and perform decoding using the MLP acoustic model. The data points were selected ranging from the early stages in training up to a much later point where the network has converged. We do not fine-tune the acoustic model in this experiment. Figure 6.2 shows the WER versus SeER during GAN training. A strong correlation exists between WER and SeER. We also evaluated 11 different types of GAN models after training. A graph of WER versus SeER for these models after training is shown in Figure 6.3. The models in this graph used different network architectures, hyperparameters and loss functions. The correlation also holds here, despite the setups being quite different from one another.

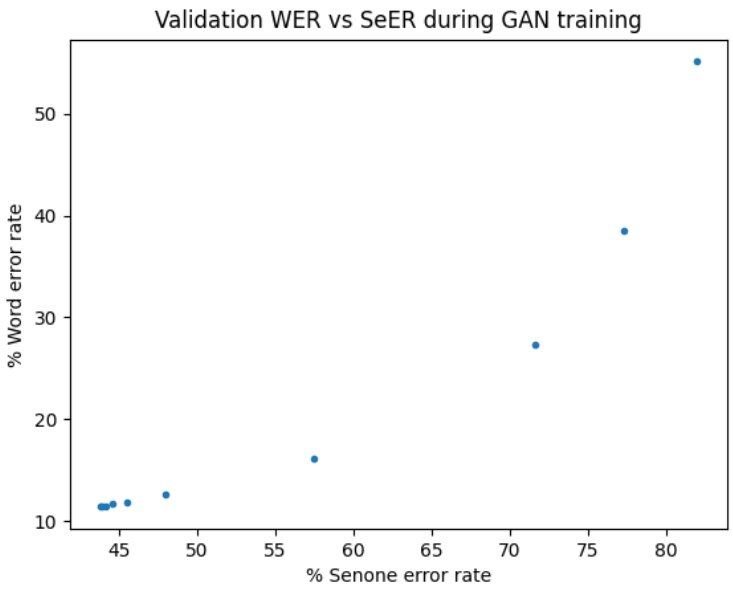

Figure 6.2: WER versus SeER on development set measured with trained acoustic model during GAN training.

6.2.3 GAN optimisation

We tune the hyperparameters of a GAN by measuring the SeER on a development set. In initial experiments, the batch size and learning rate had the biggest impact on the SeER. We find the best combination of batch size and learning rates in a grid search. After the search is complete, we then expand the search in that area to find better

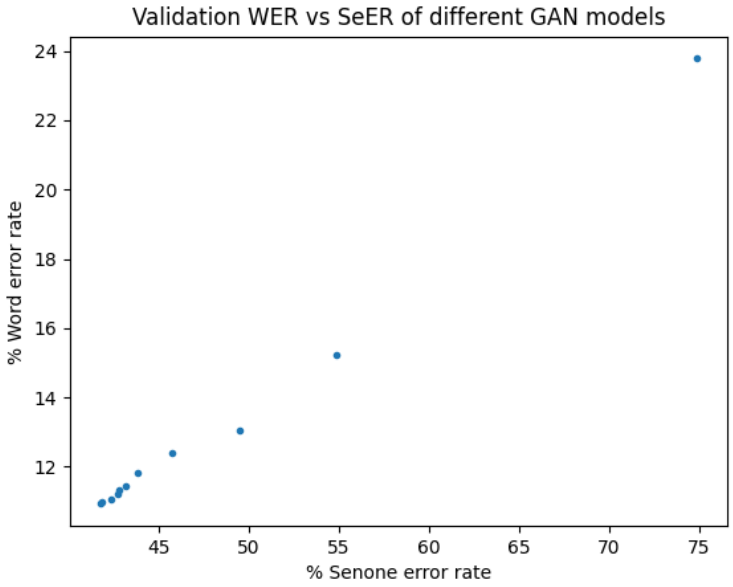

Figure 6.3: WER versus SeER on development set for 11 different GAN models after training.

results when varying the acoustic model λ. In initial experiments, we also varied the dropout probability in the discriminator, but later kept this constant. When changing these parameters, we also change the batch size and learning rates again.

Only the GAN with the lowest SeER is used to evaluate the acoustic model. We typically report on two versions of the acoustic model: one where the model is used directly to decode the test set, or a version that is fine-tuned. To fine-tune a model, we continue training the acoustic model from a previous point using the GAN on the features of the labelled training set. A new set of hyperparameters is used for fine-tuning since the model has already been trained until convergence occurred using the original settings. Fine-tuning can reduce the WER because the initial acoustic model was not trained on the new transformed features. The acoustic model is fine-tuned until the SeER converges. After fine-tuning, we decode the development set and select the best combination of language model weight and word insertion penalty using a grid search. Testing is done once, using the best settings found on the development set.

6.2.4 Network architectures

We use two different network architectures for both the generator and discriminator in a Guided-GAN. The first architecture is based on the encoder-decoder architecture from SEGAN [17]. The original network operated on raw audio and added random noise between the encoder and decoder networks. Our implementation uses fMLLR features as input and we do not add any noise. The second architecture we use improves the efficiency of the first by reducing the number of layers and removing the downsampling and upsampling processes. The parameters are fewer and allows for much faster training. This architecture is referred to as a fully-convolutional network because the output dimension will always equal the input dimension for any given input dimension. We will show later in Section 6.3.1, that the performance of the two networks is similar in terms of the final WER for a system.

Encoder-decoder architecture

A diagram of the generator and discriminator architectures for the SEGAN inspired networks are shown in Figures 6.4 and 6.5. We do not add any random noise or dropout to the generator during training or evaluation, which allows for a deterministic network. We found that the results were more consistent this way. The encoder network (shown in grey blocks) uses five convolutional layers with a stride of two for downsampling. The first two layers have a kernel size of seven, then two layers with a kernel size of five and the last layer has a kernel size of three. All layers in the generator, except for the output layer, use ReLU activation functions. The decoder network (shown in orange blocks) has five transposed convolutional layers with a stride of two for upsampling. The first and second layer uses a kernel size of four and five, respectively, followed by three layers with a kernel size of six. The choices for the kernel sizes were based on experimental results and architectural reasons (to upsample to the same dimension as the corresponding layer in the encoder). Skip connections are used to splice the feature maps of the encoder layers to the corresponding decoder layers. This is done because the encoder network extracts high-dimensional features and would struggle to produce an accurate output if it only

upsamples these high-dimensional features. By splicing the feature maps of the encoder network, the features before the encoder operated on them can assist the generator to produce a better output. The high-dimensional features then assist the layers in the decoder to improve the upsampling process by removing noise from the original features.

The discriminator network that works with the SEGAN-inspired generator has eight convolutional layers followed by a fully-connected output layer. The first layer uses a kernel size of 41, the rest of the layers use a kernel size of 13. The network downsamples the features with max-pooling layers. All convolutional layers use leaky ReLU activation functions with a negative slope of 0.2. Dropout is used only in the first two layers with a probability of 0.3. The fully-connected output layer uses spectral normalisation and has a sigmoid activation function.

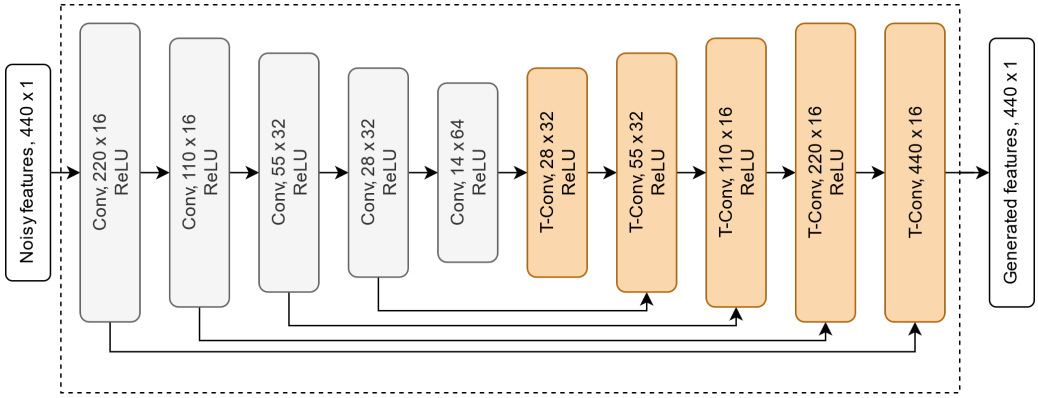

Figure 6.4: Diagram of encoder-decoder generator network architecture. ('Conv' is a convolutional layer with a stride of two. 'T-Conv' is a transposed convolutional layer.)

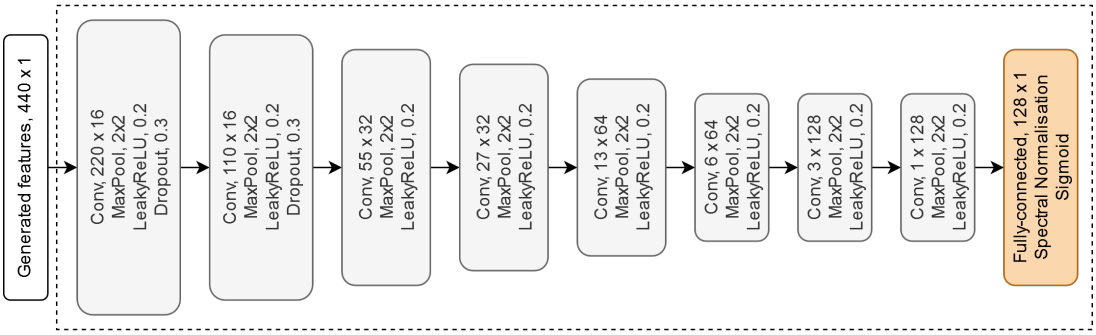

Figure 6.5: Diagram of discriminator network architecture for the encoder-decoder generator. ('Conv' is a convolutional layer.)

Fully-convolutional architecture

The fully-convolutional generator architecture is shown in Figure 6.6. We use a kernel size of five for all layers, with zero padding to ensure that the dimension stays the same. The first four layers use leaky ReLU activation functions with a negative slope of 0.2. The last layer creates the final output without any activation function. Similar to the encoder-decoder generator network, we also do not use dropout or add random noise at any stage.

The discriminator, shown in Figure 6.7, is a deep convolutional neural network with max-pooling layers and a fully-connected output layer. All layers, except the output, use leaky ReLU activation functions with a negative slope of 0.2. Dropout is used in the first three layers with a probability of 0.25. The output layer is a fully-connected layer with spectral normalisation and a sigmoid activation function.

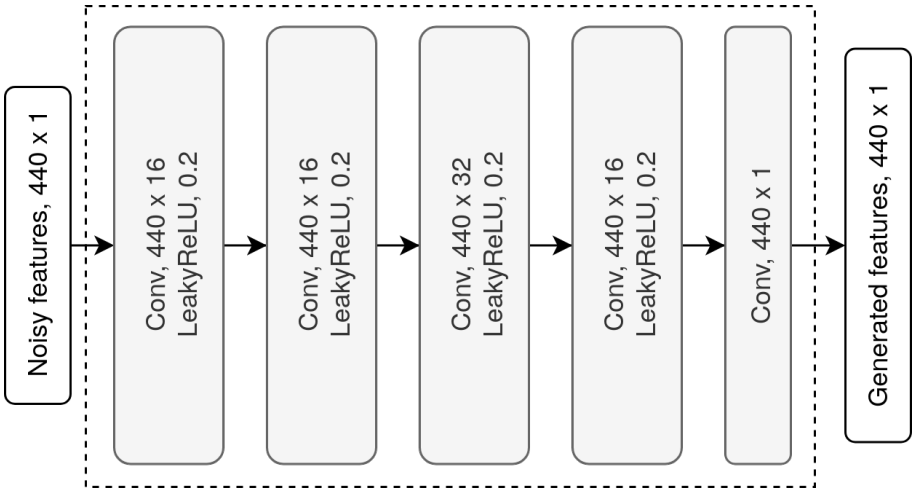

Figure 6.6: Diagram of fully-convolutional generator network architecture. ('Conv' is a convolutional layer.)

6.2.5 Loss functions

The adapted SN-GAN loss functions we use for the Guided-GAN was introduced in Eq. 6.2 and 6.3. We also compare these loss functions with the standard NS-GAN [16] and the WGAN loss function with gradient penalty (GP) [55]. These loss functions replace

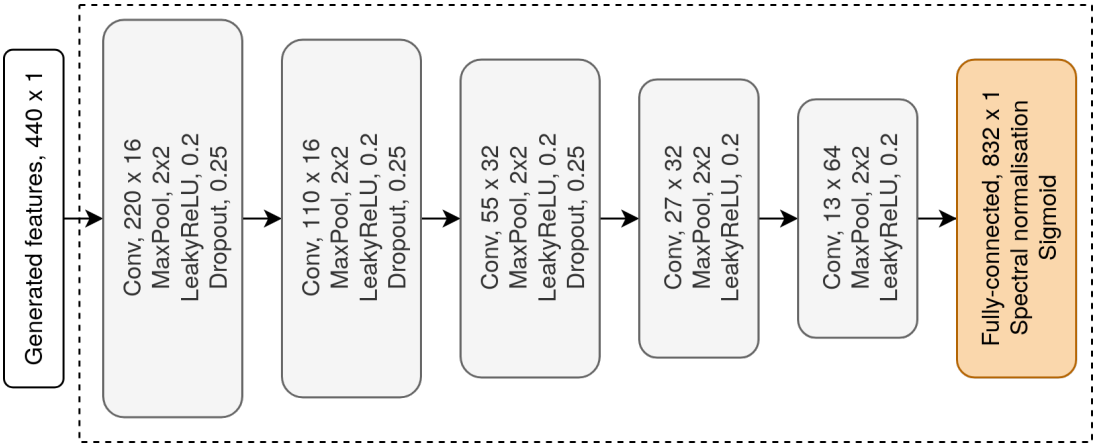

Figure 6.7: Diagram of discriminator network architecture for the fully-convolutional generator. ('Conv' is a convolutional layer.)

the SN-GAN loss functions in Algorithm 1 when training a Guided-GAN.

The NS-GAN loss function for the generator is:

$$\mathcal{L}_G^{NS-GAN} = -\mathbb{E}_{\hat{\mathbf{x}}\sim p_g}[\log D(\hat{\mathbf{x}})] \tag{6.4}$$

with a corresponding discriminator loss function:

$$\mathcal{L}_D^{NS-GAN} = -\mathbb{E}_{\mathbf{x}\sim p_d}[\log\left(D(\mathbf{x})\right)] - \mathbb{E}_{\hat{\mathbf{x}}\sim p_g}[\log\left(1 - D(\hat{\mathbf{x}})\right)]. \tag{6.5}$$

The WGAN-GP loss function for the generator is the same as for the SN-GAN in Eq. 6.1. The Lipschitz constraint is satisfied in the discriminator by adding a gradient penalty term to Eq. 6.3:

$$\begin{aligned}\mathcal{L}_D^{WGAN-GP} = -\mathbb{E}_{\mathbf{x}\sim p_d}[D(\mathbf{x})] + \mathbb{E}_{\hat{\mathbf{x}}\sim p_g}[D(\hat{\mathbf{x}})] + \\ \lambda_{gp}\mathbb{E}_{\hat{\mathbf{x}}\sim p_g}[(||\nabla D(\alpha\mathbf{x} + (1-\alpha)\hat{\mathbf{x}})||_2 - 1)^2],\end{aligned} \tag{6.6}$$

where λ_{gp} is a hyperparameter controlling the gradient penalty, and α is a uniform random number between zero and one.

6.3 Guided-GANs in a controlled environment

In this section, we evaluate Guided-GANs on three datasets in a controlled environment. We first compare different loss functions and architectures on the clean WAV49-encoded

LibriSpeech corpus to select the best setup and evaluate the GAN against a strong baseline (Section 6.3.1). Next, we compare the Guided-GAN using the fully-convolutional architecture to the best MTR systems developed in Chapter 5 on the noisy WAV49-encoded LibriSpeech corpus with mismatched noise conditions (Section 6.3.2). Finally, we investigate the use of a Guided-GAN in a resource-scarce environment (Section 6.3.3) and compare the training time of a Guided-GAN to that of an MTR system (Section 6.3.4).

6.3.1 Clean WAV49-encoded LibriSpeech corpus

We first train a Guided-GAN on the clean WAV49-encoded LibriSpeech corpus to demonstrate the ability of the GAN to correct sampling rate differences and reduce the effect of encoding. The GAN is compared to the clean and encoded baselines that were developed in Section 4.3.

We use the *train-clean-100* subset of the LibriSpeech corpus as the 'clean' set and the *train-clean-e* set as the 'noisy' set for GAN training. The datasets we use are described in Section 4.3.1. A Guided-GAN is trained and optimised using the protocol described in Sections 6.2.1 and 6.2.3. A summary of the hyperparameters is given in Appendix A.1.1. The first GAN we train uses the baseline *train-clean-100* acoustic model for guidance. This model was trained using 16kHz unencoded audio which requires the GAN to compensate for sampling rate differences and encoding. Another GAN is trained using the *train-clean-e* acoustic model, which already used 8kHz encoded audio for training. In this case, the GAN does not have to compensate for sampling rate differences or encoding because the acoustic model was already trained on the same type of audio. However, the GAN may be able to improve the performance by reducing the effect of encoding artefacts because we still use unencoded clean data to train the GAN.

Table 6.1 shows the results on the *dev-clean-e* subset of the LibriSpeech corpus. The top section shows the WERs for the two baseline models and an MTR system that used speed perturbation (*train-clean-e + s*). The GAN is evaluated before and after fine-tuning.

The GANs using the encoder-decoder network architecture are labeled with 'E-D' and the networks using the fully-convolutional architecture with 'F-C'.

The GANs that were trained using the *train-clean-100* acoustic model reduced the WER of the baseline significantly. Fine-tuning these models further reduced the WER. The two SN-GAN networks outperformed the NS-GAN and WGAN-GP networks. The network using the fully-convolutional architecture matched the performance of the MTR model.

When using the *train-clean-e* model to train the GAN and fine-tuning, the performance improves further. Both network architectures performed slightly better than the MTR model. An interesting result is that without fine-tuning, both these models have a higher WER than the baseline model used to train them.

The two GAN models using the fully-convolutional architecture are also evaluated on the *test-clean-e* set to confirm the results. Table 6.2 shows the WER results on the test set. The results are similar to those on the development set, although the average WER of the

Table 6.1: WER results of Guided-GANs compared to baseline, encoded and MTR models on *dev-clean-e*. Average WER and standard error is shown over three seeds.

Model	**WER**	**WER (fine-tuning)**
train-clean-100	19.32 ± 0.11	-
train-clean-e	11.02 ± 0.03	-
train-clean-e + s	10.93 ± 0.08	-
train-clean-100		
NS-GAN (E-D)	13.45 ± 0.12	11.14 ± 0.05
WGAN-GP (E-D)	13.52 ± 0.04	11.14 ± 0.02
SN-GAN (E-D)	12.80 ± 0.01	11.04 ± 0.03
SN-GAN (F-C)	13.41 ± 0.04	10.93 ± 0.03
train-clean-e		
SN-GAN (E-D)	11.29 ± 0.04	10.81 ± 0.04
SN-GAN (F-C)	11.36 ± 0.02	**10.79 ± 0.03**

MTR model was slightly lower than the *train-clean-100* GAN. These results confirm that the Guided-GAN is effective to recover most of the lost performance due to the reduced sampling rate and WAV49 encoding. The results of the GAN is comparable to MTR.

Table 6.2: WER results of Guided-GAN compared to baseline, encoded and MTR models on *test-clean-e*. Average WER and standard error is shown over three seeds.

Model	**WER**	**WER (fine-tuning)**
train-clean-100	19.15 ± 0.11	-
train-clean-e	11.10 ± 0.02	-
train-clean-e + s	11.00 ± 0.07	-
train-clean-100		
SN-GAN (F-C)	13.59 ± 0.03	11.04 ± 0.02
train-clean-e		
SN-GAN (F-C)	11.58 ± 0.11	**10.94 ± 0.02**

6.3.2 Noisy WAV49-encoded LibriSpeech corpus

We also apply the Guided-GAN to the noisy WAV49-encoded LibriSpeech corpus created in Section 5.3.1 to determine if the Guided-GAN is effective when using noisy data. The GAN is compared to the best MTR models that were developed in Chapter 5.

The GAN is trained the same way as in Section 6.3.1, except that we use the *train-musan-e-15* set for our 'noisy' set and the *train-clean-e* set as our 'clean' set for the GAN using the *train-clean-e* acoustic model. The GAN using the *train-clean-100* set still uses the *train-clean-100* set as the 'clean' set for GAN training. The systems are evaluated on the *dev-noisy-e-5* and *test-noisy-e-5* sets. Similar to before, two GANs are developed using different acoustic models for guidance, the *train-clean-100* and *train-clean-e* models. The first GAN has to compensate for sampling rate difference, encoding and additive noise, while the second GAN only has to remove noise. Both GANs use the fully-convolutional network architecture and are evaluated with and without fine-tuning.

Table 6.3 shows the results on the *dev-noisy-e-5* set. The top section shows the WERs for the baseline and MTR models. The two MTR models, *train-clean-e + s* and *train-musan-e-15*, achieved the best results in Chapter 5, which is why we compare the GAN to them here. The GAN using the *train-clean-100* acoustic model improved the baseline result by 31.1% WER (relative improvement). Using the *train-clean-e* acoustic model further improved the WER of the GAN. Despite this improvement, the *train-musan-e-15* model still outperformed both GANs.

We also evaluate the models using the *test-noisy-e-5* set. Similar to the results on the development set, the *train-musan-e-15* model outperformed both GANs. Regardless, the two GAN models improved the WER of the baseline acoustic models by 27.2% and 9.0%, respectively (relative improvements). This shows that the Guided-GAN is effective to remove a large mismatch created by differences in sampling rate, WAV49 encoding and additive noise, but may still be outperformed by MTR.

Table 6.3: WER results of Guided-GAN compared to baseline, encoded and MTR models on *dev-noisy-e-5*. Average WER and standard error is shown over three seeds.

Model	**WER**	**WER (fine-tuning)**
train-clean-100	36.46 ± 0.14	-
train-clean-e	28.06 ± 0.03	-
train-clean-e + s	27.83 ± 0.02	-
train-musan-e-15	**23.30 ± 0.06**	-
train-clean-100		
Guided-GAN	31.84 ± 0.06	25.76 ± 0.01
train-clean-e		
Guided-GAN	29.21 ± 0.02	25.30 ± 0.03

Table 6.4: WER results of Guided-GAN compared to baseline, encoded and MTR models on *test-noisy-e-5*. Average WER and standard error is shown over three seeds.

Model	**WER**	**WER (fine-tuning)**
train-clean-100	36.92 ± 0.18	-
train-clean-e	29.16 ± 0.12	-
train-clean-e + s	28.61 ± 0.11	-
train-musan-e-15	**24.54 ± 0.22**	-
train-clean-100		
Guided-GAN	32.74 ± 0.09	26.88 ± 0.01
train-clean-e		
Guided-GAN	30.15 ± 0.06	26.54 ± 0.05

6.3.3 Resource-scarce environment using LibriSpeech corpus

ASR systems generally struggle to perform well when matched training and evaluation data is limited. This is especially true in under-resourced environments where it is often prohibitively expensive to collect enough transcribed training data to cover even the most common application environments.

To evaluate the Guided-GAN in a simulated resource-scarce environment, we create subsets ranging from 1 hour to 100 hours using the *train-other-500* subset of the LibriSpeech corpus. We use this set because it contains true noisy data that was not artificially created. Random speakers are selected from the *train-other-500* set to get a diverse training set. The datasets we use are shown in Table 6.5. We later refer to these sets using the abbreviation in the *Ref* column. The 100-hour set, containing 250 speakers, is used to create speed- and volume-perturbed sets for MTR. Testing is done using the *dev-other-e* and *test-other-e* sets. We only use the small language model (tg-small) for decoding.

We use the baseline model that was trained on the *train-clean-100* set in Section 4.3 to decode the development and test sets. The WER of this model on the *test-other-e* set is 42.52%. Noise and encoding is highly detrimental to the performance of this model.

Table 6.5: LibriSpeech dataset partitions used to simulate a resource-scarce environment.

Dataset	**Ref**	**Encoding**	**Hours**
train-clean-100	TC	-	100.6
train-clean-e	TCE	WAV49	100.6
train-other-e-1	TOE1	WAV49	1.0
train-other-e-5	TOE5	WAV49	5.0
train-other-e-10	TOE10	WAV49	10.0
train-other-e-20	TOE20	WAV49	20.3
train-other-e-50	TOE50	WAV49	50.0
train-other-e-100	TOE100	WAV49	106.2
Multi-style training sets			
train-clean-e-s	-	WAV49	100.6
train-other-e-100-s	-	WAV49	106.2
train-other-e-100-v	-	WAV49	106.2
train-other-e-100-sv	-	WAV49	106.2
dev-other-e	-	WAV49	5.3
test-other-e	-	WAV49	5.1

We improve this model by adding a Guided-GAN using different amounts of training data. The GANs are trained and optimised using the protocol described in Sections 6.2.1 and 6.2.3. We use the fully-convolutional network architecture and fine-tune the acoustic model each time.

Table 6.6 shows the WER on the *dev-other-e* and *test-other-e* sets. Using only one hour of noisy data for GAN training improved the baseline by 20.2% WER on the test set (relative improvement). Adding additional noisy data reduced the WER further, however, the returns were diminishing.

The Guided-GAN can significantly improve a clean baseline acoustic model using very little matched training data. In resource-scarce environments, where it is sometimes

Table 6.6: WER results on *dev/test-other-e* (noisy, encoded data) using different amounts of training data. Average WER is shown over three seeds.

Model	**dev**	**test**
train-clean-100	39.42	42.52
Guided-GAN with TOE1	31.79	33.92
Guided-GAN with TOE5	29.84	31.26
Guided-GAN with TOE10	28.90	30.67
Guided-GAN with TOE20	27.53	29.47
Guided-GAN with TOE50	26.61	28.29
Guided-GAN with TOE100	**25.79**	**27.57**

difficult to collect many hours of transcribed data, a Guided-GAN can be a useful tool to adapt an existing system. An additional advantage of the architecture is that the GAN is quite fast to train, especially on small datasets.

To improve the baseline results, we train MTR models with the same combinations of datasets used in Section 5.3.2. Speed perturbation is used for the *train-clean-e* model. For the MTR models using the *train-other-e-100* set, we include results of the network using speed and volume perturbation in a single set as well as two sets separately. We also train a Guided-GAN using the *train-clean-e* acoustic model. We fine-tuned the GAN using the *train-other-e-100* set. Additionally, we fine-tuned the GAN using the MTR datasets to improve the performance even more.

Table 6.7 shows the WER results of the baselines, MTR systems and Guided-GANs on the *dev-other-e* and *test-other-e* sets. The top section shows the baseline acoustic models that are used by the GAN for guidance. The second section shows models that improve the baseline results using MTR and/or matched training data. The third section shows the results of the GANs using the *train-clean-100* acoustic model for guidance. The last section shows the results for the GANs that use the *train-clean-e* acoustic model. The best results (in bold) are comparable. Fine-tuning is indicated with an 'FT' label.

Applying the GANs to the *train-clean-100* and *train-clean-e* models improved the WER of the baseline acoustic models by 20.7% and 6.0%, respectively (relative improvements). Fine-tuning the acoustic models using the *train-other-e-100* set further reduced the WER by 18.2% and 7.7%, respectively. Fine-tuning the *train-clean-e* acoustic model using the speed- and volume-perturbed datasets yielded the best performance of all GANs.

The results of the Guided-GAN are very comparable to MTR in terms of WER. In Section 6.3.4, we compare the training time of the models to see the true benefit of the GAN.

Table 6.7: WER results on *dev/test-other-e* of baseline, MTR and GAN models. Average WER and standard error is shown over three seeds.

Model	**dev**	**test**
Baseline models		
train-clean-100	39.42 ± 0.57	42.52 ± 0.57
train-clean-e	29.01 ± 0.10	31.16 ± 0.16
Improved models		
train-clean-e + s	28.57 ± 0.04	30.67 ± 0.09
train-other-e-100	**25.32 ± 0.08**	**26.87 ± 0.11**
train-other-e-100 + sv	**25.19 ± 0.08**	**26.61 ± 0.10**
train-other-e-100 + s + v	**24.99 ± 0.01**	**26.49 ± 0.12**
train-clean-100		
Guided-GAN	31.25 ± 0.07	33.71 ± 0.03
Guided-GAN + FT	25.79 ± 0.10	27.57 ± 0.07
train-clean-e		
Guided-GAN	27.82 ± 0.20	29.29 ± 0.02
Guided-GAN + FT	**25.27 ± 0.01**	**27.02 ± 0.02**
Guided-GAN + MTR FT (s + v)	**24.98 ± 0.05**	**26.70 ± 0.07**

6.3.4 Training time of GAN vs MTR

DNNs are famous for being very expensive to train computationally. This is even more apparent in MTR systems because the training data is multiplied by the number of styles that are added. In Section 6.3.3, we showed that a Guided-GAN is comparable to an MTR system in terms of WER. In this section, we compare the training time of a Guided-GAN with that of MTR systems.

Training a GAN requires more memory than training a standard MLP acoustic model because the GAN uses three networks during training (generator, discriminator and acoustic model). Although it uses more memory, the training time is less than training an MTR model because the GAN converges much faster than an acoustic model. The GAN also uses much larger batch sizes during training, which also speeds it up significantly. Table 6.8 shows the training time for the Guided-GAN and MTR models used in Section 6.3.3. The WER is reported on the *test-noisy-e-5* set and the training time is shown in minutes. Fine-tuning is indicated with an 'FT' label. The training time is measured using a desktop computer running an Intel i7 8700k CPU with an Asus RTX 3080 GPU. The system has 32 GB RAM running at 3 200 MHz. The baseline and MTR models were trained for 24 epochs. The GANs using the 100-hour training set were trained for 20 epochs. The GANs using less data were trained for more epochs. All networks converged before the end of training was reached. We always use early stopping on the validation SeER to select the best GAN network, but we still compare the training time until the end of training.

Training a Guided-GAN using the fully-convolutional architecture and fine-tuning the acoustic model is about three times faster than training an MTR model using two datasets and four and a half times faster than one using three sets. The MTR systems perform very similarly to the GAN in terms of WER. Fine-tuning the GAN with the MTR datasets is still two times faster than the MTR model using the same sets. The GAN is also extremely efficient with limited training data. A relative WER improvement of 20.2% on the *train-clean-100* model is achieved with only 10 minutes of GAN training, including fine-tuning time. With 10 hours of training data, the relative WER improvement increases

to 27.9% with a GAN trained in 52 minutes.

This shows that the Guided-GAN is not only a viable replacement for MTR in resource-scarce environments, it is also an efficient technique to improve a good baseline system on mismatched data. The only requirement is that a small, transcribed dataset with conditions similar to the test set is available. Even when an exact match is not available, data augmentation can be used to create a noisy set that is better matched.

Table 6.8: Training time (minutes) comparison of Guided-GANs, baselines and MTR systems on a 100-hour training set (*train-other-e-100*).

Model	**Test WER**	**Training Time**
train-clean-100	42.52	267
train-clean-e	31.16	271
train-other-e	26.87	288
train-other-e + sv	26.61	583
train-other-e + s + v	26.49	875
train-clean-100		
Guided-GAN with TOE1 + FT	33.92	10 (4 + 6)
Guided-GAN with TOE5 + FT	31.26	35 (16 + 19)
Guided-GAN with TOE10 + FT	30.67	52 (30 + 22)
Guided-GAN with TOE20 + FT	29.47	96 (46 + 50)
Guided-GAN with TOE50 + FT	28.29	140 (60 + 80)
Guided-GAN with TOE100 + FT	27.57	195 (80 + 115)
train-clean-e		
Guided-GAN	29.29	80
Guided-GAN + FT	27.02	195 (80 + 115)
Guided-GAN + MTR FT (s + v)	26.70	425 (80 + 345)

6.4 Analysis

In this section, we investigate the improvements made by a Guided-GAN. We compare the errors made between the baseline and the GAN (Section 6.4.1) and visualise the improvements on a frame level (Section 6.4.2).

6.4.1 Error comparison

We compare the errors made by the *train-clean-100* baseline and Guided-GAN by using the *train-clean-100* acoustic model. Table 6.9 shows a summary of the errors made by the two models. The GAN reduced the number of insertions, deletions and substitutions, which resulted in a lower WER. On the frame level, the GAN improved the SeER by 14.9% relatively. This shows that a Guided-GAN improves the acoustic model on all types of errors and not just a specific type.

Table 6.9: Comparison of errors made by the *train-clean-100* baseline and Guided-GAN trained by using the *train-clean-100* acoustic model on the LibriSpeech *dev-clean-e* set.

Metric	**Baseline**	**Guided-GAN**
Insertions	984	746
Deletions	1 317	616
Substitutions	8 228	4 569
Total errors	10 529	5 931
Total words	54 402	54 402
WER	19.35%	10.90%
SeER	48.90%	41.60%

6.4.2 Feature transformation

To visualise the transformation that a Guided-GAN makes on a frame level, we plot the fMLLR features of two clean, encoded and generated samples. We use the features of a

clean sample and the WAV49-encoded version of the same sample. We show two examples in Figure 6.8. The encoded and generated features on the left were classified incorrectly by the acoustic model. In the figure on the right, the encoded features were classified incorrectly, but the generated features were classified correctly.

On average, the encoded features correlate slightly better with the clean features than the generated features with the clean features. This indicates that the generator does not aim to recover the original features, but instead changes the features in some way so that the acoustic model better classifies them. This is further confirmed by using a trained generator on another acoustic model that was not used for guidance (different initialisation seed used for the acoustic model). If the features were closer to the original features, we would expect that the new acoustic model will also be improved, but this is not the case. Instead, the performance is worse than when using the model without a GAN. The Guided-GAN can only be used to improve the model that was used for guidance during training because the generator only learns how to improve the classification accuracy of that specific model.

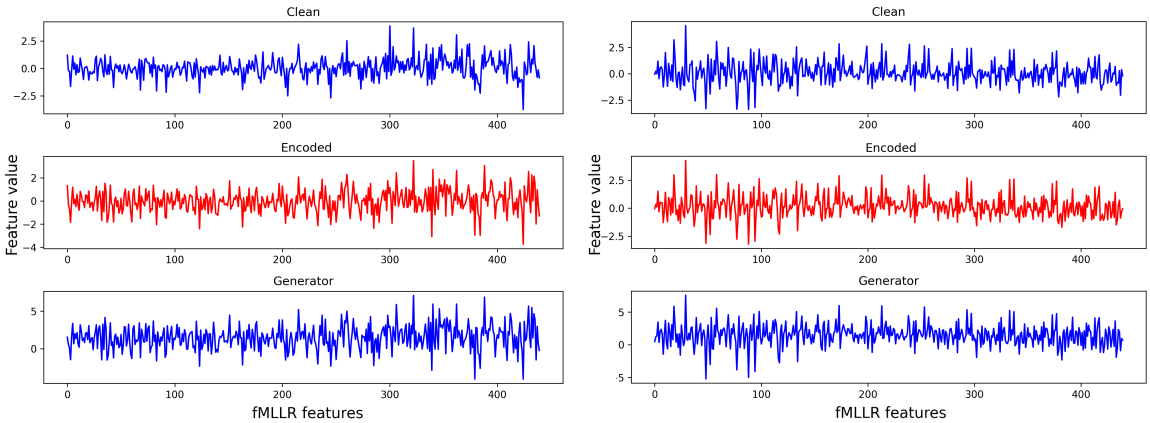

Figure 6.8: Features of clean, encoded and generated samples of two different utterances.

6.5 Discussion

In this chapter, we developed a Guided-GAN that transforms noisy audio features to be better classified by an acoustic model. The GAN uses an SN-GAN loss function with the

NLL loss of a trained acoustic model appended to guide the generator during training. This eliminates the requirement to artificially create a parallel dataset that does not truly represent the target conditions. The Guided-GAN is much faster to train than an MTR system with similar performance. We demonstrated that the Guided-GAN can reduce the effect of sampling rate mismatch, encoding artefacts and background noise. We also showed that the GAN is effective in under-resourced environments where computational resources and training data are limited.

Although we applied the Guided-GAN to a DNN-HMM ASR system, we believe its applications can be extended to many other domains. Any classification task that is evaluated on mismatched data should be able to benefit from using a Guided-GAN.

Chapter 7

Real-world experiment

A Guided-GAN is applied to proprietary South African call centre datasets to demonstrate its ability to work in the real world.

7.1 Introduction

In this chapter, we use the Guided-GAN on the SACC and MSCC corpora described in Sections 3.3.4 and 3.3.5. This is done to demonstrate the ability of the GAN to be used in a real-world scenario. In Section 7.2, we describe the development of baseline, MTR and Guided-GAN systems using the SACC and MSCC corpora. We present the results of these systems in Section 7.3.

7.2 System overview

In this section, we first provide an overview of the baseline system that is trained on the SACC corpus (Section 7.2.1). We then describe an MTR system that improves the

baseline results by using three training datasets (Section 7.2.2). Finally, we describe the Guided-GAN that is developed to improve the baseline results on the MSCC corpus (Section 7.2.3).

7.2.1 Baseline ASR

A baseline ASR system is trained on the SACC corpus using the unencoded training set (*train*). Table 3.2 provides a summary of the datasets in the SACC corpus. We use the MLP network architecture described in Section 4.2.1. The network is trained and optimised using the protocol described in Sections 3.4.1 and 3.4.2, except that we train the network for 30 epochs because the training data consists of only 48.8 hours. At this point, the network has converged. All data in the SACC corpus are single channel recordings sampled at 8kHz. The ASR systems developed for this corpus use 8kHz audio without upsampling. We also train a model with WAV49-encoded training data (*train-e*). This is done to improve the unencoded baseline on encoded audio. The lexicon and language model are specific to the SACC corpus. We use the same lexicon and language model on the MSCC corpus, despite it only including vocabulary from the SACC corpus.

7.2.2 MTR system

To improve the baseline results even more, we create an MTR system that uses three datasets for training: unencoded (*train*), WAV49-encoded (*train-e*) and speed-perturbed WAV49-encoded audio. We include the unencoded training set because we want to jointly perform well on both encoded and unencoded test sets. The network architecture, training and optimisation protocol are the same as for the baseline models.

7.2.3 Guided-GAN

A Guided-GAN is trained on the SACC and MSCC corpora using the fully-convolutional architecture described in Section 6.2.4. The GAN is trained and optimised using the protocol described in Sections 6.2.1 and 6.2.3, except that the GAN is trained for more epochs depending on the amount of data in the subset.

We use the unencoded SACC *train* set as the 'clean' set for the GAN. Four different GANs are trained using subsets of the MSCC corpus as the 'noisy' sets. The GAN is first trained using only the training sets of each call centre (see Table 3.3). All results are presented after fine-tuning the acoustic model. The development sets are used to validate the SeER of the acoustic model using the GAN. The best GAN is used to decode the development set. A new GAN is trained on the training and development sets using the hyperparameters that was selected on the development set. This is done to expand the GAN's training data because the subsets contain very little data. The training data for individual call centres range from 6.6 minutes to 43.2 minutes, and only 351.2 minutes for the entire corpus. The test set is decoded using the final system without optimising the hyperparameters again.

7.3 Results

In this section, we present the results of the baseline, MTR and Guided-GAN systems on a real-world dataset. First, the results are given for the baseline and MTR systems on the SACC corpus (Section 7.3.1). We then use a Guided-GAN to improve the baseline model on the MSCC corpus (Section 7.3.2).

7.3.1 Results on the SACC corpus

We evaluate the baseline and MTR models on the *dev*, *test* and *held-out test* sets of the SACC corpus. The average WERs over three seeds are shown in Table 7.1.

The model that used only the encoded training set performed better on encoded test sets, but similar to the baseline model on the *dev* set and worse on the *test* set. The MTR model performed the best across all five datasets with relative WER improvements of between 1.1% and 2.9%.

Since the training data is quite well matched with the testing data, MTR does not provide large improvements. It does not hurt performance, and small consistent improvements are possible, but the real advantage of MTR is only observed if there is a significant mismatch. The largest mismatch on the LibriSpeech corpus was the sampling rate difference and loud background noise. On the SACC corpus, the unencoded training data is already sampled at 8kHz and contains a lot of noise. The only mismatch here is WAV49-encoding, which does not have a very large effect on its own, as shown in Section 4.3.2. Because the mismatch is not very large, we do not train a Guided-GAN to improve the results on the SACC corpus.

Table 7.1: WER results on *dev/test* sets for the SACC corpus. Average WER is shown over three seeds.

Model	**dev**	**dev-e**	**test**	**test-e**	**held-out test**
train	28.41	28.91	33.14	33.43	41.90
train-e	28.40	28.63	33.36	33.04	41.80
MTR	**27.98**	**28.19**	**32.77**	**32.46**	**41.42**

7.3.2 Results on the MSCC corpus

We evaluate the baseline model trained on the SACC corpus using the development and test sets of the MSCC corpus. The baseline results are presented using the exact same system that was trained in Section 7.3.1. Three Guided-GANs are trained for individual call centres and a final one for the entire corpus. Table 7.2 shows the WERs on the development sets. The relative improvement is shown in the right column. The GAN consistently outperformed the baseline model with relative WER improvements of between 5.45% and 24.74%.

Table 7.3 shows the WERs on the test sets after the development set has been folded back. The relative improvements on the test set range from 9.01% to 19.70%. The Guided-GANs consistently improve the baseline MLP acoustic model on the development and test sets. With very little data (from 9.5 minutes), the GAN is able to reduce the WER on all test sets. All three GANs used on individual call centre data were trained in less than 10 minutes on an NVidia RTX 2080 ti GPU. The fine-tuning also took less than 15 minutes for the largest set (Call centre 1 *train* + *dev*). This confirms that the Guided-GAN is an effective tool when an existing, strong baseline model needs to be used to decode mismatched data.

Table 7.2: WER results on the MSCC development sets.

Dataset	**Baseline**	**GAN**	**Improvement**
Call centre 1	56.58	45.67	19.28%
Call centre 2	48.67	36.63	24.74%
Call centre 3	70.66	66.81	5.45%
All centres	61.68	56.11	9.03%

Table 7.3: WER results on the MSCC test sets.

Dataset	**Baseline**	**GAN**	**Improvement**
Call centre 1	55.11	48.77	11.50%
Call centre 2	59.05	51.40	12.96%
Call centre 3	52.38	42.06	19.70%
All centres	60.63	55.17	9.01%

7.4 Discussion

In this chapter, we applied the Guided-GAN to a real-world problem. First, a good baseline ASR system was trained on unencoded call centre audio. MTR was used to

improve the baseline on unencoded and WAV49-encoded test sets. The improvements on these sets, although consistent, were relatively small. We attribute this to the sets being well matched, with the only difference between them being WAV49 encoding.

We then used a Guided-GAN to improve the baseline model on four mismatched datasets that came from different call centres. Although the training data was extremely limited, the Guided-GAN consistently improved all results. This demonstrated the ability of the Guided-GAN to work on mismatched poor quality audio obtained from a real-world source.

Chapter 8

Conclusion

The key findings of this study are summarised, followed by the contributions made. We provide avenues for possible future work utilising a Guided-GAN architecture.

8.1 Overview

The goal of this study was to investigate the use of GANs to improve speech recognition performance of poor quality audio obtained from a real-world source. In this chapter, we review how the initial objectives of the study were met. We discuss the key findings of this study, followed by the contributions made. We propose possible directions for future work that can utilise a Guided-GAN architecture.

8.2 Key findings

We compared many different ASR systems on several datasets in this study. We investigated how GANs can be used to improve ASR systems on poor quality audio. The key

findings of this study include:

- A Guided-GAN used as a front-end to an existing baseline ASR system is an effective tool to recover lost performance due to sampling rate mismatches, background noise and encoding. We showed this on three datasets in a controlled environment and confirmed it again on a real-world dataset. The GAN-based feature transformation technique is comparable to MTR in terms of performance, but more cost-effective computationally.

- Fine-tuning is a useful technique to improve the performance of a Guided-GAN. Noisy features are changed by the generator, which means the acoustic model was never trained directly using the transformed features. By fine-tuning the acoustic model, the DNN can learn to use the transformed features correctly, which then improves the WER. Fine-tuning with MTR datasets can improve the performance even more.

- A Guided-GAN benefits from having a better acoustic model for guidance. On all three experiments in the controlled environment, the GAN using the acoustic model trained with encoded data outperformed the one that used unencoded data. The acoustic model that was trained with encoded data can already compensate for some mismatch (WAV49 encoding in this case), while the GAN using the acoustic model trained with unencoded data also needs to learn how to handle encoding.

- A Guided-GAN trained with narrow-band audio can improve an ASR system trained only with wide-band audio.

- The SN-GAN loss function performed better than both the NS-GAN and WGAN-GP functions for a Guided-GAN. Gradient penalty is computationally much more expensive than spectral normalisation, which makes the SN-GAN loss even more attractive.

- The fully-convolutional network architecture is able to achieve comparable results to the encoder-decoder networks, but has fewer parameters and is computationally more cost-effective to train.

- SeER is a good metric to predict the WER of a model. We confirmed that these two metrics are well correlated during the training process of a single network, but also after training of a number of networks using different architectures, hyperparameters and loss functions. SeER is, therefore, a good metric to select hyperparameters of a Guided-GAN applied to an ASR task.

While not the primary focus of this study, we also investigated the effect of different conditions that contribute to decreased ASR performance. Additional findings related to MTR include:

- MTR can significantly benefit ASR systems when there is a large mismatch between training and evaluation data. We showed that a baseline system can be improved by up to 33.5% relative WER by selecting appropriate MTR styles.
- Noise perturbation is a very effective method for data augmentation when only clean training data is available, and a noisy test set needs to be evaluated. The effectiveness of the perturbation relies on selecting the correct SNR value to match the data. If a significantly different SNR is used, the performance may be worse than that of the baseline.
- Speed and volume perturbation can improve an ASR system slightly, given that the network has enough capacity to benefit from the increased training dataset size. When an MLP acoustic model does not have enough capacity, the performance can get worse when MTR is used. We showed this when using clean and noisy datasets for perturbation.
- MTR still has shortcomings compared to using a matched training set. The best MTR system on the noisy WAV49-encoded LibriSpeech corpus still performed 16.5% (relative WER) worse than the model using matched training data. This confirms that it is very difficult to artificially create a training dataset that is perfectly matched to a test environment, which generally has unknown conditions.
- WAV49 encoding does not contribute to a large mismatch when clean speech data is encoded. However, the mismatch is much more significant when encoded noisy audio

is evaluated. The relative difference in WER on the LibriSpeech corpus between a model using unencoded, narrow-band audio and one using WAV49-encoded audio on clean speech data is 2.0%, and 15.6% on noisy speech data.

- In the setups tested, fMLLR input features were found to generalise better to audio with different sampling rates and encoding than MFCCs with delta and delta-delta coefficients and MFCCs with an i-vector appended. This was shown by evaluating networks using these features on sets with the same and different conditions (one set contained clean unencoded data and the other set clean WAV49-encoded data). The two networks using fMLLR features not only generalised better to the mismatched datasets, but they also achieved the best results on the matched datasets.

8.3 Contributions

During this research, a number of contributions were made that can be applied to ASR systems. The contributions include:

- A new GAN-based method was developed to improve ASR systems on mismatched and/or poor quality audio. The technique is very efficient and provides gains similar to MTR but at a reduced computational cost.
- The GAN training toolkit has been made available for future researchers to use. It has been integrated into the in-house codebase of the MUST Deep Learning research group.
- Different effects that contribute to mismatched data in ASR systems were isolated which allowed us to measure the performance impact of each effect. The effects that were studied include sampling rate differences, encoding, additive noise, speed and volume perturbation. The outcome of these experiments can be used to select MTR styles without having to experiment with all possible combinations in the future.
- A new state-of-the-art result was achieved on the 100-hour LibriSpeech corpus when

using an MLP acoustic model. We suspect this was possible because the hyperparameters of published results were not optimised enough. This sets a new benchmark for ASR systems using a similar setup.

8.4 Future work

Further development of the Guided-GAN will focus on improving the network architectures and extending the applications to other domains. The work will aim to address the following:

- Investigate alternative network architectures, including an RNN based on long short-term memory units, similar to what was used by Wang et al. [47]
- Alternative methods for fine-tuning, such as freezing layers and fine-tuning or re-training the other layers, can be investigated to possibly further improve the performance gain provided by a Guided-GAN.
- A Guided-GAN can be applied to an end-to-end ASR system to operate on embeddings rather than fMLLR features. This can extend the applications of the Guided-GAN to work with more modern architectures as well.
- Investigate if a conditional GAN can be used to improve a Guided-GAN. The ground truth labels of the features can be used as an additional input to the discriminator to improve the output it gives, provided that there are enough examples of each class in the dataset. This may create a stronger discriminator which in turn improves the generator.
- In an exploratory study, the Guided-GAN can be extended to other domains. An image classification network trained on a specific dataset can possibly be adapted to work with samples from a completely different dataset. Images from the new dataset are given to the GAN as input, which then transforms these images to be better classified by the classification network (trained with another dataset). Instead of

using a Guided-GAN to compensate for mismatched audio, the GAN is used to allow an image classification network to be used with a different image dataset that was not used during training.

8.5 Final remarks

The characteristics that contribute to poor quality audio in a call centre environment were studied. A series of ASR systems were trained on different datasets to determine what can be done to improve these systems on mismatched data. Noise perturbation and encoding were the most successful approaches to improve a system that never included these effects in training. Speed and volume perturbation are able to slightly reduce the WER of an MTR system, given that the network has enough capacity to fit the data.

A Guided-GAN was developed to solve many of the shortcomings of existing GANs or enhancement systems. Current techniques require parallel clean and noisy training data. We showed that it is extremely difficult to accurately reproduce a training set that is matched to unknown noise conditions. The developed GAN can utilise a small matched dataset to adapt an existing baseline system trained on good quality data to also perform well on the mismatched data.

The practical relevance of a Guided-GAN is realised when a strong commercial-grade ASR system is required to decode new mismatched data. In commercial applications, it is very expensive to retrain or tweak an ASR system that has already been optimised for a given set of conditions. The strong ASR system can be re-used on the new dataset by using a Guided-GAN. A small training set from the target environment can be collected and manually transcribed without much difficulty. The Guided-GAN can then use this training set to adapt the strong ASR system to improve its performance on the new dataset.

Bibliography

[1] J. Baker, "The DRAGON system – An overview," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 24–29, 1975.

[2] F. Jelinek, L. Bahl, and R. Mercer, "Design of a linguistic statistical decoder for the recognition of continuous speech," *IEEE Transactions on Information Theory*, vol. 21, no. 3, pp. 250–256, 1975.

[3] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE access*, vol. 7, pp. 19 143–19 165, 2019.

[4] X. Lu, S. Li, and M. Fujimoto, "Automatic speech recognition," in *Speech-to-Speech Translation*, Springer, 2020, pp. 21–38.

[5] V. Peddinti, D. Povey, and S. Khudanpur, "A time delay neural network architecture for efficient modeling of long temporal contexts," in *Proceedings of Interspeech*, 2015, pp. 3214–3218.

[6] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, *et al.*, "Deep speech 2: End-to-end speech recognition in English and Mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[7] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A simple data augmentation method for automatic speech recognition," in *Proceedings of Interspeech*, 2019, pp. 2613–2617.

[8] Q. Xu, A. Baevski, T. Likhomanenko, P. Tomasello, A. Conneau, R. Collobert, G. Synnaeve, and M. Auli, "Self-training and pre-training are complementary for speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2021, pp. 3030–3034.

[9] R. Lippmann, E. Martin, and D. Paul, "Multi-style training for robust isolated-word speech recognition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 12, 1987, pp. 705–708.

[10] J. Li, D. Yu, J.-T. Huang, and Y. Gong, "Improving wideband speech recognition using mixed-bandwidth training data in CD-DNN-HMM," in *IEEE Spoken Language Technology Workshop*, 2012, pp. 131–136.

[11] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Proceedings of Interspeech*, 2015, pp. 3586–3589.

[12] M. Doulaty, R. Rose, and O. Siohan, "Automatic optimization of data perturbation distributions for multi-style training in speech recognition," in *IEEE Spoken Language Technology Workshop*, 2016, pp. 21–27.

[13] I. Szöke, M. Skácel, L. Mošner, J. Paliesek, and J. H. Černocky, "Building and evaluation of a real room impulse response dataset," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 4, pp. 863–876, 2019.

[14] A. Sriram, H. Jun, Y. Gaur, and S. Satheesh, "Robust speech recognition using generative adversarial networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2018, pp. 5639–5643.

[15] C. Donahue, B. Li, and R. Prabhavalkar, "Exploring speech enhancement with generative adversarial networks for robust speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2018, pp. 5024–5028.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[17] S. Pascual, A. Bonafonte, and J. Serrà, "SEGAN: Speech enhancement generative adversarial network," in *Proceedings of Interspeech*, 2017, pp. 3642–3646.

[18] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "LibriSpeech: An ASR corpus based on public domain audio books," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2015, pp. 5206–5210.

[19] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, *et al.*, "The Kaldi speech recognition toolkit," in *IEEE Workshop on Automatic Speech Recognition and Understanding*, 2011.

[20] M. Ravanelli, T. Parcollet, and Y. Bengio, "The Pytorch-Kaldi speech recognition toolkit," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2019, pp. 6465–6469.

[21] W. Heymans, M. H. Davel, and C. van Heerden, "Multi-style Training for South African Call Centre Audio," in *Artificial Intelligence Research (SACAIR 2021), Communications in Computer and Information Science*, vol. 1551, Springer, 2022, pp. 111–124.

[22] W. Heymans, M. H. Davel, and C. J. Van Heerden, "Efficient acoustic feature transformation in mismatched environments using a Guided-GAN," *Speech Communication*, under review.

[23] D. Yu and L. Deng, *Automatic Speech Recognition*. Springer, 2016.

[24] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International Conference on Machine Learning*, 2014, pp. 1764–1772.

[25] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2016, pp. 4945–4949.

[26] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2016, pp. 4960–4964.

[27] M. Shahin, B. Ahmed, J. McKechnie, K. Ballard, and R. Gutierrez-Osuna, "A comparison of GMM-HMM and DNN-HMM based pronunciation verification techniques for use in the assessment of childhood apraxia of speech," in *Proceedings of Interspeech*, 2014, pp. 1583–1587.

[28] C. Lüscher, E. Beck, K. Irie, M. Kitza, W. Michel, A. Zeyer, R. Schlüter, and H. Ney, "RWTH ASR systems for LibriSpeech: Hybrid vs Attention," in *Proceedings of Interspeech*, 2019, pp. 231–235.

[29] C. J. Leggetter and P. C. Woodland, "Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models," *Computer Speech & Language*, vol. 9, no. 2, pp. 171–185, 1995.

[30] R. A. Gopinath, "Maximum likelihood modeling with Gaussian distributions for classification," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 2, 1998, pp. 661–664.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 164–167, 224–270, `http://www.deeplearningbook.org`.

[32] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[33] L. Bottou *et al.*, "Stochastic gradient learning in neural networks," *Proceedings of Neuro-Nımes*, vol. 91, no. 8, p. 12, 1991.

[34] 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.

[35] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems*, 1992, pp. 950–957.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[38] L. Prechelt, "Automatic early stopping using cross validation: Quantifying the criteria," *Neural Networks*, vol. 11, no. 4, pp. 761–767, 1998.

[39] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *International Conference on Machine Learning*, 2006, pp. 369–376.

[40] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Machine Learning*, vol. 79, no. 1, pp. 151–175, 2010.

[41] A. Acero and R. M. Stern, "Environmental robustness in automatic speech recognition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1990, pp. 849–852.

[42] Y. Qian, T. Tan, and D. Yu, "An investigation into using parallel data for far-field speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2016, pp. 5725–5729.

[43] L. Mošner, M. Wu, A. Raju, S. H. K. Parthasarathi, K. Kumatani, S. Sundaram, R. Maas, and B. Hoffmeister, "Improving noise robustness of automatic speech recognition via parallel data and teacher-student learning," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2019, pp. 6475–6479.

[44] H. Tang, W.-N. Hsu, F. Grondin, and J. Glass, "A study of enhancement, augmentation, and autoencoder methods for domain adaptation in distant speech recognition," in *Proceedings of Interspeech*, 2018, pp. 2928–2932.

[45] M. L. Seltzer, D. Yu, and Y. Wang, "An investigation of deep neural networks for noise robust speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 7398–7402.

[46] S. Sun, B. Zhang, L. Xie, and Y. Zhang, "An unsupervised deep domain adaptation approach for robust speech recognition," *Neurocomputing*, vol. 257, pp. 79–87, 2017.

[47] K. Wang, J. Zhang, S. Sun, Y. Wang, F. Xiang, and L. Xie, "Investigating generative adversarial networks based speech dereverberation for robust speech recognition," in *Proceedings of Interspeech*, 2018, pp. 1581–1585.

[48] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *International Conference on Learning Representations*, 2016.

[49] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4401–4410.

[50] Y. Choi, M. Choi, M. Kim, J. W. Ha, S. Kim, and J. Choo, "StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8789–8797.

[51] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015, pp. 234–241.

[52] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International Conference on Machine Learning*, 2017, pp. 214–223.

[53] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *International Conference on Learning Representations*, 2018.

[54] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *IEEE International Conference on Computer Vision*, 2017, pp. 2794–2802.

[55] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of Wasserstein GANs," in *International Conference on Neural Information Processing Systems*, 2017, pp. 5769–5779.

[56] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, and S. Gelly, "Are GANs created equal? A large-scale study," in *International Conference on Neural Information Processing Systems*, 2018, pp. 698–707.

[57] K. Kurach, M. Lucic, X. Zhai, M. Michalski, and S. Gelly, "The GAN landscape: Losses, architectures, regularization, and normalization," in *International Conference on Learning Representations*, 2019.

[58] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[59] T. C. Wang, M. Y. Liu, J. Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional GANs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8798–8807.

[60] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4681–4690.

[61] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *Computing Research Repository*, 2014.

[62] M. A. Haidar and M. Rezagholizadeh, "Fine-tuning of pre-trained end-to-end speech recognition with generative adversarial networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2021, pp. 6204–6208.

[63] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? A new look at signal fidelity measures," *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, 2009.

[64] D. Dean, S. Sridharan, R. Vogt, and M. Mason, "The QUT-NOISE-TIMIT corpus for evaluation of voice activity detection algorithms," in *Proceedings of Interspeech*, 2010, pp. 3110–3113.

[65] D. Snyder, G. Chen, and D. Povey, *MUSAN: A Music, Speech, and Noise Corpus*, arXiv:1510.08484v1, 2015.

[66] I. Siegert, A. F. Lotz, L. L. Duong, and A. Wendemuth, "Measuring the impact of audio compression on the spectral quality of speech data," *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung*, pp. 229–236, 2016.

[67] E. ETSI, "300 961 v7. 0.2 (1999)," *Digital cellular telecommunications system (Phase 2+); Full rate speech; Transcoding (GSM 06.10 version 7.0. 2 Release 1998)*, 1999.

[68] J. Van Meggelen, R. Bryant, and L. Madsen, *Asterisk: The Definitive Guide: Open Source Telephony for the Enterprise*. O'Reilly Media, 2019.

[69] M. J. Gales, "Maximum likelihood linear transformations for HMM-based speech recognition," *Computer Speech & Language*, vol. 12, no. 2, pp. 75–98, 1998.

[70] V. Mitra, H. Franco, M. Graciarena, and A. Mandal, "Normalized amplitude modulation features for large vocabulary noise-robust speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012, pp. 4117–4120.

[71] G. Saon, H. Soltau, D. Nahamoo, and M. Picheny, "Speaker adaptation of neural network acoustic models using i-vectors," in *IEEE Workshop on Automatic Speech Recognition and Understanding*, 2013, pp. 55–59.

[72] D. Povey, G. Cheng, Y. Wang, K. Li, H. Xu, M. Yarmohammadi, and S. Khudanpur, "Semi-orthogonal low-rank matrix factorization for deep neural networks," in *Proceedings of Interspeech*, 2018, pp. 3743–3747.

[73] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust DNN embeddings for speaker recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2018, pp. 5329–5333.

Appendix A

Supplemental content

We provide supplementary information that was used to train and optimise the GAN models.

A.1 Appendix: Chapter 6

A.1.1 Guided-GAN hyperparameters on clean WAV49-encoded LibriSpeech corpus

This section provides additional information about the hyperparameter choices and ranges searched for the Guided-GAN networks on the clean WAV49-encoded LibriSpeech corpus. The results for these networks are shown in Section 6.3.1. Table A.1 and A.2 show the hyperparameters for the Guided-GAN using the SN-GAN loss function with the fully-convolutional network architecture for the *train-clean-100* and *train-clean-e* acoustic models, respectively. Tables A.3 and A.4 show the hyperparameters for the same setup, but using the encoder-decoder network architecture. The hyperparameters for the

WGAN-GP and NS-GAN networks are shown in Tables A.5 and A.6, respectively.

Table A.1: Hyperparameters for the Guided-GAN using 'F-C' network architecture and SN-GAN loss on the clean WAV49-encoded LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-100* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.25	0 - 0.3
Loss function	SN-GAN + AM-loss	-
Optimiser	Adam	-
G learning rate	0.000 3	0.000 1 - 0.000 5
D learning rate	0.000 05	0.000 025 - 0.000 2
Batch size	1 024	256 - 2 048
Epochs	20	10 - 30
Acoustic model lambda	1	0 - 10
Features	fMLLR	-

Table A.2: Hyperparameters for the Guided-GAN using 'F-C' network architecture and SN-GAN loss on the clean WAV49-encoded LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-e* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.25	0 - 0.3
Loss function	SN-GAN + AM-loss	-
Optimiser	Adam	-
G learning rate	0.000 3	0.000 1 - 0.000 5
D learning rate	0.000 05	0.000 025 - 0.000 2
Batch size	1 024	256 - 2 048
Epochs	20	10 - 30
Acoustic model lambda	1	0 - 10
Features	fMLLR	-

Table A.3: Hyperparameters for the Guided-GAN using 'E-D' network architecture and SN-GAN loss on the clean WAV49-encoded LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-100* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.3	0 - 0.5
Loss function	SN-GAN + NLL-loss	-
Optimiser	Adam	-
G learning rate	0.000 3	0.000 1 - 0.000 5
D learning rate	0.000 01	0.000 005 - 0.000 2
Batch size	2 048	512 - 4 096
Epochs	20	20 - 30
Acoustic model lambda	1	0 - 10
Features	fMLLR	-

Table A.4: Hyperparameters for the Guided-GAN using 'E-D' network architecture and SN-GAN loss on the clean WAV49-encoded LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-e* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.3	0 - 0.5
Loss function	SN-GAN + NLL-loss	-
Optimiser	Adam	-
G learning rate	0.000 3	0.000 1 - 0.000 5
D learning rate	0.000 01	0.000 005 - 0.000 2
Batch size	2 048	512 - 4 096
Epochs	20	20 - 30
Acoustic model lambda	1	0 - 10
Features	fMLLR	-

Table A.5: Hyperparameters for the Guided-GAN using 'E-D' network architecture and WGAN-GP loss on the clean WAV49-encoded LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-100* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.3	0 - 0.5
Loss function	WGAN-GP + NLL-loss	-
Optimiser	Adam	-
G learning rate	0.000 1	0.000 05 - 0.000 5
D learning rate	0.000 1	0.000 05 - 0.000 5
Batch size	1 024	512 - 4 096
Epochs	20	20 - 30
Acoustic model lambda	5	0 - 10
Gradient penalty weight	2	0 - 10
Features	fMLLR	-

Table A.6: Hyperparameters for the Guided-GAN using 'E-D' network architecture and NS-GAN loss on the clean WAV49-encoded LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-100* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.3	0 - 0.5
Loss function	NS-GAN + NLL-loss	-
Optimiser	Adam	-
G learning rate	0.000 1	0.000 05 - 0.000 5
D learning rate	0.000 1	0.000 05 - 0.000 5
Batch size	512	128 - 1 024
Epochs	20	20 - 30
Acoustic model lambda	5	0 - 10
Features	fMLLR	-

A.1.2 Guided-GAN hyperparameters on noisy WAV49-encoded LibriSpeech corpus

This section shows the hyperparameters for the GAN networks trained in Section 6.3.2. Tables A.7 and A.8 show the hyperparameters for the networks trained using the *train-clean-100* and *train-clean-e* acoustic models, respectively. Both GANs used the fully-convolutional network architecture.

Table A.7: Hyperparameters for the Guided-GAN on the noisy WAV49-encoded LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-100* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.25	0 - 0.3
Loss function	SN-GAN + AM-loss	-
Optimiser	Adam	-
G learning rate	0.000 35	0.000 1 - 0.000 5
D learning rate	0.000 05	0.000 025 - 0.000 2
Batch size	1 024	512 - 2 048
Epochs	20	20
Acoustic model lambda	1	0 - 5
Features	fMLLR	-

Table A.8: Hyperparameters for the Guided-GAN on the noisy WAV49-encoded LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-e* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.25	0 - 0.3
Loss function	SN-GAN + AM-loss	-
Optimiser	Adam	-
G learning rate	0.000 5	0.000 1 - 0.000 6
D learning rate	0.000 05	0.000 025 - 0.000 2
Batch size	2 048	512 - 4 096
Epochs	20	20
Acoustic model lambda	1	0 - 5
Features	fMLLR	-

A.1.3 Guided-GAN hyperparameters for the resource-scarce experiment

This section shows the hyperparameters for the GAN networks trained in Section 6.3.3. Tables A.9 and A.10 show the hyperparameters for the networks trained using the *train-clean-100* and *train-clean-e* acoustic models, respectively. Both GANs used the fully-convolutional network architecture.

Table A.9: Hyperparameters for the Guided-GAN on the *train-other-e-100* subset of the LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-100* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.25	0 - 0.3
Loss function	SN-GAN + AM-loss	-
Optimiser	Adam	-
G learning rate	0.000 3	0.000 1 - 0.000 5
D learning rate	0.000 05	0.000 025 - 0.000 2
Batch size	2 048	512 - 4 096
Epochs	20	20
Acoustic model lambda	1	0 - 5
Features	fMLLR	-

Table A.10: Hyperparameters for the Guided-GAN on the *train-other-e-100* subset of the LibriSpeech corpus. The acoustic model used for this network was trained on the *train-clean-e* subset.

Hyperparameter	**Value**	**Searched**
Dropout in D	0.25	0 - 0.3
Loss function	SN-GAN + AM-loss	-
Optimiser	Adam	-
G learning rate	0.000 3	0.000 1 - 0.000 5
D learning rate	0.000 05	0.000 025 - 0.000 2
Batch size	2 048	512 - 4 096
Epochs	20	20
Acoustic model lambda	1	0 - 5
Features	fMLLR	-