BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

SPEECH ENHANCEMENT WITH CYCLE-CONSISTENT NEURAL NETWORKS

ODSTRAŇOVÁNÍ ŠUMU POMOCÍ NEURONOVÝCH SÍTÍ S CYKLICKOU KONZISTENCÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE
PAVOL KARLÍK

SUPERVISOR
VEDOUCÍ PRÁCE
Ing. ŽMOLÍKOVÁ KATEŘINA

BRNO 2020

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Master's Thesis Specification

23134

Student:	**Karlík Pavol, Bc.**
Programme:	Information Technology Field of study: Information Systems
Title:	**Speech Enhancement with Cycle-Consistent Neural Networks**
Category:	Speech and Natural Language Processing

Assignment:

1. Get acquainted with the problem of speech enhancement using neural networks.
2. Get acquainted with the generative-adversarial neural networks, CycleGANs and their use for speech enhancement.
3. Implement the method, train, evaluate and compare with published results.
4. Further analyze obtained results.
5. Suggest ways to improve the results or extend the method.

Recommended literature:

- Meng, Zhong, Jinyu Li, and Yifan Gong. "Cycle-consistent speech enhancement." *arXiv preprint arXiv:1809.02253* (2018).
- dle doporučení vedoucí

Requirements for the semestral defence:

- Items 1 and 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Žmolíková Kateřina, Ing.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2019
Submission deadline:	June 3, 2020
Approval date:	November 5, 2019

Abstract

Deep neural networks (DNNs) have become a standard approach for solving problems of speech enhancement (SE). The training process of a neural network can be extended by using a second neural network, which learns to insert noise into a clean speech signal. Those two networks can be used in combination with each other to reconstruct clean and noisy speech samples. This thesis focuses on utilizing this technique, called cycle-consistency. Cycle-consistency improves the robustness of a network without modifying the speech-enhancing neural network, as it exposes the SE network to a much larger variety of noisy data. However, this method requires input-target training data pairs, which are not always available. We use generative adversarial networks (GANs) with cycle-consistency constraint to train the network using unpaired data. We perform a large number of experiments using both paired and unpaired training data. Our results have shown that adding cycle-consistency improves the models' performance significantly.

Abstrakt

Hlboké neurónové siete sa bežne používajú v oblasti odstraňovania šumu. Trénovací proces neurónovej siete je možné rozšíriť využitím druhej neurónovej siete, ktorej cieľom je vložiť šum do čistej rečovej nahrávky. Tieto dve siete sa môžu spolu využiť k rekonštrukcii pôvodných čistých a zašumených nahrávok. Táto práca skúma efektivitu tejto techniky, zvanej cyklická konzistencia. Cyklická konzistencia zlepšuje robustnosť neurónovej siete bez toho, aby sa daná sieť akokoľvek modifikovala, nakoľko vystavuje sieť na odstraňovanie šumu rôznorodejšiemu množstvu zašumených dát. Avšak, táto technika vyžaduje trénovacie dáta skladajúce sa z párov vstupných a referenčných nahrávok. Tieto dáta niesu vždy dostupné. Na trénovanie modelov s nepárovanými dátami využívame generatívne neurónové siete s cyklickou konzistenciou. V tejto práci sme vykonali veľké množstvo experimentov s modelmi trénovanými na párovaných a nepárovaných dátach. Naše výsledky ukazujú, že využitie cyklickej konzistencie výrazne zlepšuje výkonnosť modelov.

Keywords

speech enhancement, GAN, generative adversarial networks, deep learning, cycle-consistency

Kľúčové slová

odstraňovanie šumu, GAN, generatívne neurónové siete, hlboké učenie, cyklická konzistencia

Reference

KARLÍK, Pavol. *Speech Enhancement with Cycle-Consistent Neural Networks.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Žmolíková Kateřina

Rozšírený abstrakt

Systémy rozpoznávania reči (ASR systémy) umožňujú interpretovať ľudskú reč v podobe textu. V posledných desaťročiach bolo v tejto oblasti dosiahnutého značného pokroku. Využitie týchto systémov je rozmanité. V dnešnej dobe sa používajú napríklad v prístrojoch pre automatizáciu domácnosti.

Jedným z hlavných problémov ASR systémov je prítomnosť šumu a reverberácie v rečových nahrávkach. Tieto artefakty výrazne zhoršujú schopnosť ASR systému vytvoriť korektnú textovú reprezentáciu hovorenej reči. Existuje veľa algoritmov nazlepšenie kvality rečovej nahrávky. Väčšina dnešných techník využíva práve *neurónové siete*.

Táto práca sa zaoberá experimentovaním s rozsiahlymi modelmi neurónových sietí na zlepšenie kvality rečovej nahrávky. Práca je založená na publikácii, v ktorej sa experimentuje s neurónovými sieťami s *cyklickou konzistenciou* pre zlepšenie kvality rečových nahrávok. Cyklická konzistencia modifikuje chybovú funkciu na trénovanie siete. V sieti s cyklickou konzistenciou figuruje druhá neurónová sieť, ktorá vykonáva presne opačnú úlohu, ako prvá neurónová sieť. V našom prípade sa jedná o sieť, ktorá do čistej rečovej nahrávky vkladá šum. Pomocou týchto dvoch neurónových sietí je možné zrekonštruovať pôvodnú čistú a zašumenú nahrávku. Chyba rekonštrukcie je zakomponovaná do celkovej chybovej funkcie modelu.

Často je ťažké zohnať dostatočne veľkú dátovú sadu, ktorá obsahuje páry vstup-referenčný výstup. Avšak, neurónové siete sa dajú trénovať aj s nepárovanými dátami pomocou tzv. *generatívnych adversných sietí* (GAN). Generatívne adversné siete sa skladajú z dvoch neurónových sietí - generátoru a diskriminátoru. Cieľom generátoru je generovať vzorky, ktoré sa podobajú vzorkám z trénovacej sady. Diskriminátor sa snaží vzorky z trénovacej sady a vygenerované vzorky korektne rozlíšiť. Architektúra *CycleGAN*, ktorú používame v tejto práci, je GAN architektúra s využitím cyklickej konzistencie. CycleGAN obsahuje štyri neurónové siete - dva diskriminátory a dva generátory. Ďalej rozširuje chybovú funkciu o tzv. *chybu identity*, ktorá má zaručiť to, že sa vstup neurónovej siete od jej výstupu nebude príliš líšiť.

V tejto práci je popísaný vplyv šumu a reverberácie v oblasti rozpoznávania reči. Ďalej sú detailne popísané neurónové siete so zameraním na ich princíp a spôsob trénovania, rekurentné neurónové siete, konvolučné neurónové siete a princíp cyklickej konzistencie. Predstavíme generatívne neurónové siete (GAN) a CycleGAN architektúru. Krátko je popísaná použitá dátová sadu *CHiME-3* a detaily implementácie. Veľká časť práce je venovaná podrobnému popisu experimentov s rôznymi modelmi, ktoré využívajú princípy generatívnych adversných sietí a cyklickej konzistencie.

Pre implementáciu modelov neurónových sietí bol použitá *Python* knižnica pre strojové učenie, *PyTorch*. Vstupom neurónových sietí sú normalizované logaritmované *mel filterbank* koeficienty. Výstupom sú tieto koeficienty bez normalizácie. Neurónová sieť sa skladá z dvojvrstvovej *Long-Short Term Memory* siete, ktorá je nasledovaná tzv. plne prepojenou vrstvou. Neurónová sieť - diskriminátor - je zložená z dvoch plne prepojených vrstiev. Pre natrénovanie jednotlivých modelov bolo venovaného veľké množstvo času hľadaním najvhodnejších trénovacích parametrov. CycleGAN bol trénovaný s využitím rôznych regularizačných a trénovacích techník, ktoré zlepšili výkonnosť modelu. Pre zhodnotenie kvality modelov bola použitá metrika *word error rate* (WER), ktorá porovnáva text vygenerovaný ASR systémom s referenčným textom.

Najprv bola natrénovaná základná neurónová sieť, ktorá slúžila ako referenčný model — *baseline*. Pre model s cyklickou konzistenciou (CSE) bola najprv natrénovaná sieť na vkladanie šumu. Tieto dva natrénované modely sa potom trénovali spolu v CSE. Pre

náš CycleGAN model, nazývaný *ACSE*, sme taktiež najprv inicializovali dve neurónové siete — generátory. Vstupom a výstupom bola tá istá čistá alebo zašumená nahrávka, pričom referenčná nahrávka nebola normalizovaná. Natrénované modely sa potom využili v trénovaní ACSE. Ďalej sme pre dosiahnutie lepších výsledkov pretrénovali *akustický model* (AM). Akustický model je jedna z komponent systému rozpoznávania reči, ktorá vytvára štatistickú reprezentáciu jednotlivých fonémov. K pretrénovaniu akustického modelu boli použité dáta z trénovacej sady, ktoré boli spracované modelmi na odstraňovanie šumu. Pomocou natrénovaného AM bolo opäť vyhodotené WER jednotlivých sietí.

Baseline model dosiahol relatívneho zlepšenia WER (RWERR) o 17.40 % oproti nespracovaným nahrávkam. S využitím cyklickej konzistencie sme dosiahli ďalšieho zlepšenia RWERR na 20.97 %. Vďaka podrobnému ladeniu trénovacích parametrov sme dosiahli lepších výsledkov, ako v publikácii, v ktorej baseline a CSE modely dosiahli 12.33 % a 19.60 % RWERR oproti nespracovaným dátam. Model trénovaný na nepárovaných dátach, ACSE, dosiahol relatívneho zlepšenia o 12.71 %, pričom ACSE v publikácii dosiahlo zlepšenia o 6.69 %. Okrem ladenia trénovacích parametrov sme zaviedli ďalšie trénovacie triky pre zlepšenie konvergencie GANov, ako napríklad využívanie histórie generovaných vzorkov pre trénovanie diskriminátorov. S pretrénovaným akustickým modelom sme dosiahli 46.86 % RWERR, oproti 38.18 % RWERR v publikácii. Oproti publikácii sme ďalej pretrénovali AM pomocou baseline a CSE, ktoré dosiahli 30.83 % a 33.50 % RWERR.

Skúmali sme vplyv adversného trénovania a chyby identity na výkonnosť modelov. Výkon baseline modelu dosiahol 19.21 % (35.38% s pretrénovaným AM) RWERR pridaním chyby identity. Pre CSE model sme tiež pridali chybu identity, čím sme dosiahli 22.92 % (34.19 % s pretrénovaným AM) RWERR. Ak v CSE s chybou identity využijeme adversné trénovanie, dôjde k zhoršeniu RWERR o 0.26 %, ale s pretrénovaným AM sa RWERR zlepší o 1.91% oproti CSE s chybou identity.

V tejto práci sme implementovali viaceré modely neurónových sietí založené na princípoch cyklickej konzistencie a generatívnych adversných neurónových sieťach. Naše experimenty preukázali, že cyklická konzistencia pomáha zlepšiť výkonnosť modelu na zlepšenie kvality rečových nahrávok. Ďalej sme zistili, že zakomponovanie chyby identity do chybovej funkcie obecne pomáha zvýšiť výkonnosť modelov nielen u ACSE, ale aj u baseline a CSE. Adversné trénovanie má väčší dopad na výsledné WER, keď sa využije pretrénovanie akustického modelu. Lepších výsledkov by molo možné dosiahnuť nahradením LSTM siete inou, napríklad tzv. *Transformer Neural Network* alebo *Dual-Path Recurrent Neural Network*.

Speech Enhancement with Cycle-Consistent Neural Networks

Declaration

I hereby declare that this Diploma thesis was prepared as an original work by the author under the supervision of Ing. Kateřina Žmolíková. All the relevant sources, which were used during the preparation of this thesis, are properly cited and included in the list of references.

. . . .
Pavol Karlík
June 1, 2020

Acknowledgements

I would like to express my deepest gratitude to my supervisor Ing. Žmolíková for her incredible guidance, valuable advice, encouragement, and immense knowledge.

Contents

Chapter 1

Introduction

Automatic speech recognition (ASR) is a widely used technology that allows transcription of a recorded speech utterance into a corresponding sequence of words. The field has been intensively researched in the past decades, with significant advances being made through the years. These improvements led to a surge in the usage of intelligent human-machine speech communication systems, such as virtual speech assistants or interactive voice response systems.

Despite significant advances in this area, there still are certain factors that limit the performance of such systems. Some of the central issues which reduce speech intelligibility are reverberation and ambient noise. Those artifacts are picked up along with spoken utterances when being captured by a microphone which results in corrupted speech. There are many speech enhancement (SE) and ASR techniques to detect and combat the effects of noise and reverberation [36, 78, 81].

Current state-of-the-art speech enhancement methods primarily employ artificial neural networks (ANNs) [82]. Recently, a new framework of ANNs has been proposed using *adversarial training* called **generative adversarial network** (GAN) [21]. In GANs, two neural networks are pitted against each other, each attempting to reach its objective, which is 'adversary' to the other network. The generative adversarial network essentially models the distribution of a given dataset. One of the modifications of the aforementioned model uses *cycle-consistency* for unpaired data training to further improve the architecture called *CycleGAN* [84]. CycleGAN learns to transform samples from one domain to another. The cycle-consistency constraint is implemented by adding another neural network. This objective is enforced by making the networks be able to reconstruct images from source domain X to target domain Y and vice versa, such that $F(G(X)) \approx X$ and $G(F(X)) \approx X$. However, its potential has mostly been explored in the image processing field. This thesis is based on a research paper recently published by Meng et al. (2018) [49] which proposes a framework inspired by [84] for a speech enhancement task. In the paper, it was demonstrated that the use of cycle-consistency paired with generative adversarial networks can yield significant improvements over standard ANN models.

This thesis is organized as follows: Chapter 2 explains the problem of noise and reverberation in the speech recognition field. Chapters 3 and 4 overview neural networks with a particular focus on GANs with cycle-consistency. Chapter 5 briefly introduces the dataset used in this thesis. Chapter 6 contains a short overview of the tools and frameworks used for the experiments that are described in Chapter 7. In the end, a conclusion is drawn from the executed experiments and possible future improvements are discussed.

Chapter 2

Speech Enhancement and Noise Reduction

Despite the widespread use of automatic speech recognition (ASR) technologies in various systems, there is a large number of challenges that such systems need to handle in order to be applicable. When a signal is captured by a microphone, the picked up signal can get corrupted, causing it to lose intelligibility and quality. Such an altered signal might then be erroneously processed by the ASR system, causing it to transcribe speech with less accuracy than if it were to process a clean speech signal. One of the fields that pursues this problem is **speech enhancement**.

Speech enhancement (SE) techniques aim to improve a degraded speech signal by using signal processing tools [44]. The specific SE techniques heavily depend on the environment from which target sources come from and a number of microphones available.

This Chapter describes the difficulties the noise and reverberation cause in the speech recognition field and reviews the current speech enhancement methods used to diminish these problems. For more detailed knowledge, the readers can refer to [41, 44, 78, 82].

2.1 Problem Introduction

Noise in a speech signal generally represents an unwanted modification that a signal may be subjected to when being captured or processed. Various types of noises exist, depending on their statistical properties and on the way they modify a signal. According to the spectral distribution, the noises can be grouped into two categories:

- **stationary noise** - a stationary noise keeps constant spectral distribution over time (e.g., white noise). The problem of eliminating short-term stationary additive noise has been solved for decades [44].
- **non-stationary noise** - a large amount of real-world environmental noise is non-stationary (e.g., people speaking in the background). Such noise is much more difficult to suppress because its statistics change over time.

In the past years, the research in the speech enhancement field has not only been focused on enhancing speech signals corrupted by non-stationary noise but also signals affected by *reverberation* [36, 81]. Reverberation is a collection of sound waves reflected from surfaces in an enclosed space — a result of a signal bouncing off physical objects, such as walls.

The reflected signals differ in delay and amplitude, which makes the transition between phonemes in the signal less distinct.

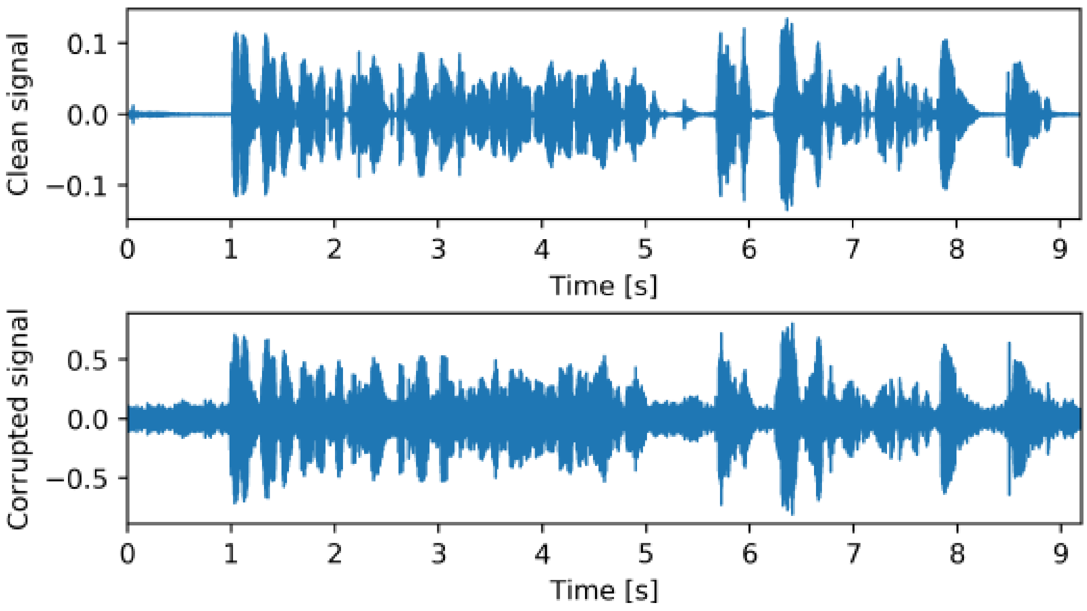


Figure 2.1: Comparison of clean and corrupted speech signal.

Mathematically, a corrupted speech signal can be defined as a linear convolution, written as

$$y_t = x_t \circledast h_t + a_t, \tag{2.1}$$

where x_t is the raw speech signal, h_t is the room impulse response – reverberation, a_t is the additive noise and y_t is the resulting degraded speech signal observed by a microphone. When applying Short-Time Fourier Transform (STFT), the room impulse response (RIR) smears across several frames because the length of RIR is often much longer than the analysis window size. The corrupted speech in the STFT domain, $\tilde{y}_{t,f}$, can therefore be modeled as

$$\tilde{y}_{t,f} \approx \sum_{d=0}^{D-1} \tilde{x}_{t-d,f}\tilde{h}_{d,f} + \tilde{a}_{t,f}, \tag{2.2}$$

where d is the frame delay, f is the frequency bin and $\tilde{h}_{d,f}$ is STFT of the room impulse response h_t corresponding to the d-th frame delay. As we can see above, the signal corruption is not linear due to the introduced frame delay [3].

2.2 Speech Enhancement Methods

Before introducing standard speech enhancement techniques, often-used features in these methods will be described. The standard described approaches use the *short-time Fourier analysis modification-synthesis* (AMS) framework [2]. The AMS-based SE methods consist of three steps:

- processing a speech signal using STFT analysis
- modifying the magnitude spectrum

- taking the inverse STFT, followed by phase reconstruction algorithm (e.g., Griffin-Lim [23]

Using AMS, the phase of the original signal is used for phase reconstruction, as it is supposed that the phase modification is unnecessary [76]. However, it has since been shown that a better phase estimation during speech enhancement can indeed lead to a significant improvement of speech quality [57].

2.2.1 Features Used in Speech Enhancement

Current speech enhancement techniques primarily utilize some higher-level feature and/or operate on the spectral domain. However, it has been shown that using raw audio in combination with neural network can yield as good [68] or even better [56] results than when using higher level features. Below, the most commonly used features are described.

Power Spectral Density

One of the common features used in speech enhancement is power spectral density (PSD). PSD describes the signal's distribution of power over frequency. Approximated from (2.2), the PSD of a corrupted speech signal can be formulated as

$$|\tilde{y}_{t,f}|^2 \approx \sum_{d=0}^{D-1} |\tilde{x}_{t-d,f}|^2 |\tilde{h}_{d,f}|^2 + |\tilde{a}_{t,f}|^2. \tag{2.3}$$

Mel Filter Bank Coefficients

Mel filter bank coefficients (MFBs) are computed from applying *mel-scaled filter bank*[1] to the power spectrum of the signal. The reason to use a *mel scale* instead of the frequency scale is that the mel scale mimics the human perception of sound, which is non-linear. The conversion formula from hertz to mels can be defined as

$$m = 2595 \log_{10}\left(1 + \frac{f}{f_0}\right), \tag{2.4}$$

where f_0 is a *corner frequency* — a point from which the scale changes from linear to logarithmic. The value typically ranges from 600 to 1000, depending on the specific formula.

Taking PSD as described in (2.3), the k-th mel filter bank output can be obtained using

$$Mel(\tilde{y}_{t,k}) \approx \sum_{d=0}^{D-1} mel(\tilde{x}_{t-d,k}) mel(\tilde{h}_{d,k}) + mel(\tilde{a}_{t,k}), \tag{2.5}$$

in which *mel* function can be defined as

$$mel(\tilde{z}_{n,k}) \approx \mathbf{B}[k] \cdot |\tilde{z}_{n,f}|^2, \tag{2.6}$$

where $\mathbf{B}[k]$ is the weight of the discrete Fourier transform bin f in the k-th mel bin $\mathbf{B}$.

[1]An array of triangular filters linearly spaced in mel scale - https://labrosa.ee.columbia.edu/doc/HTKBook21/node54.html

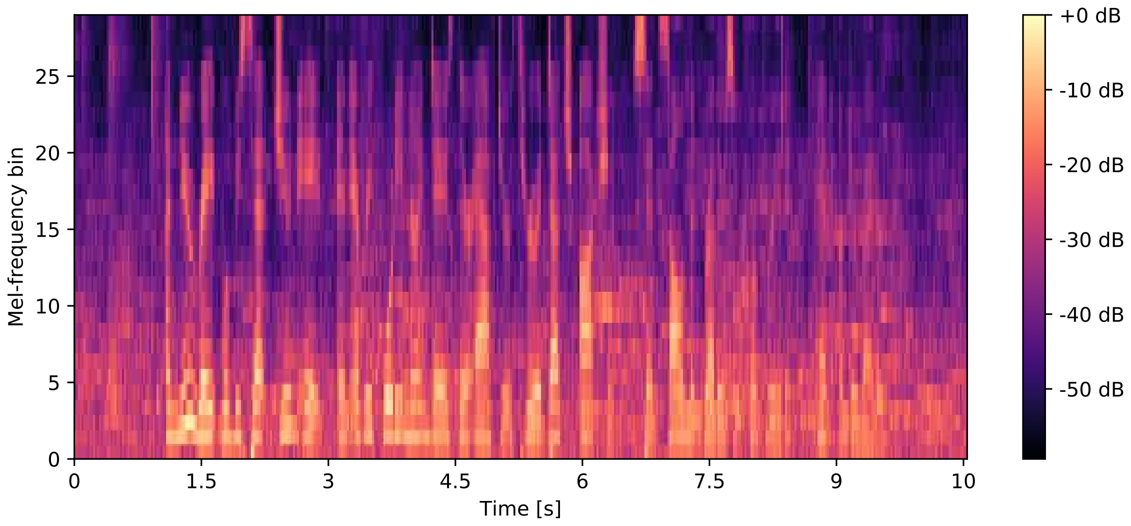


Figure 2.2: Mel-frequency spectrogram of corrupted speech signal.

Additionally, an element-wise log operation is applied to the function when using MFBs as a feature. This thesis uses log mel filterbank coefficients as a feature of choice, since the feature was used in [49] as well.

Mel-Frequency Cepstral Coefficients

Besides mel filter bank features, SE and ASR systems often derive Mel-frequency cepstral coefficients (MFCCs). From (2.2.1), the i-th MFC coefficient can be written as

$$Dct(\tilde{y}_{t,i}) \approx dct(\tilde{x}_{t,i})dct(\tilde{h}_{0,i}) + dct(M_{t,i}), \tag{2.7}$$

in which dct is defined as

$$dct(f_{t,i}) = \mathbf{C}[i]\log(mel(f_{t,k})), \tag{2.8}$$

where $\mathbf{C}$ denotes a discrete cosine transformation matrix, and

$$M_{t,i} = 1 + \frac{\sum_{d=0}^{D-1} mel(\tilde{y}_{t-d,k})mel(\tilde{h}_{d,k}) + mel(\tilde{a}_{t,k})}{mel(\tilde{y}_{t,k})mel(\tilde{h}_{0,k})}. \tag{2.9}$$

2.2.2 Standard Speech Enhancement Solutions

The most common speech enhancement methods are *spectral subtraction*, methods based on *linear filtering* and methods utilizing *neural networks*.

- **spectral subtraction** - spectral subtraction is historically one of the first algorithms proposed for single channel speech enhancement [10]. In this method, the average of the noise spectrum is estimated during a non-speech period. Then, the estimated spectrum is subtracted from the noisy signal spectrum. The phase information is kept unchanged.

- **linear filter-based methods** - linear filtering approaches attempt to enhance the corrupted speech signal in the STFT or time domain. In contrast to spectral subtraction, linear filtering exploits both the amplitude and the phase of the speech signal.

The phase information is especially useful in terms of removing reverberation — which is, essentially, a superposition of several time-shifted and attenuated versions of clean signal [81]. Additionally, these methods can take advantage of multiple microphone sources of the same speech signal.

- **neural network-based methods** - these approaches vastly outperform standard speech enhancement techniques and their usage is currently considered a standard [78, 82]. Both high-level features and raw speech can be used as an input and output of the network. Speech enhancement methods using ANNs will be discussed in more detail in 3.

Readers interested in various forms of SE methods can refer to [41, 44, 78] for more details.

While the ASR performance in difficult noisy and reverberant conditions has significantly improved over the past years [78, 80], there still are certain areas of focus where such systems perform poorly. It has been observed that, when ASR systems are given a challenging environment with distant noisy and overlapping conversational speech, the system performance suffers significantly [6].

2.3 Automatic Speech Recognition

Enhanced speech can improve the quality of many devices, some of which require to transcribe the spoken utterance to a sequence of words, such as intelligent home assistants and devices for the hearing impaired. This is done by *automatic speech recognition* (ASR) systems. In this work, we measure the performance of speech enhancement models based on ASR results. This section provides a brief summary of how ASR works.

Generally, the ASR pipeline consists of two main parts — the *back end* and the *front end*. A standard ASR framework is shown in Figure 2.3. The front end pre-processes the recorded speech utterance and extracts features, most commonly MFCC and Constrained Maximum Likelihood Linear Regression (CMLLR) [15] features. Those extracted features are then used in the back end portion of ASR. The back end consists of two major components, an *acoustic model* and a *language model*.

Acoustic Model

The role of an acoustic model is to represent a relationship between audio and phoneme units. Acoustic model outputs the probability of a feature sequence given a sequence of words.

Formerly, the acoustic model was primarily composed via Gaussian mixture model based Hidden Markov Model (GMM-HMM) approach, which uses CMLLR features. Nowadays, deep neural networks (DNNs) in conjunction with HMMs (DNN-HMM) are used for acoustic modeling [13], as they significantly outperform GMM-HMM acoustic models [27].

Language Model

The language model then estimates the likelihood of a sequence of words. It provides context to distinguish between similar-sounding words and phrases.

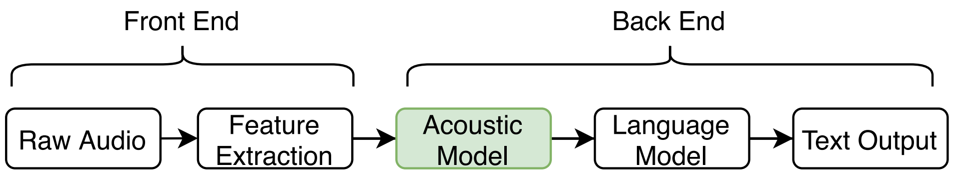


Figure 2.3: Speech-to-text pipeline.

Using the scores from the acoustic and the language model, a most likely sequence of words given the sequence of feature vectors is computed, called *hypothesis*. In modern ASR systems, such as Kaldi [64], the work of the acoustic model and the language model is combined by *weighted finite state transducers* [53].

Chapter 3

Artificial Neural Networks

The application of neural networks (NNs) has become a standard in many fields. Larger amount of computational power has become available over the years, which led to increased interest in research and use of NNs in various systems. Current state-of-the-art speech recognition and speech enhancement algorithms employ neural networks to improve their performance [38, 78]. In this work, we implement three models for enhancing speech signals: a network with recurrent layers, that same model expanded with cycle-consistency constraint, and a generative adversarial network (GAN). However, this Chapter only explains the fundamentals of neural networks and cycle-consistency. For a concise description of GANs, see Chapter 4.

In this Chapter, we outline the structure of a standard neural network model and a training process of a network. Followed by that, we describe classes of neural networks, called *recurrent neural networks* (RNNs and *convolutional neural networks* (CNNs). Lastly, we introduce an interesting concept, called *cycle-consistent* neural networks. The main sources of information for this Chapters were Goodfellow's *Deep learning* book [38], Olah's web article, *Understanding LSTM Networks* [55] and Bishop's *Pattern Recognition and Machine Learning* [9] book.

3.1 Structure of a Neural Network

A neural network consists of several simple elements called *neurons*. Neurons receive input, change their internal state accordingly, and apply *activation function* to produce output. The network consists of several layers made of interconnected neurons. The significance of a neuron within the model is specified by its *weights*, which are adjusted during the training period.

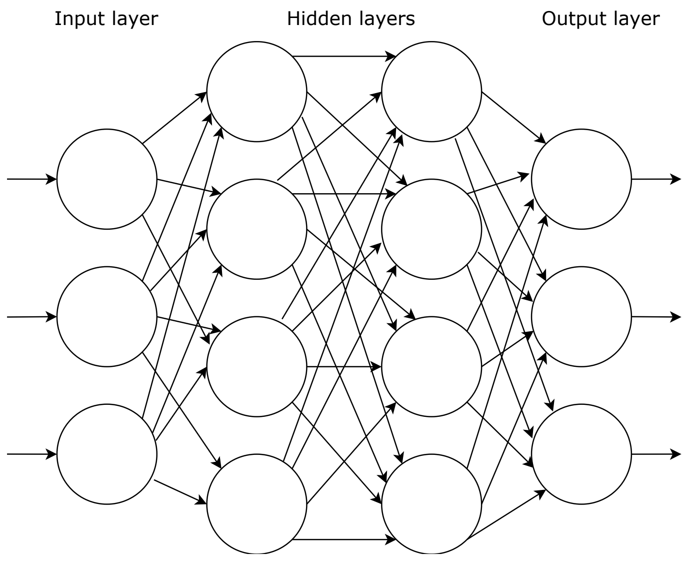


Figure 3.1: Simple feedforward ANN with two hidden layers.

As seen in Figure 3.1, the outputs of the neurons are connected to the neuron inputs in the next layer. The initial layer is called *input layer*, the final layer is called *output layer* and the layers in between are called *hidden layers*.

The neuron output y is defined as

$$y = \tilde{f}(\sum_{i=1}^{N} x_i w_i + w_0), \tag{3.1}$$

where $\tilde{f}$ is an activation function, x_i is the output of the i-th neuron from the previous layer, w_i is its weight and w_0 is *bias*[1].

Neural networks can model problems that are highly non-linear. The non-linear relationships within the network can be formed due to the presence of non-linear *activation functions* in hidden layers. Besides non-linearity, other core properties of these functions are that they are continuous and differentiable. Those characteristics are required for training the network via *backpropagation* algorithm, which is explained in subsection 3.2.2.

Hyperbolic Tangent

Hyperbolic tangent (tanh) is often used as an activation function. The function is defined as

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1, \tag{3.2}$$

$$\frac{d}{dx} f(x) = 1 - f(x)^2. \tag{3.3}$$

[1]Each layer has a vector of *biases* with each scalar corresponding to a specific neuron. Biases are independent on the input.

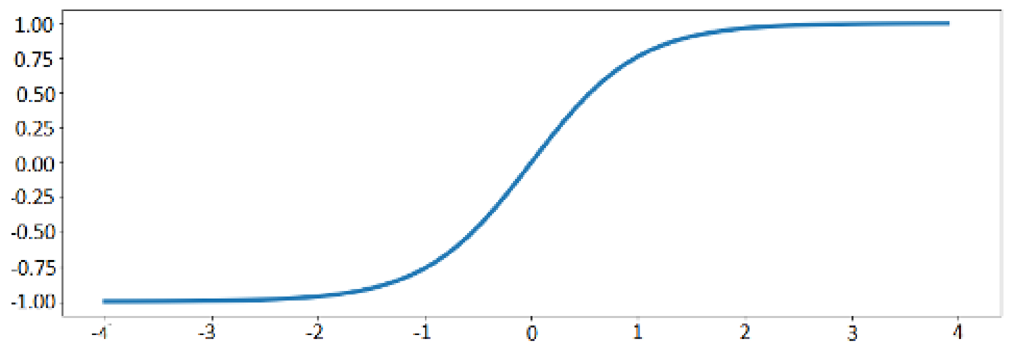


Figure 3.2: The tanh function.

Figure 3.2 shows a plot of the tanh function. The main drawback of this function is that it can lead to **vanishing gradient**. This is caused by small derivatives of the function and is especially present in multi-layer networks — which is a large majority. The gradient decreases exponentially during *backpropagation*, which in result leads to little to no changes to the weights of the network.

Rectified Linear Unit

The most common activation function is rectified linear unit (*ReLU*) [54], defined as

$$f(x) = \max(0, x), \tag{3.4}$$

$$\frac{d}{dx}f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}. \tag{3.5}$$

The gradient derivative reaches a value of either 0 or 1. This effectively eliminates the possibility of gradients vanishing, which is present in other activation functions, such as tanh. By definition, *ReLU* is not differentiable at 0. However, the derivative for $x = 0$ is explicitly defined in machine learning libraries.

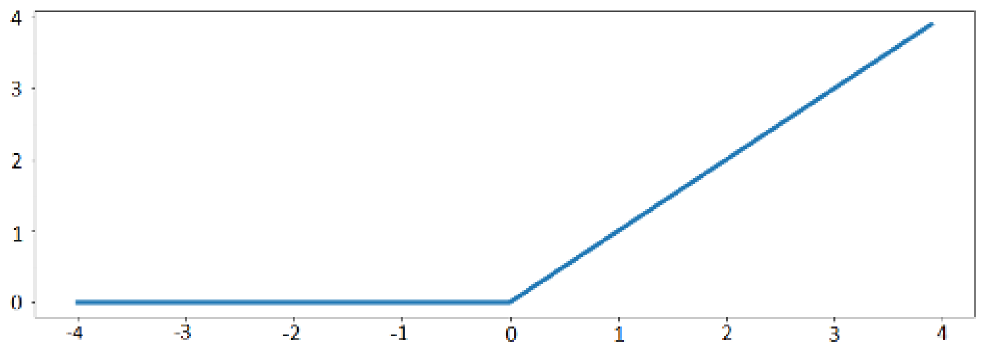


Figure 3.3: The rectified linear activation function.

Figure 3.3 shows the plot of *ReLU*. Since the activation function is very close to being linear, it preserves the properties that make linear models using gradient-based methods optimize well [20]. Additionally, *ReLu* introduces sparsity in the network, and therefore performs well when used for tasks with a large degree of data sparsity [19].

3.2 Training Process

Neural networks are usually trained by using iterative, gradient-based algorithms. The goal of those optimization algorithms is to modify the weights of the neuron connections between the layers in a way that increases the accuracy of classifying or modifying the input data. The most common gradient-based optimization algorithm, **gradient descent** [66], is used in this work.

The goal of these algorithms is to find the values of weights that minimize a **cost function**. To modify the weights of the network, we compute the gradient of the cost function and subtract the gradient from the given weight. The gradient refers to a direction in which the cost function moves. To optimize the weights, we move in the opposite direction. The weights adjustment — the optimization step — is defined as

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_n(w^{(\tau)}), \tag{3.6}$$

where $w^{(\tau)}$ is the current value of the weight, η is the *learning rate*[2], $w^{(\tau+1)}$ is the value of the weight after the optimization step and ∇E_n is the gradient of the cost function computed from a subset n of the whole training set. When the goal is to find a local maxima instead of a local minima, the process is called *gradient ascent*, defined as

$$w^{(\tau+1)} = w^{(\tau)} + \eta \nabla E_n(w^{(\tau)}). \tag{3.7}$$

Initial values of the weights, $w^{(0)}$, can be defined randomly by using, for example, uniform distribution or by using one of the initialization algorithms [18]. Iterating over all the samples of the dataset once is called an *epoch*. Usually, the optimization process iterates over the same training set multiple times in order to properly adjust the network weights.

The subset n used in a single step is called *batch*. The size of a batch used for the step affects the convergence of the optimization algorithm and robustness of the trained model [73]. For more information regarding choosing the optimal combination of the learning rate, batch size, and epoch count, the reader can refer to [29, 73].

3.2.1 Cost Functions

Depending on the target objective, there are several cost functions that can be applied to train neural networks. Objective functions used in this work include *mean squared error* (MSE) and *binary cross-entropy* (BCE).

Mean Squared Error

The mean squared error cost function can be defined as

$$MSE = E_n(x_n, y_n) = \frac{1}{N}\sum_{i=1}^{N}(y_{n,i} - \theta(x_{n,i})), \tag{3.8}$$

where N is the batch size, $y_{n,i}$ is the expected output of the network, given the input $x_{n,i}$ and $\theta(x_{n,i})$ is the actual output of the network θ, given the same input. MSE is often used in linear regression models.

[2]Learning rate, commonly referred as "step size", is one of the most important hyperparameters used in training.

Binary Cross-Entropy

For binary classification tasks, binary cross-entropy is often used as a cost function. BCE is usually coupled with a sigmoid activation function. In generative adversarial networks, BCE is used as a cost function to classify the discriminator network output. The function can be defined as

$$BCE = E_n(x_n, y_n) = -\frac{1}{N}\sum_{i=0}^{N} y_{n,i}\log\left(\theta(x_{n,i})\right) + (1-y_{n,i})\log\left(1-\theta(x_{n,i})\right), \tag{3.9}$$

where $y_{n,i}$ is the expected outcome (either 0 or 1) and $\theta(x_{n,i})$ is the predicted probability, given input $x_{n,i}$.

3.2.2 Error Backpropagation

As defined in (3.6), the weight vector is moved in the direction of the greatest decrease of the cost function. The gradient of the cost function is computed using the backpropagation algorithm.

In feedforward networks, each unit computes a weighted sum of its input, which can be written as

$$a_j = \sum_i w_{j,i} z_i, \tag{3.10}$$

where z_i is the output of the i-th neuron from the previous layer used as an input to j-th neuron of the current layer and $w_{j,i}$ is the weight associated with the connection. When evaluating the derivative of the cost function E_n with respect to $w_{j,i}$, we apply the chain rule for partial derivation, which can be formulated as

$$\frac{\partial E_n}{\partial w_{j,i}} = \delta_j \frac{\partial a_j}{\partial w_{j,i}}, \tag{3.11}$$

in which δ_j is the error, written as

$$\delta_j = \frac{\partial E_n}{\partial a_j}. \tag{3.12}$$

The backpropagation formula for k-th layer is defined as

$$\delta_j^{(k)} = \tilde{f}'(a_j^{(k)}) \sum_l w_{l,j}^{(k+1)} \delta_l^{(k+1)}, \tag{3.13}$$

where $\tilde{f}'$ is the derivative of the activation function, $w_{l,j}^{(k+1)}$ is the weight of the connection, and $\delta_l^{(k+1)}$ is the gradient derivation of the previous layer $k+1$. Overall, the training process can be split into four steps:

- **forward pass** - computing the output via propagating the input through the network.
- **computing the cost derivative** - the cost and its derivative, δ_l is computed from the network output.
- **backward pass** - using backpropagation algorithm, δ_j is computed for all network connections.
- **gradient descent** - the weight vector is updated according to the computed gradients.

3.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) [66] are a family of neural networks for processing sequential data. In RNNs, the connections between the units form a directed cycle. This is a major characteristic in which RNNs differ from standard feedforward neural networks, where the outputs of the neurons are connected to the neuron inputs of a lower level layer. This unique property allows the networks to keep an internal state while processing input sequences of arbitrary lengths. The application of recurrent neural networks includes speech recognition, speech enhancement, sentiment analysis, and machine translation.

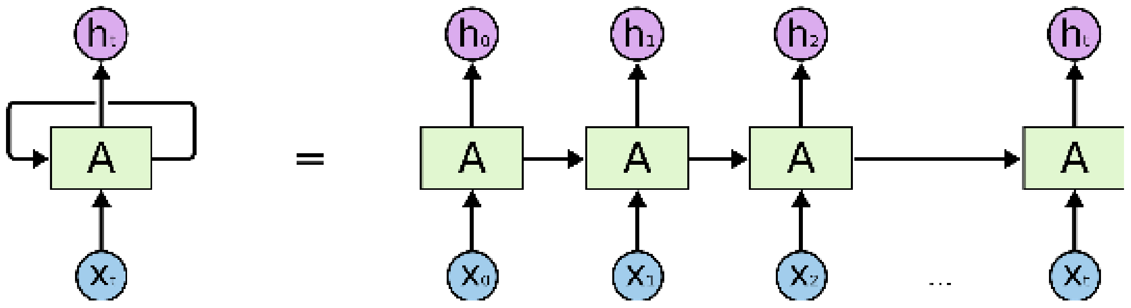


Figure 3.4: A single unrolled recurrent neural network layer. Taken from [55].

RNN layer forms a chain-like structure of repeating modules, as can be seen in Figure 3.4. The network processes a vector of inputs, $(x_0, ..., x_t)$ and produces output that that acts as an input to the next unit and as a temporal output $(h_0, ..., h_t)$ corresponding to the unit. These repeating modules consist of a single tanh activation function, as can be seen in Figure 3.5.

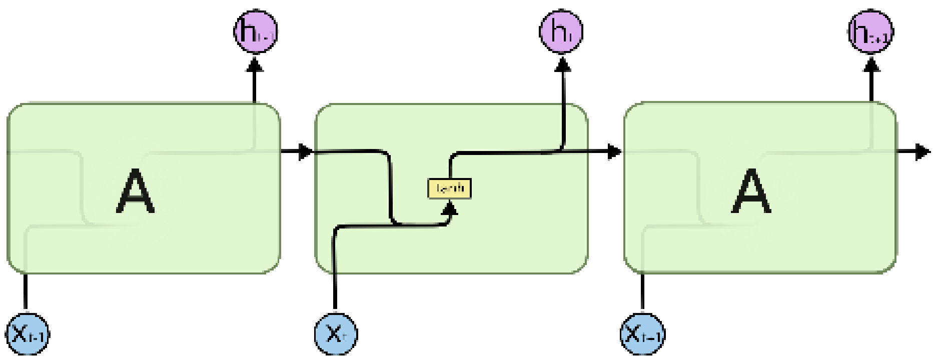


Figure 3.5: The single-layered repeating module of a standard RNN. Taken from [55].

The main issue of RNNs is that they suffer from the vanishing and exploding gradient problems [8]. Hence, the network is unable to store information about past inputs for very long [61]. Therefore, the network's predictions are based only on the last several inputs.

Long Short-Term Memory Networks

The problems occurring in standard RNNs are largely solved in long short-term memory networks (LSTMs) [28]. LSTMs consist of the same structure of repeating modules. The more advanced internal architecture of the module is designed to preserve proper error propagation during a backward pass.

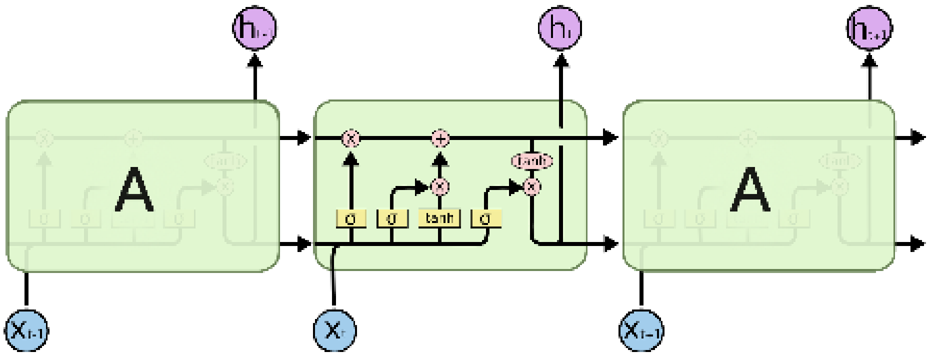


Figure 3.6: The repeating module of a LSTM network. Taken from [55].

As seen in Figure 3.6, each unit consists of four layers. The black line depicts a vector, which is transformed through network layers denoted as yellow boxes and element-wise operations displayed as pink circles. Lines forking and merging denote vector copy and concatenation, respectively.

The cell state vector, which is denoted as the upper line in Figure 3.6 is the key part of the network. The network is able to add or remove information from the vector during each time step. This action is regulated by internal components called *gates*. Each unit consists of three gates — *forget gate layer*, *input gate layer*, and *output gate layer*.

The first layer of LSTM, *forget gate layer*, decides what information gets removed from the cell state vector. The forget gate output f_t can be formulated as

$$f_t = \sigma(\mathbf{W_t}\tilde{x}_t + b_f), \tag{3.14}$$

$$\tilde{x}_t = [h_{t-1}, x_t], \tag{3.15}$$

where σ denotes the sigmoid activation function, $\mathbf{W_t}$ is the weight matrix, $\tilde{x}_t$ is the concatenated temporal output vector of the previous unit with the input vector. The old cell state is then multiplied by f_t. In this work, the LSTM networks have all forget gate biases initialized to 1 [31].

The cell state is then appended by the input gate output, i_t multiplied by new candidate values, $\tilde{C}_t$, defined as

$$i_t = \sigma(\mathbf{W_i}\tilde{x}_t + b_i), \tag{3.16}$$

$$\tilde{C}_t = \tanh(\mathbf{W_C}\tilde{x}_t + b_C), \tag{3.17}$$

where $\mathbf{W_i}$, $\mathbf{W_C}$ and b_i, b_C are weight matrices and bias vectors of input gate layer and tanh layer, respectively. The complete cell state update can be written as

$$C_t = f_t C_{t-1} + i_t \tilde{C}_i, \tag{3.18}$$

where C_{t-1} is the cell state of the previous time step. To sum it up, the previous cell state C_{t-1} is filtered via f_t and the values in $\tilde{C}_t$ chosen by i_t are appended to the filtered cell state to produce the current cell state.

Lastly, temporal output of the cell is computed. The output of the output gate layer, o_t, filters the output candidates using sigmoid activation function, which is formulated as

$$o_t = \sigma(\mathbf{W_o}\tilde{x}_t + b_o). \tag{3.19}$$

The output of the final layer is then multiplied by the cell state ran through tanh activation function to produce the temporal output h_t, defined as

$$h_t = o_t \tanh(C_t). \tag{3.20}$$

Different internal architectures of LSTM cells have been proposed, such as Gated Recurrent Unit (GRU) [11]. GRUs combine the forget gate and input gate into a single layer called *update gate* and merge the temporal state h_t and cell state C_t into one state as well. However, it has been demonstrated that there is no significant difference between the LSTM variants [22].

3.4 Convolutional Neural Networks

Convolutional neural network (CNN) [14, 39] is a class of neural networks that excels in image processing tasks. CNNs essentially capture the spatial and temporal (when applicable) dependencies in input through the application of filters. They take less time to train, as they have many fewer trainable parameters. CNNs gained popularity in 2012, when Krizhevsky et al. created *AlexNet* [37] CNN architecture, which outperformed other architectures in ImageNet 2012 Visual Recognition Challenge [67] by a large factor.

In this section, we only describe basic building blocks of a CNN, as this work only uses CNNs for a small set of experiments. For a more in-depth description, see [20].

Basic Building Blocks

Following a terminology from [20], a typical layer of a convolutional network consists of three sub-layers — **convolutional layer**, **detector layer**, and **pooling layer**.

Each **convolutional layer** generates a higher-level abstraction of the input data that preserves important information. A convolutional layer performs a convolutional operation on its input by applying a *filter* (also called a kernel), producing *feature map* as an output. The neurons between the feature map and a subsequent layer are not interconnected by weights, as was the case with feedforward networks. Instead, each feature map shares the same set of weights — the filter weights. This principle, which greatly reduces the number of learnable parameters, is called *weight sharing*. This reduction is accomplished by making the filter smaller than the input. Typically, several filters are applied to the input, producing a distinct feature map for each of the filters. Each of those feature maps is called a *channel*.

Convolution is a linear operation. To perform a non-linear transformation, an activation function is used on the convolved output in the **detector layer**. Followed by that, a *pooling function* is used in the **pooling layer**.

A pooling function splits the feature map into multiple smaller parts, calculates summary statistics for each of those parts, and produces a single output for each part. The pooling function operates on each feature map independently. This leads to an additional size reduction of the feature map. For example, the *max pooling* [83] operation splits the layer input into rectangular areas and produces the maximum of that area as the output. This makes the application approximately *invariant* to small translations of the input [21]. This means that the output of the pooling layer does not change when the input changes by a small amount. For example, this helps the network for handwritten digit recognition to recognize digits more properly, even if these digits are rotated by a certain degree.

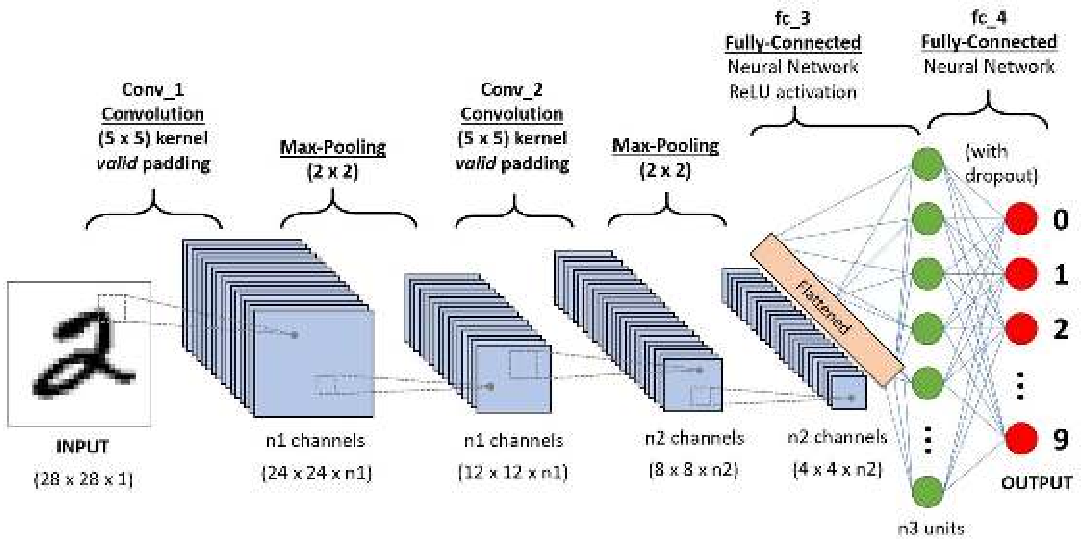


Figure 3.7: An example structure of a convolutional neural network. The network takes a grayscale image consisting of a single channel as an input, producing the estimated recognized digit as an output. Taken from [3].

Figure 3.7 shows an example of a simple convolutional network for digit recognition. The input is a single-channel two-dimensional image, that is progressively reduced in size by applying convolution and max pool operations.

3.5 Neural Networks with Cycle-Consistency

The main purpose of a neural network is to map input X to another output $\tilde{Y}$, formally written as

$$F : X \longrightarrow \tilde{Y}, \tag{3.21}$$

$$\tilde{Y} \approx Y, \tag{3.22}$$

where the network F attempts to produce an output that is similar to a reference sample Y with respect to the cost function. A concept widely used in machine translation and visual tracking, called *cycle-consistency*, can be applied to enforce additional constraints within the framework [25, 32]. For example, when translating a sentence from language A to language B, the machine should be able to transform the translated sentence back to the original sentence in language A. This form of cycle-consistency is called *forward-backward consistency*.

Cycle-consistency can be achieved by introducing an additional neural network to the framework. The network serves as an inverse mapping function

$$G : Y \longrightarrow \tilde{X}, \tag{3.23}$$

$$\tilde{X} \approx X, \tag{3.24}$$

[3]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

where G is the neural network performing a *dual task* — attempting to produce X when given Y as an input. Both networks are then used in conjunction to be consistent with each other. The *forward cycle-consistency* objective aims to accurately reconstruct X, and can be defined as

$$X \longrightarrow F(X) \longrightarrow G(F(X)) \approx X. \tag{3.25}$$

Similarly, we can define backward cycle-consistency, where the goal is to reconstruct Y, as follows:

$$Y \longrightarrow G(Y) \longrightarrow F(G(Y)) \approx Y. \tag{3.26}$$

The constraints are enforced by adding the cost functions for (3.25) and (3.26) to the main objective function, which is defined as

$$L_{CSE} = \lambda_1 L(F) + \lambda_2 L(G) + \lambda_3 L(F, G) + \lambda_4 L(G, F), \tag{3.27}$$

where $L(F)$, $L(G)$, $L(F, G)$ and $L(G, F)$ is a cost function of F, G, a forward and a backward cycle, respectively, with λs being weight coefficients. The architecture of a cycle-consistent framework used in this work can be seen in 7.1.

Enforcing forward-backward consistency plays an important role in speech enhancement [49]. When used together with generative adversarial networks, such systems can perform difficult tasks, such as unpaired image-to-image translation significantly better [84]. Recently, several speech enhancement frameworks inspired by [84] were published [49, 51]. In this work, which is based on [49], we implement a cycle-consistent neural network both with and without GAN for speech enhancement.

Chapter 4

Generative Adversarial Networks

Generative modeling is an *unsupervised learning* approach that involves discovering and learning data distribution. The model can be used to output new samples that could have been considered as if they originated from the training set. Generative adversarial network (GAN) is a class of machine learning systems based on generative models invented by Goodfellow et al. (2014) [21]. In GANs, two neural networks are pitted against each other, each attempting to reach its objective, which is 'adversary' to the other network. The discovered framework quickly gained popularity due to its ability to generate samples that are reminiscent of the training set [82, 77, 30].

This Chapter explains the unsupervised approach taken by GANs along with how adversary training works. Additionally, we describe CycleGAN architecture, followed by a showcase of some interesting publications using GANs.

4.1 Generative Modeling

As defined in Chapter 3, the goal during the training of a neural network is to produce an output that resembles a certain target output — the label. The supervised learning model is essentially a function that maps inputs X to samples $\tilde{Y} \approx Y$. The training data samples can therefore be formulated as pairs of inputs and labels, (X, Y). This approach is called *supervised learning*.

However, neural networks require a significant amount of training data and having labelled sample for each training input can be a costly task. A training method without the requirement of having labels is used in generative modeling. In that approach, called *unsupervised learning*, the model's main objective is to is learn an underlying hidden structure of some data. More specifically, generative models attempt to model a probability distribution of the given set, as depicted in Figure 4.1.

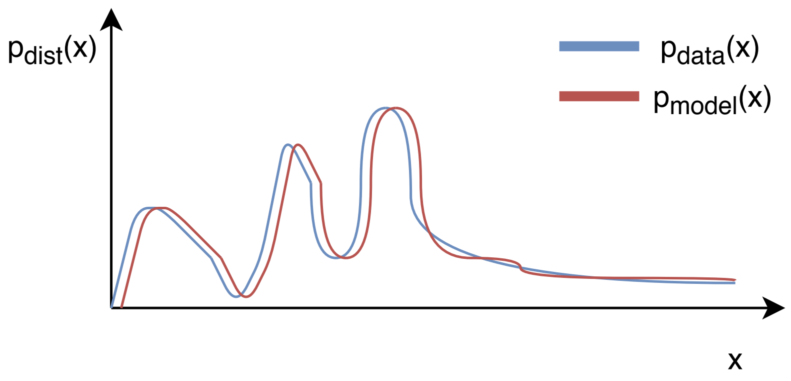


Figure 4.1: The goal of a generative model is to be able to model the probability distribution of data, p_{model}, that resembles the data from p_{data}.

Like generative models, GANs are based on *unsupervised* learning. However, GANs frame the problem as supervised — by using a cost function to predict whether the generated data belongs to the probability distribution or not. GANs are based on a scenario in which the **generator network** must compete against an adversary, the **discriminator network**.

The generator, G, produces samples X, given the generated noise (e.g., uniform noise) Z:

$$G : Z \longrightarrow X. \tag{4.1}$$

Its adversary, D, attempts to recognize whether its inputs have been drawn from the training data or not. The output of the discriminator is defined as

$$D : X \longrightarrow < 0, 1 >, \tag{4.2}$$

where p is the probability of the input sample belonging to the data distribution.

4.2 Training GANs

Generative adversarial networks are jointly trained in a minimax game, where the generator attempts to generate samples that the discriminator recognizes as samples from the training set. Contrarily, the discriminator attempts to recognize the samples from training set as real samples, while recognizing the samples generated by G as fake samples.

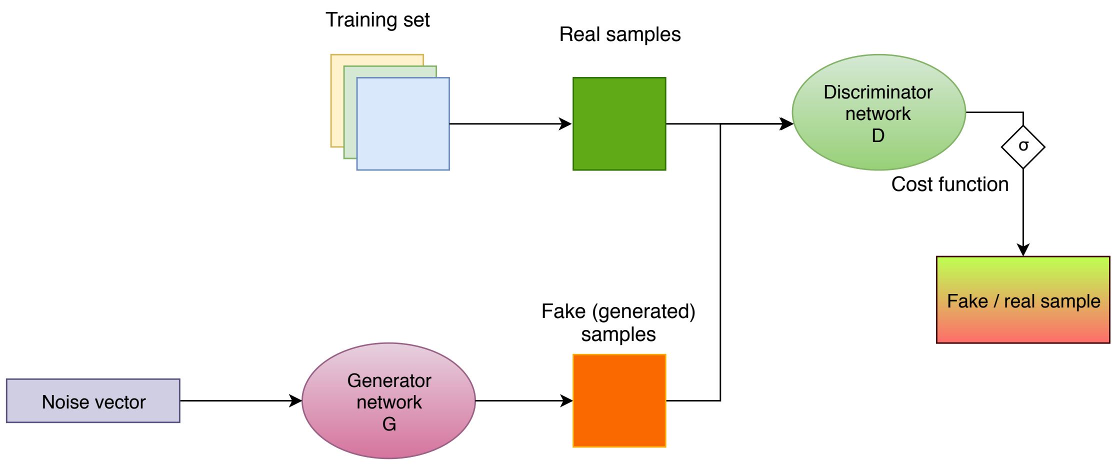


Figure 4.2: A structure of generative adversarial network.

Figure 4.2 shows the standard structure of generative adversarial network. The optimization problem is defined as

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)} \log D(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z))), \tag{4.3}$$

where x is a sample from p_{data} and $G(z)$ is a sample generated from noise distribution, $p_z(z)$. The discriminator loss is used to adjust the discriminator network weights using gradient ascent:

$$\max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)} \log D(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z))). \tag{4.4}$$

Ultimately, the best-case scenario for D is $D(x) = 1$ and $D(G(z)) = 0$. The generator loss is evaluated using gradient descent:

$$\min_G V(D,G) = \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z))), \tag{4.5}$$

where the optimal generator deceives the discriminator with fake samples, such that $D(G(z))$ $= 1$. In practice, using this generator objective does not work well. Early in learning, when the generator is poor, the discriminator can reject samples with high confidence as they are visibly different from the training set. This causes $\min_G V(D,G)$ to *saturate* [21]. Instead, the objective can be shifted to find local maxima, which results in stronger gradients during early training:

$$\max_G V(D,G) = \mathbb{E}_{z \sim p_z(z)} \log D(G(z)). \tag{4.6}$$

4.2.1 Conditional Generative Adversarial Network

The generator network in standard GAN takes a random data point from p_z as input, which makes the output rather unpredictable, as there is no way to control the generator output besides trying to figure out the complex relationship between the latent space $p_z(z)$ and the generated output. For example, when given a training set of images depicting handwritten numbers, there is no way to generate handwritten digits specifically representing the number 5.

As mentioned in In [21, 52], GANs can be extended to contain additional information. This extension is called *Conditional GAN* (CGAN). In CGAN, an additional information,

$\mathbf{y}$, is used to condition both the generator and the discriminator model on extra information. Continuing the example, the condition could therefore represent a certain class label — the number we want to generate. The objective function (4.3) is modified in CGAN as follows:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} \log D(x, \mathbf{y}) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z, \mathbf{y}))). \tag{4.7}$$

4.2.2 Gradient Reversal Layer

As described in Section 3.2, to find a local maxima of a cost function, its gradient derivative needs to be added to the weights, as opposed to being subtracted. A simple way to make the algorithm perform as if it were performing gradient ascent is by inserting *Gradient Reversal Layer* (GRL) [16] into the network. During the forward propagation, GRL acts as an identity transform. During the backward pass, the only action GRL performs is multiplying the already computed gradient by a constant $\lambda < 0$. In this work, we use GRL implementation made by Joris van Vugt[1].

4.3 CycleGAN

CycleGAN is a GAN framework that uses a cycle-consistency loss to enable training without the need for paired data. Proposed by Zhu et al. (2017) [84], the architecture was extensively evaluated on image-to-image translation problems[2], such as zebra-horse image transfer.

The goal of CycleGAN is to learn a mapping from the source domain to the target domain and vice versa. The framework consists of *four* neural networks in total — two generator-discriminator pairs. Forward and backward cycle-consistency losses are added to the cost function.

Additionally, the full cost function is extended with *identity mapping loss*. The generator networks are kept close to the identity mappings by the following constraints:

$$X \longrightarrow G(X) \approx X, \tag{4.8}$$

$$Y \longrightarrow F(Y) \approx Y. \tag{4.9}$$

The full objective function of CycleGAN is defined as

$$\begin{aligned} L_{CycleGAN} = \lambda_1 L(F, G) + \lambda_2 L(G, F) - \lambda_3 L_D(D_F) \\ -\lambda_4 L_D(D_G) + \lambda_5 L_I(F) + \lambda_6 L_I(G), \end{aligned} \tag{4.10}$$

where $L(F, G) + L(G, F)$ is a forward-backward cycle-consistency loss, $L_D(D_F)$, $L_D(D_G)$ are discriminator losses, and $L_I(F)$, $L_I(G)$ are F and G identity losses, respectively, with λs being weight coefficients. The whole CycleGAN architecture used in this work is shown in Figure 7.2.

It is important to note that CycleGAN builds on a variation of conditional GAN we described in 4.2.1, as was explained by the authors. The generator input does not use

[1] `https://github.com/jvanvugt/pytorch-domain-adaptation/blob/master/utils.py`

[2] Image-to-image translation is a task that takes images from the source domain and transforms them in such a way that they have a style of images from another domain.

generated noise z, the input is conditioned on information present in x instead[3]. Discriminator input does not contain information y [84]. This makes the input-target mapping deterministic, as there is no randomness factor present anymore.

Recently, experiments using CycleGAN for single-channel speech enhancement problem have been conducted [49]. This work, which is based on [49], uses the framework for speech enhancement with *both* paired and unpaired training data.

4.4 Application of GANs

It has been shown that the use of GANs for speech enhancement significantly boosts performance [62, 77]. Since GANs work with probability distributions, the generator network can be trained to generate completely new data that can be barely distinguishable from the original set. Notable example is a human face generator, StyleGAN[4], which can effectively synthesise various facial attributes [33] to create a unique, realistic face. Similar to vector arithmetics in language modeling [50], GAN model is able to generate images with or without certain attributes [65], as shown in Figure 4.3. Another interesting example includes upscaling images to high resolution [40]. A detailed list of relevant publications can be found at https://github.com/zhangqianhui/AdversarialNetsPapers.

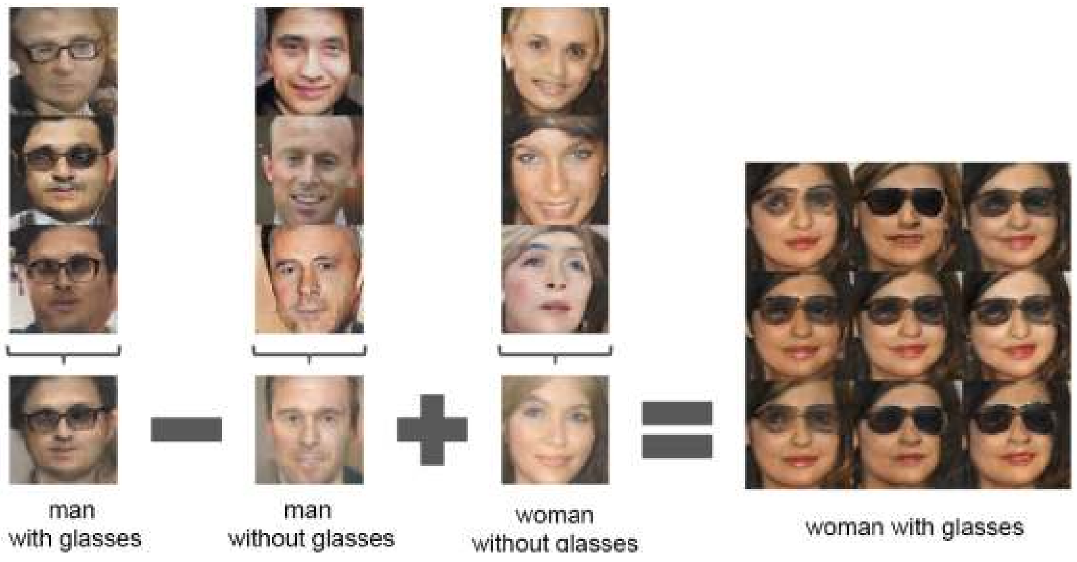


Figure 4.3: Vector arithmetics for visual concepts, published by Radford et al. [65]. For each column, the vectors of samples are averaged. The arithmetic is performed on mean vectors of those samples.

[3] https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/issues/152#issuecomment-345907486

[4] A website for StyleGAN demonstration - https://thispersondoesnotexist.com/

Chapter 5

CHiME-3 Dataset

In this Chapter, we describe the dataset used to train and evaluate our speech enhancement models. We briefly mention the challenge this dataset is associated with.

5.1 CHiME-3

Throughout recent years, several challenges in the speech recognition field called CHiME challenges are being held. Each of the challenges focuses on a certain subfield of speech recognition with a specific dataset and various other tools provided for the specific task. The most recent challenge, CHiME-6 Challenge, focuses on distant multi-microphone conversational speech diarization and recognition in home environments[1]. For the challenge, a dataset containing recordings of conversations in home environments (e.g., during dinner) is used. These recordings contain a heavy amount of reverberation and several people speaking at once.

The third CHiME challenge, **CHiME-3 Challenge**[2], focuses on the performance of ASR in real-world scenarios. A portion of the dataset used in the challenge, CHiME-3 dataset [5], is used in this work, as well. The following section describes the dataset contents in great detail.

5.2 Properties of the Dataset

The dataset scenario is ASR for a multi-microphone tablet device used in everyday environments. The dataset consists of four distinct environments:

- **BUS** - recorded in public transports, contains heavier amount of background noise.
- **CAF** - recorded in café environments, has a larger presence of other speakers in the background.
- **PED** - recorded in pedestrian areas.
- **STR** - recorded in street junctions.

For each environment, two types of noisy data have been provided — *real* and *simulated* data. Additionally, the dataset contains **BTH** data — utterances recorded in a recording

[1]CHiME-6 Challenge - https://chimechallenge.github.io/chime6/
[2]CHiME-3 Challenge - http://spandh.dcs.shef.ac.uk/chime_challenge/chime2015/

booth. These utterances were use to create a portion of simulated data. The data is divided into *development* (dev), *training* (train) and *evaluation* (eval) sets. The utterances are provided as 16 kHz, 16 bit stereo WAV files.

The file naming format of the noisy speech recordings is $S\_T\_E.C.wav$, where S, T, E, and C denote speaker, transcription, environment, and channel, respectively. Channel indexes 1-6 specify the tablet microphones (see Figure 5.1), while channel index 0 denotes the close-talking microphone. Clean speech recordings are formatted as $S\_T\_ORG.wav$, where ORG depicts that the speech data is based on the original WSJ0 training data [17].

5.2.1 Real Data Recordings

The real data consists of 6-channel recordings of the WSJ0 corpus sentences [17] spoken in the environments.

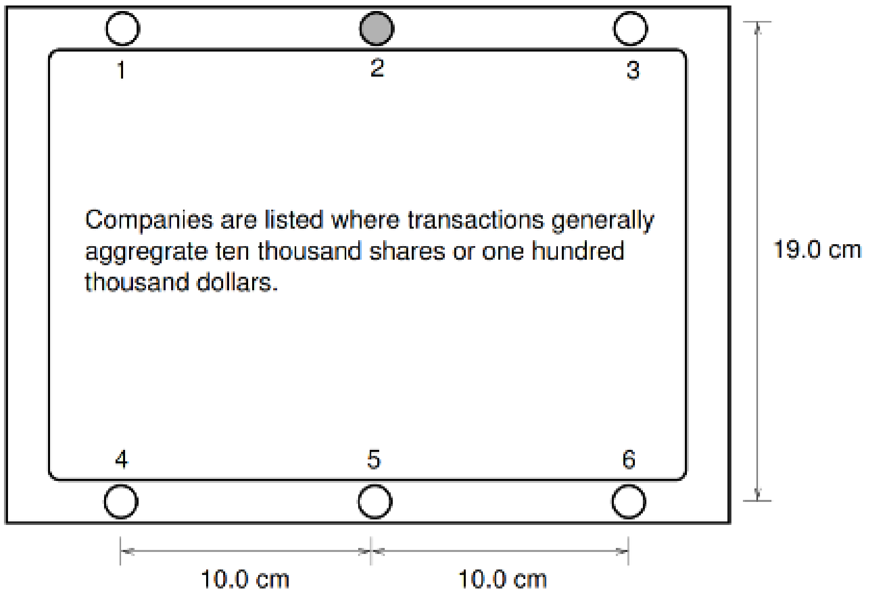

1
2
3
Companies are listed where transactions generally aggregrate ten thousand shares or one hundred thousand dollars.
19.0 cm
4
5
6
10.0 cm
10.0 cm

Figure 5.1: The microphone array geometry of the recording tablet. The spoken utterances are read from the tablet, which is placed 40 cm away from the speaker. All microphones face forward the speaker, with the exception of microphone 2. Taken from [5].

Figure 5.1 shows the tablet device used for recording. In addition, a close-talking microphone of a headset was used to capture speech.

The recordings have been made by 6 male and 6 female speakers. About 100 sentences were recorded in each of the four environment locations. In addition, these sentences were recorded in the acoustically isolated booth (BTH) environment. Those sentences were then used to produce simulated data. The *training set* data consists of 1600 real noisy utterances: four speakers, each reading 100 utterances in each of the four environments. The sentences were randomly selected from the WSJ0 5k training data. The *development set* and *evaluation set* data consist of 410 and 330 utterances, respectively, spoken by four different speakers. Each utterance was spoken in four environments. The sentences spoken for development and evaluation set data were the same as those in WSJ0 5k development and evaluation set, respectively.

5.2.2 Simulated Data

The simulated data in the CHiME-3 dataset was constructed by mixing clean speech recordings with noise representing the aforementioned environments. Impulse response (IR) for the tablet microphones was estimated. These IRs were used to estimated signal-to-noise ratio (SNR) at each tablet microphone. For development and evaluation set data, IRs were used to estimate the noise signal by subtracting the convolved signal recorded by the close-talking microphone. The information representing the spatial position of the speaker with respect to the recording microphone was convolved with a clean speech signal before being mixed with a noise signal.

The training set simulated data consists of 7138 original recordings from WSJ0 mixed with separately recorded noise background. The clean utterances were spoken by 83 speakers. In the case of the development set and the evaluation set data, the clean speech signal was taken from the booth recordings. These recordings were then mixed with the original real noisy recordings from which the speech was taken out.

5.2.3 Data Used For Experiments

This work is based on [49], which used two subsets of CHiME3 — training set for training the NN models and development set for ASR evaluation.

We extend the training set with additional data, as was done by [49]. In addition to 7138 pairs, we train the models with 1600 real recordings and 399 BTH recordings, as well, totaling **9137** pairs. From these additional recordings, the 0th channel, which is recorded by the close-talking microphone, is used as a target. Utterances recorded by the 5th microphone are used as input. The total length of data used for training is 18 hours 48 minutes. For evaluation, we also use 5th channel real data recordings from the development set. The summary of the CHiME-3 dataset, highlighting the data used in this work, is shown in Table 5.1.

Subset	Real/Simulated	Environments	Utterances
dev	Simulated	{BUS, PED, CAF, STR,}	{410, 410, 410, 410}
dev	**Real**	**{BUS, PED, CAF, STR}**	**{410, 410, 410, 410}**
dev_bth	Real	{BTH}	{410}
eval	Simulated	{BUS, PED, CAF, STR}	{330, 330, 330, 330}
eval	Real	{BUS, PED, CAF, STR}	{330, 330, 330, 330}
train	**Simulated**	**{BUS, PED, CAF, STR}**	**{1728, 1765, 1794, 1851}**
train	**Real**	**{BUS, PED, CAF, STR}**	**{400, 400, 400, 400}**
train_bth	**Real**	**{BTH}**	**{399}**

Table 5.1: The full composition of CHIME-3 dataset. The subsets used in this work were made bold.

Chapter 6

Implementation

This Chapter contains a basic overview of the technologies used in this work, the implemented neural networks, and the evaluation tools.

6.1 PyTorch

The speech enhancement architectures in this work were implemented and trained using **PyTorch** [63]. PyTorch is an open-source machine learning Python library. The framework is primarily developed by Facebook's AI Research lab (FAIR)[1]. The strength of PyTorch lies within its optimization, interoperability, and extensibility. Two versions of PyTorch are available, a CPU version and a GPU version.

6.1.1 Deep Learning Model as Program

Neural networks evolved rapidly from simple sequences of feedforward layers into incredibly varied and large architectures. PyTorch has taken an imperative approach to preserve the imperative programming model of Python. Components used for deep learning models, such as layers and data-loading scripts can all be expressed using concepts familiar to standard developers.

This approach ensures that any neural network architecture can be implemented and trained with ease. For example, layers are represented as Python classes, in which constructors create and initialize layer parameters. These classes contain methods to process input activation. Models are also expressed as classes that consist of several layers. This way, the user can easily modify the execution process of the whole architecture. Optimizers, data-loading scripts, and datasets are implemented as classes, as well. The user can extend the specific class and enhance its behavior as suited.

The base PyTorch class is **Tensor**. Tensor is a homogeneous multi-dimensional matrix on which various operations can be performed. It is similar to NumPy's *ndarray*[2]. Unlike NumPy's ndarrays, Tensors can be used on a GPU to accelerate computing. Tensors track operations performed on it. The tracking history is then used to compute gradients as described in 6.1.3.

[1] FAIR - https://ai.facebook.com/

[2] *numpy.ndarray* - https://numpy.org/doc/1.18/reference/generated/numpy.ndarray.html

6.1.2 Performance

In Python, *global interpreter lock* (GIL)[3] ensures that no more than one thread out of any concurrent threads can be run at once. This drastically slows operations, which could take advantage of multithreading, for example, backpropagation. Some deep learning frameworks, such as **TensorFlow** [1] avoid this problem by using *data-flow graphs*, which defer the evaluation of the computation to a custom interpreter. Most of PyTorch is written in C++ and CUDA[4] to achieve high performance. This ensures that the important computationally expensive operations can take advantage of multithreading and are not bottlenecked by GIL.

6.1.3 Automatic Differentiation

To compute gradients, PyTorch uses a method called **automatic differentiation** (autodiff) [7]. Autodiff is a set of techniques used to numerically evaluate the derivative of a function specified by a computer program. All numerical computations are compositions of a set of elementary operations that have known derivatives. By applying the chain rule repeatedly on these operations, derivatives of arbitrary order can be computed automatically and effectively. Autodiff is numerically stable. PyTorch's autodiff package, Autograd, is used for backpropagation.

6.2 Implemented Scripts and Modules

For training a large number of different architectures, we implemented a neural network training pipeline, which makes changing NN models or switching certain components easier. The scripts were implemented in Python 3.7 using the PyTorch framework. For testing purposes, ASR training and acoustic model re-training tools were provided.

6.2.1 Python Modules

Below, some of the important Python modules used in this work are described.

- **dataset** - classes, which enable work with paired and unpaired data. These classes can be used for both training NNs and enhancing data. It is possible to choose a subset of the original dataset for training, as well.
- **modules** - this file contains all of the implemented neural network-related classes, including custom loss functions that allow working with batches of variable length.
- **tcnn** - contains the implementation of experimental Temporal Convolutional Neural Network (TCNN) [60] module.
- **train** - used for handling the evaluation and optimization process of NNs.
- **tools** - methods for processing speech data (e.g., feature extraction).
- **enhance_data** - script used for dataset enhancement.

[3]Global interpreter lock - https://wiki.python.org/moin/GlobalInterpreterLock
[4]CUDA - https://developer.nvidia.com/cuda-zone

6.2.2 ASR Tools

For this work, two tools were provided:

- **acoustic model re-training script** - a script for re-training ASR's acoustic model.
- **ASR evaluation scripts** - scripts for ASR evaluation using Kaldi speech recognition toolkit [64] with base and re-trained acoustic model.

6.2.3 TensorBoard

For monitoring the results, we used TensorFlow's visualization toolkit, TensorBoard [5]. TensorBoard enables tracking experiment metrics like loss and accuracy, histogram visualization of weights, et cetera.

These tracked values are stored in a hierarchical folder structure. These data are interpreted using a TensorBoard server. The tracked data can then viewed in a browser, as shown in Figure 6.1. The graph and histogram contents can then be filtered to only show portions of tracked data.

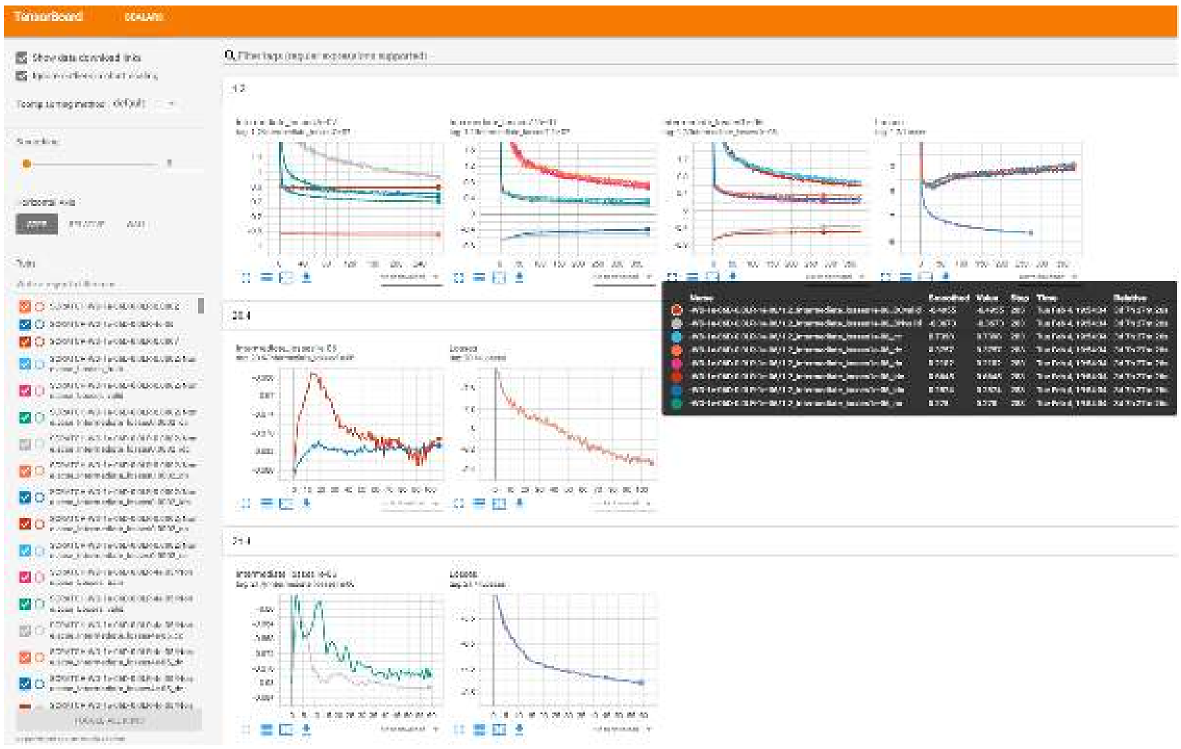


Figure 6.1: TensorBoard's browser visualization tool.

[5]TensorBoard - https://www.tensorflow.org/tensorboard

Chapter 7

Experiments

For training and evaluation, we use the CHiME-3 dataset [5], which incorporates Wall Street Journal (WSJ) corpus sentences spoken in challenging noisy environments, as described in Chapter 5.

We evaluate three major architectures — noisy-to-clean mapping network, which will serve as a baseline, a network with cycle-consistency, and a generative adversarial network with cycle-consistency. These models were proposed in [49]. We further experiment with modifying the paired and unpaired cycle-consistent architectures in order to find the best-possible configuration, without changing the layer structure of the networks themselves. In addition, we experiment with a CNN-based architecture, which will be described on its own in Section 7.6. Lastly, we re-train the acoustic model of ASR with data enhanced using trained models. For testing, only the noisy-to-clean mapping network portion of the models is used to produce enhanced speech utterances.

Training and evaluation were done using Sun Grid Engine[1] job scheduling system. The evaluation is performed by using a provided ASR system. ASR scripts and acoustic model re-training scripts were provided for this work, as well. Training framework and the implemented neural network architectures are the author's own work.

Word Error Rate

In this work, we use **word error rate** (WER) metric to evaluate the performance of the models. Kaldi ASR toolkit [64] is used to evaluate WERs for the given test set. The data were fed into a trained network before being passed through Kaldi ASR. For testing, only the real data portion was used.dwar

The evaluation of WER is done by aligning the recognized word sequence with reference word sequence. The formula is defined as follows:

$$WER = \frac{S + D + I}{N}, \tag{7.1}$$

where S is the number of incorrectly detected words - *substitutions*, D is the number of words which were not detected by ASR - *deletions*, and I is the number of redundant words which were incorrectly detected by ASR - *insertions*. N is the total number of words in the reference text, equally defined as

$$N = S + D + C, \tag{7.2}$$

[1]Sun Grid Engine - http://www.fit.vutbr.cz/CVT/cluster/SGE-UsersGuide.pdf

where C is the number of correctly detected words.

7.1 Description of the Core Architectures

In this section, we describe the core architectures used in the experiments. These modules are trained using datasets that contain pairs of clean spoken utterances and the same spoken utterances with incorporated noise and reverberation — *paired* data. For adversarial training, we use the dataset as if it consisted of *unpaired* data — samples, which have no clean-noisy pair.

7.1.1 Baseline

This subsection describes models trained with *paired* data. We describe the training process of the baseline model. This trained model is used to further train paired cycle-consistent architecture (CSE).

Using standard supervised training, we first train a neural network for suppressing noise, F. The network input consists of log Mel-filterbank (MFB) features appended with first and second-order delta features, forming an 87-dimensional vector. The output is a 29-dimensional MFB without delta features. The network consists of two Long-Short Term Memory [28] layers followed by a linear layer. Each LSTM layer has 512 units. The input features were globally normal mean and variance normalized before being fed into the network.

We heavily tuned network parameters in order to achieve satisfying results. The following specification describes the best-performing baseline model. In LSTM layers, forget gate biases are initialized to 1 (otherwise 0) [31]. The weights were initialized using Xavier normal distribution [18]. For optimization, we use AdamW algorithm [46]. The learning rate is set at $9 \cdot 10^{-4}$ and batch size is set at 48. The weight decay of AdamW is set at $1 \cdot 10^{-4}$. We find that using recurrent dropout in the first LSTM layer slightly lowers the model performance. We use Mean Squared Error (MSE) as a cost function. We perform *gradient clipping* [61] with the cut off threshold set at 1. Gradient clipping ensures that the norm of the gradient is rescaled to a fixed size specified by a cut off point to prevent the gradients from exploding. Gradient clipping is not only used for training the baseline, but for all subsequent models, as well.

This baseline setup slightly deviates from [49], in which the network was optimized by using stochastic gradient descent (SGD) optimizer. No weight initialization techniques nor any other training parameters were mentioned in the reference paper. We use standard LSTM layer, while [49] uses LSTM with projection layers (LSTMP) [69][2].

7.1.2 CSE

We train a neural network, G, that inserts noise into clean speech utterance. The input and the output feature dimensions are 29 and 87, respectively. The learning rate is set at $8 \cdot 10^{-4}$. Other parameters and a cost function are the same as specified in 7.1.1.

Then, we use the pre-trained networks, F and G, and jointly train them using cycle-consistency loss. We train the model with forward cycle-consistency and a model with both forward and backward cycle-consistency. When computing cycle-consistency loss, the input

[2]In the original article, LSTMP is shown to perform similarly to standard LSTM. The main advantage of LSTMPs is that they dramatically reduce the number of trainable parameters.

of one network is normalized before being fed to the other network. We set the learning rate at $4 \cdot 10^{-4}$. The batch size is set at 24. The λ loss function coefficients are the same as in [49].

The training process of the best model converges in 7 epochs. We assume that, by pre-training F and G with carefully tuned hyperparameters, the networks adjust the weights to a relatively proper state rather quickly.

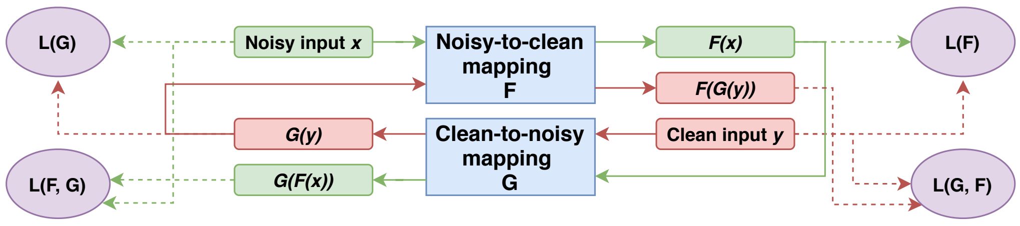


Figure 7.1: The architecture of cycle-consistency training framework for speech enhancement (CSE). Green and red lines depict forward and backward reconstruction process, respectively. Based on [49].

7.1.3 ACSE

We use the same training set that contains noisy-clean sample pairs. However, the dataset is used as if it contained no related pairs. In practice, we take a batch of random noisy samples, a different batch of random clean samples, and work with these during the training iteration.

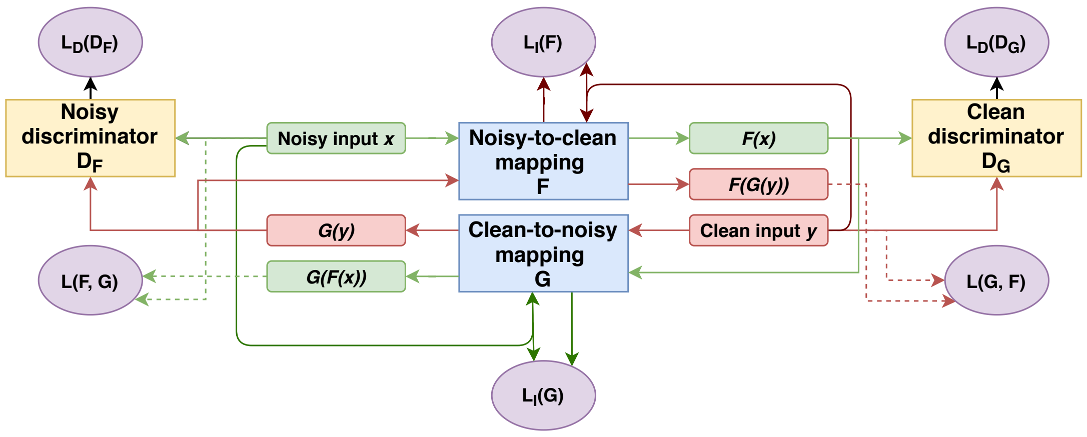


Figure 7.2: The architecture of adversarial training framework with cycle-consistency for speech enhancement (ACSE). Green and red lines depict forward and backward reconstruction process, respectively. Based on [49].

For generator networks, we use the same architecture as F and G. The discriminator networks consist of two fully-connected hidden layers. Each hidden layer has 512 units. The output layer has 1 unit. The discriminators, D_F and D_G, take 87-dimensional inputs (appended with delta features) and 29-dimensional inputs, respectively. D_F and D_G evaluate the probability of the input belonging to the noisy and clean set, respectively. We use AdamW optimizer for both generator and discriminator training.

Generally, GANs are difficult to train, as they can be very sensitive to changing hyperparameters. A large amount of minor training process adjustments was proposed [70, 72] that can significantly improve convergence and prevent common pitfalls, such as mode collapse [70].

From our experiments, these techniques were important to make the adversarial training converge:

- **initialization of generators** - The initialization is done by pre-training the generators as identity mapping functions — the target sample is the same as the input sample, but without normalization. The training hyperparameters for noisy-to-clean and clean-to-noisy generator networks are the same as of F and G, respectively. The initialization procedure in [49] may differ, as the initialization details were not mentioned.

- **buffer of generated samples** - As suggested by Shrivastava et al. [72], we update the discriminators by using a history of generated utterances rather than the ones produced by the latest generators. We store two sample buffers of size 72 that keep previously generated noisy and clean samples[3]. The original Cycle-GAN uses a history of 50 samples [84]. It is not specified whether the framework in [49] uses such technique.

- **one-sided label smoothing** - We modify the cross-entropy cost functions of the discriminators by employing one-sided label smoothing. We change the positive ground-truth labels in the cost functions from 1 to 0.9. Label smoothing is a regularization technique that prevents the discriminators from predicting the labels too confidently during training, which can result in poor generalization [74].

Before beginning adversarial training, the generator networks need to be initialized in order to learn an underlying structure. Otherwise, the model would have trouble converging. Using pre-trained generators, we perform adversarial training. The learning rate and weight decay are set at $1 \cdot 10^{-6}$. During each iteration, the discriminator networks are trained before the generator networks. The λ loss function coefficients are the same as in [49].

7.2 Results

7.2.1 Baseline

Due to carefully optimizing training hyperparameters and using proper weight and bias initialization methods, the baseline model alone reduces the ASR word error rate (WER) by 17.40% as opposed to no enhancement.

[3]Our implementation of the buffer is built on top of CycleGAN authors' buffer (`https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/blob/master/util/image_pool.py`) to allow storing additional information for each sample.

Model	Noise Environment					
	BUS	PED	CAF	STR	Avg.	RWERR
None	41.27	17.48	27.09	24.97	27.70	-
Baseline	28.91	16.36	27.58	18.68	22.88	**17.40**
None ([49])	36.25	31.78	22.76	27.78	29.44	-
Baseline ([49])	31.35	28.64	19.80	23.61	25.82	12.33

Table 7.1: The ASR WER (%) performance of real noisy test data in CHiME-3 enhanced by baseline model, including comparison with the reference paper. Relative WER reductions (%) are shown in the last column. BUS, PED, CAF, STR refer to 4 different recording environments.

As can be seen in Table 7.1, our baseline results are significantly better than the results from [49]. The main reasons are the following:

- **hyperparameter tuning** - we spent a lot of time tuning hyperparameters. It is possible that the baseline in [49] was loosely trained for the sole purpose of having a reference result, with which the cycle-consistent models could be compared.
- **AdamW optimization algorithm** - we use AdamW [46] for optimization, while [49] use stochastic gradient descent (SGD). AdamW modifies how weight decay is computed. The original Adam [35] has shown to generalize worse than stochastic gradient descent (SGD) with momentum [79]. We reached 0.32 % lower WER using AdamW when compared to Adam.
- **long enough training** - we wanted to make sure that when further training the baseline using cycle-consistency, the model does not simply improve because it has been trained for more epochs. We gradually monitored the model's ASR performance during the training process. When the ASR WER performance started degrading, signaling overfitting, the network training was stopped.
- **different ASR systems** - while we use Kaldi ASR, the system used in [49] is unspecified. Minor differences in WER for unenhanced data between our ASR and ASR in [49] can be seen in "None" rows. However, we believe that using relative WER improvement over unenhanced data is a fair comparison method.

While the baseline performs better on average, the results on enhanced real CAF recordings were worse than the original noisy data. This phenomenon was not found in [49], where the performance of enhanced CAF data relatively improves by 13 % over unenhanced data. We explored the possibility of further training the baseline with only a subset of training set. First, we experimented with training the models from scratch using only the training data from a single environment. The hyperparameters are the same as those described in 7.1.1. The results are shown in Table 7.2.

Model	Noise Environment				
	BUS	PED	CAF	STR	AVG
BUS	30.76	20.09	32.30	21.46	26.16
PED	38.46	19.88	33.70	25.94	29.50
CAF	37.51	20.06	31.12	24.91	28.40
STR	**30.45**	**18.11**	**30.09**	**20.00**	**24.67**

Table 7.2: The ASR WER (%) performance of real noisy test data in CHiME-3 enhanced by models trained in single environment.

Surprisingly, the models trained on a certain environment subset do not perform best on that specific environment. Model trained on street junction (STR) environment data has shown to perform better than other models.

Now we try to continue training the baseline model with a portion of the original training set. Because the baseline model performs worst on the CAF data, and the STR environment has the biggest impact on model quality, we further train the baseline with only CAF and STR subsets. We use a learning rate of $6 \cdot -10^{-4}$.

Model	Noise Environment					
	BUS	PED	CAF	STR	Avg.	RWERR
None	41.27	17.48	**27.09**	24.97	27.70	
Baseline	**28.91**	**16.36**	27.58	**18.68**	**22.88**	**17.40**
Baseline + (CAF + STR)	30.05	16.57	27.99	19.48	23.52	15.09

Table 7.3: Comparison of ASR WER (%) performances of baseline model and that same model further trained on CAF + STR subset.

The model trained on a subset did not further improve neither RWERR, nor reduced WER of any of the subsets, as shown in Table 7.3. The simplest possible explanation is that there are certain differences between the real data from test set and the data from training set, such as speakers or background noise levels.

7.2.2 CSE

The results in Table 7.4 confirm that training the network using cycle-consistency constraints improves model performance. Having both forward and backward cycle constraints has shown to further improve the model's robustness. The model with only the forward cycle-consistency constraint (CSE-FW) slightly boosts the model performance, up to 18.23% relative WER reduction (RWERR). The model with both cycle-consistency constraints reaches a relative WER reduction of 20.97%.

Model	Noise Environment					
	BUS	PED	CAF	STR	Avg.	RWERR
None	41.27	17.48	27.09	24.97	27.70	-
Baseline	28.91	16.36	27.58	18.68	22.88	17.40
CSE-FW	29.41	15.68	26.68	18.82	22.65	18.23
CSE	**28.35**	**15.40**	**25.24**	**18.57**	**21.89**	**20.97**

Table 7.4: The ASR WER (%) performance of real noisy test data in CHiME-3 enhanced by different models. Relative WER reductions (%) are shown in the last column. BUS, PED, CAF, STR refer to 4 different recording environments.

As with the baseline, we spent a lot of time tinkering with hyperparameters to train CSE and reached better results over CSE from [49]. Table 7.5 shows RWERR comparison of our models and the models in [49] over noisy data.

Model origin	Architecture		
	Baseline	CSE-FW	CSE
Our work	**17.40**	**18.23**	**20.97**
Publication	12.33	14.30	19.60

Table 7.5: Comparison of relative WER improvement (%) of models over noisy data.

In both cases, CSE-FW only marginally improves the model performance, when compared to CSE. It is possible that weaker baseline results in [49] led to steeper performance improvement of CSE. In our case, CSE-FW and CSE were trained only for 13 and 7 epochs, respectively, before WER started to decline.

7.2.3 ACSE

As mentioned in 7.1.3, we first initialized noisy-to-clean network F and clean-to-noisy network G before beginning adversarial training. The networks essentially learned to take a normalized speech sample and denormalize that sample. Then, those pre-trained generator networks were used for adversarial training. Without initializing the generators, ACSE had great difficulty with converging — F was not able to learn to produce meaningful samples. The network without pre-trained generators was not able to reach $L_{CycleGAN}$ loss of initial pre-trained ACSE before diverging, as shown in Figure 7.3.

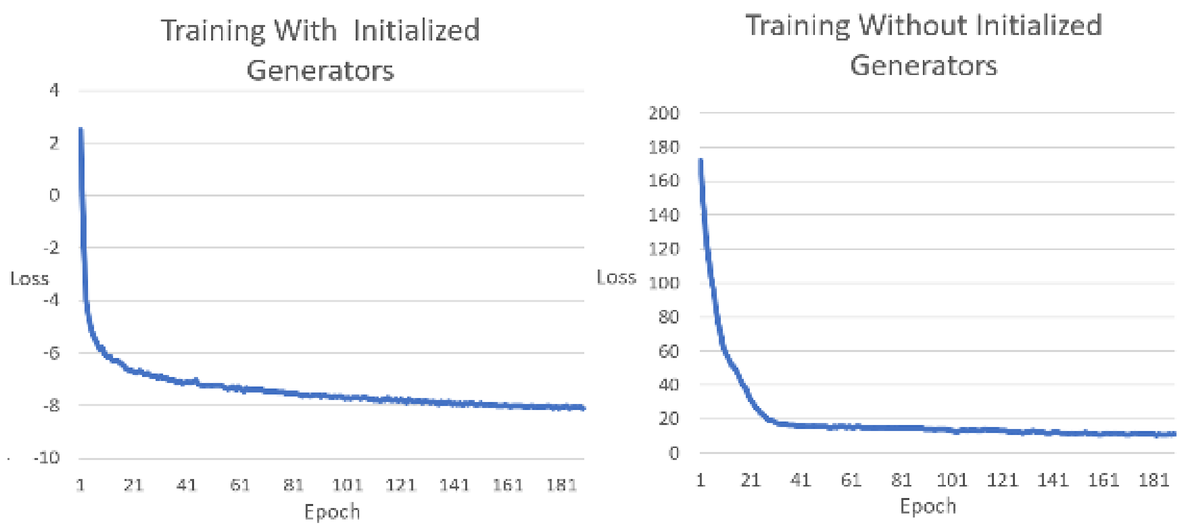


Figure 7.3: Comparison of $L_{CycleGAN}$ change of ACSE with (left) and without (right) pre-trained generators over training period.

We used several training techniques to make training ACSE work properly. In addition, we experimented with the idea of *pre-training discriminators* D_F and D_G. Training the discriminators to learn the difference between real and generated samples before beginning adversarial training could improve model convergence and stability. Discriminators were initialized by freezing the generator weights for two epochs. After that, the adversarial training has resumed.

Model	Noise Environment					
	BUS	PED	CAF	STR	Avg.	RWERR
None	41-27	17.48	27.09	24.97	27.70	-
Initialized F	34.59	18.78	28.11	23.49	26.24	5.2
ACSE	32.91	**16.27**	**26.39**	**21.16**	**24.18**	**12.71**
ACSE (pre-trained D_F, D_G)	**32.84**	16.36	26.56	21.57	24.33	12.16

Table 7.6: The ASR WER (%) performance of real noisy test data in CHiME-3 enhanced by different models. These models were trained using unpaired data.

Table 7.6 shows the performance of ACSE. The initialized generator F trained to produce denormalized noisy data relatively improves WER by 5.2 %. Using unpaired data, our variation of CycleGAN, named ACSE, has reached 24.18 % WER, which is 12.71 % improvement over noisy data. That result is only slightly worse than the baseline from Table 7.1, which was trained with paired data. ACSE performs better on the CAF subset than the baseline, which scored 27.58 %. Pre-training the discriminator networks has shown to not improve the model performance.

Model	Noise Environment					
	BUS	PED	CAF	STR	Avg.	RWERR
None	41.27	17.48	27.09	24.97	27.70	-
ACSE	32.91	16.27	26.39	21.16	24.18	**12.71**
None ([49])	36.25	22.76	31.78	27.18	29.44	-
ACSE ([49])	33.93	20.72	29.87	25.53	27.47	6.69

Table 7.7: Comparison of ASR WER (%) performance of ACSE models.

Table 7.7 compares the performance of our ACSE and ACSE from [49]. Our Cycle-GAN variant, ACSE, achieved 12.71% RWERR, which is a significant improvement when compared to [49]'s 6.69 % RWERR. This proves that the preparations mentioned in 7.1.3 have clearly helped with model's performance.

Uncooperative ACSE

Harley et al. (2019) [24] argue that there is no "fidelity loss" in forward translation and subsequent backward translation that reconstructs the original input. In other words, the error generated during forward translation might be overshadowed, as long as the backward translator properly reconstructs the input. This allows two functions to optimize for each other's outputs.

We implement uncooperative ACSE by freezing the training of F and letting G train in one step, and freezing G while training F in other step, repeatedly. The setup is the same as for ACSE without pre-training discriminators).

Model	Noise Environment					
	BUS	PED	CAF	STR	Avg.	RWERR
None	41-27	17.48	27.09	24.97	27.70	-
Uncooperative ACSE	33.15	17.64	27.08	22.30	25.04	9.6
ACSE	**32.91**	**16.27**	**26.39**	**21.16**	**24.18**	**12.71**

Table 7.8: The ASR WER (%) performance comparison of ACSE and uncooperative ACSE.

The results in Table 7.8 show that in our case, the uncooperative training did not improve performance.

7.2.4 Conclusions

The table below depicts relative WER improvements over noisy data. The WERs of noisy data in our work and the publication are 27.70% and 29.44%, respectively.

Model origin	Architecture		
	Baseline	CSE	ACSE
Our work	**17.40**	**20.97**	**12.71**
Publication	12.33	19.60	6.9

Table 7.9: Comparison of relative WER improvement (%) of models over noisy data.

Table 7.9 shows that by carefully picking hyperparameters and using various NN training enhancements, our models have performed significantly better compared to [49]. While the models in [49] were evaluated using a different ASR system, the relative WER improvement is shown over WER of noisy data from the publication, which was 29.44%, whereas the noisy data WER in our work was 27.70%.

7.3 Re-Training the Acoustic Model

The authors of [49] performed acoustic model re-training using enhanced data in order to improve ASR performance. The acoustic model portion of the ASR is trained by taking speech recordings and their text transcriptions, from which a statistical representation of the sounds that make up each word is created. The ASR system provided for this work, trained using Kaldi[4] has DNN-HMM acoustic model, which we re-trained on speech utterances enhanced by our models. The dataset used for re-training the acoustic model is the same set that was used for training the neural networks.

In the publication, the re-training was performed on data enhanced from the adversarial model (ACSE), but not others. We have re-trained the acoustic model not only on ACSE but also on baseline and CSE, as well.

Acoustic Model	Architecture			
	Baseline	CSE	ACSE	ACSE ([49])
Clean	22.88	21.89	24.18	27.47
Re-trained	19.16	18.42	**14.72**	18.20

Table 7.10: Comparison of ASR WER (%) performances of speech enhancement models evaluated with clean and re-trained ASR acoustic model.

As seen in Table 7.10, acoustic model re-training significantly boosts the performance, causing total relative WER reduction up to 46.86% for ACSE. Surprisingly, acoustic model re-trained using ACSE-enhanced speech samples shows biggest performance improvement. Similar improvements can be seen in ACSE ([49]). CSE and baseline only slightly improve the ASR performance. The possible reason for this is that while ACSE performs worse on a test set, it can generalize to unseen data better, and thus be more effective for re-training the acoustic model.

[4]Kaldi ASR - https://kaldi-asr.org/

7.4 Adversarial Training Using Paired Data

We extend the experiments to explore the possible impact of adversarial training. The presence of discriminator could further improve CSE, which was trained using *paired* data.

The CSE model with adversarial training using paired data (paired ACSE) objective function extends the original CycleGAN loss (4.7) as follows:

$$\begin{aligned} L_{ACSE_p} = {} & \lambda_1 L(F,G) + \lambda_2 L(G,F) - \lambda_3 L_D(D_F) - \lambda_4 L_D(D_G) \\ & + \lambda_5 L_I(F) + \lambda_6 L_I(G) + \lambda_{\mathbf{7}} \mathbf{L}(\mathbf{F}) + \lambda_{\mathbf{8}} \mathbf{L}(\mathbf{G}), \end{aligned} \tag{7.3}$$

where $\mathbf{L}(\mathbf{F})$ and $\mathbf{L}(\mathbf{G})$ is the cost function of F and G, respectively. The respective weights, $\lambda_{\mathbf{6}}$ and $\lambda_{\mathbf{7}}$, are the same as those in (3.27). We used the same hyperparameters as those used for ACSE, which are described in 7.1.3. Paired ACSE was trained on the same pre-trained model as CSE.

Model	Acoustic Model	AVG	RWERR
None	Clean	27.70	-
CSE	Clean	21.89	20.97
Paired ACSE	Clean	21.69	21.70
ACSE	Clean	24.18	12.71
CSE	Re-trained	18.42	33.50
Paired ACSE	Re-trained	17.72	36.10
ACSE	Re-trained	**14.72**	**46.86**

Table 7.11: Comparison of ASR WER (%) performances of CSE, paired ACSE and ACSE models.

Table 7.11 shows that paired ACSE slightly improves the model performance. Using a clean acoustic model, the RWERR improved by 0.73 %. Re-training the AM with enhanced data, the CSE extension improved RWERR by an additional 2.60 %. However, the model performance with a re-trained acoustic model is dwarfed by ACSE, which reached 46.86 % RWERR over noisy data. To get a better understanding of the result, we explore more options in Section 7.5.

Adversarial training is extremely sensitive to hyperparameters [70]. Arguably, better results could be obtained after trying to find a more appropriate combination of hyperparameters. Also, it is possible that different λ weights could work better for models trained with paired data. However, the goal of this experiment was to find whether adversarial training impacts the performance in a positive way, which we have confirmed.

7.5 The Importance of Identity Loss

Paired ACSE showed to improve WER performance over CSE by 0.20 %. It is important to note that Paired ACSE differs from CSE in two major points — Paired ACSE uses *identity loss* and performs *adversarial training*. In this Section, we dive deeper into the experiments to find out the specific role of identity loss and adversarial training. In total, we train four additional model variations. We examine CSE and baseline with identity losses. We also look at how ACSE and paired ACSE perform without identity losses.

This Section is split into two parts — experimenting with models trained with *paired* data and models trained with *unpaired* data. At the end, we discuss the results.

7.5.1 Models Trained with Paired Data

Baseline

For the baseline, we expand the objective function with identity loss, $\lambda L_I(F)$, where the λ weight is the same as the weight used in other models. First 60 iterations, we let the model train without identity loss to learn an underlying structure. Then, the identity loss is added to the objective function. The hyperparameters are the same as for standard baseline.

Model	Acoustic Model	AVG	RWERR
None	Clean	27.70	-
Baseline	Clean	22.88	17.40
Baseline (+ identity)	Clean	**22.38**	**19.21**
Baseline	Re-trained	19.16	30.83
Baseline (+ identity)	Re-trained	**17.90**	**35.38**

Table 7.12: Comparison of ASR WER (%) performances of baseline and baseline trained with identity loss.

Table 7.12 shows that the cycle-consistency constraint improves the performance of baseline. The impact of identity loss is visible more when using a re-trained acoustic model, which reached 17.90 % WER. Surprisingly enough, when compared to other models with re-trained AM, baseline with identity loss outperforms CSE's 18.42 % WER and is only slightly higher than and paired ACSE's 17.72 % WER.

CSE and Paired ACSE

We further explore the impact of identity loss on model performance. We add $\lambda_1 L_I(F)$ and $\lambda_2 L_I(G)$ to CSE. In addition, we remove identity losses from paired ACSE. The hyperparameters remained the same as those used in previous experiments. The λ weights used for CSE are the same as those used in ACSE.

Model	Acoustic Model	AVG	RWERR
None	Clean	27.70	-
CSE	Clean	21.89	20.97
CSE (+ identity)	Clean	**21.35**	**22.92**
Paired ACSE	Clean	**21.69**	**21.70**
Paired ACSE (- identity)	Clean	22.35	19.31
CSE	Re-trained	18.42	33.50
CSE (+ identity)	Re-trained	**18.23**	**34.19**
Paired ACSE	Re-trained	**17.72**	**36.10**
Paired ACSE (- identity)	Re-trained	18.74	32.35

Table 7.13: Comparison of ASR WER (%) performances of models with and without identity loss.

Table 7.13 shows that identity loss indeed improves the models' performance. When extending CSE with only adversarial training, the model degrades in performance. On the other hand, CSE with identity loss only slightly improved the performance when using re-trained AM.

7.5.2 ACSE

We remove identity losses from ACSE to see how it impacts the model performance when trained with unpaired data.

Model	Acoustic Model	AVG	RWERR
None	Clean	27.70	-
ACSE	Clean	**24.18**	**12.71**
ACSE (- identity)	Clean	24.51	11.52
ACSE	Re-trained	**14.72**	**46.86**
ACSE (- identity)	Re-trained	14.99	45.88

Table 7.14: Comparison of ASR WER (%) performances of models with and without identity loss.

As shown in Table 7.14, the removal of identity loss slightly lowers performance. However, it does not hurt the ASR performance with the re-trained acoustic model as much as the removal did in paired ACSE.

7.5.3 Conclusion

Overall, the best model trained with paired data is CSE enhanced with identity losses, without the addition of adversarial training. When evaluating with re-trained AM, the best-performing model is CSE with both adversarial training and identity losses — paired ACSE.

However, standard ACSE, which is trained on unpaired data, performs much better when using a re-trained acoustic model. Its performance with clean AM is worse than the performance of baseline, but with the use of re-trained AM, ACSE reaches 3 % less WER than paired ACSE.

Our deduction from Section 7.3 was observed to be correct. The models were trained using an extension of the training set. However, the evaluation set used to measure ASR WER performance is different. The utterances are spoken by different speakers and the background noise in real data might differ from the noise in simulated data[5]. Also, the provided ASR system might perform better when the acoustic model is trained with noisier data. While ACSE performed worse with clean AM, using adversarial training, in which there is no label, helped the network generalize on unseen data better. Therefore the enhanced data used to re-train the acoustic model reflected further ASR inputs more accurately than data enhanced with other models.

[5]The reasoning is that the process of adding noise to clean recordings improperly reflects real noise and speech captured by the microphone.

7.6 Experimenting with Temporal Convolutional Neural Networks

To measure the impact of the cycle-consistency constraint in other architectures, we wanted to apply the constraint to the state-of-the-art speech enhancement DNN framework. We chose a speech enhancement architecture based on convolutional neural networks called *temporal convolutional neural network* (TCNN) [60]. Our TCNN implementation performed rather poorly. Nevertheless, we summarize the performed experiments here as they can serve as a basis for future work. We briefly describe TCNN basic building blocks, TCNN architecture, and our results.

7.6.1 Basic Building Blocks of TCNN

TCNN is based on a generic temporal convolutional network [4], which utilizes the following principles:

- **fully convolutional networks** [45] - TCNN is able to take input of arbitrary size and effectively produce output of that same size.
- **dilated causal convolutions** [56] - in causal convolutions, the output at time t is convolved only with elements from time t and earlier in the previous layer. The convolution filter is applied over an area larger than the filter by skipping input values with a certain step, called *dilation rate*.
- **residual connections** [26] - residual connections are made within a block, which contains a series of transformations. The output of that sequence is added to the first input of that block at the end of the transformation.

7.6.2 TCNN Architecture

The TCNN architecture is composed of an encoder, **temporal convolutional module** (TCM), and a decoder. The encoder and decoder are composed of two-dimensional causal convolutional layers. The TCM is inserted between the encoder and its mirror-image, the decoder. TCM is largely based on **Conv-TasNet** [48] architecture, which is used for speaker separation. It comprises of one-dimensional causal and dilated convolutional layers. More specifically, TCM consists of three dilation blocks, each formed by stacked six residual blocks. Those residual blocks use *depthwise convolution* [12], which further reduces the number of parameters. The architecture is shown in Figure 7.4.

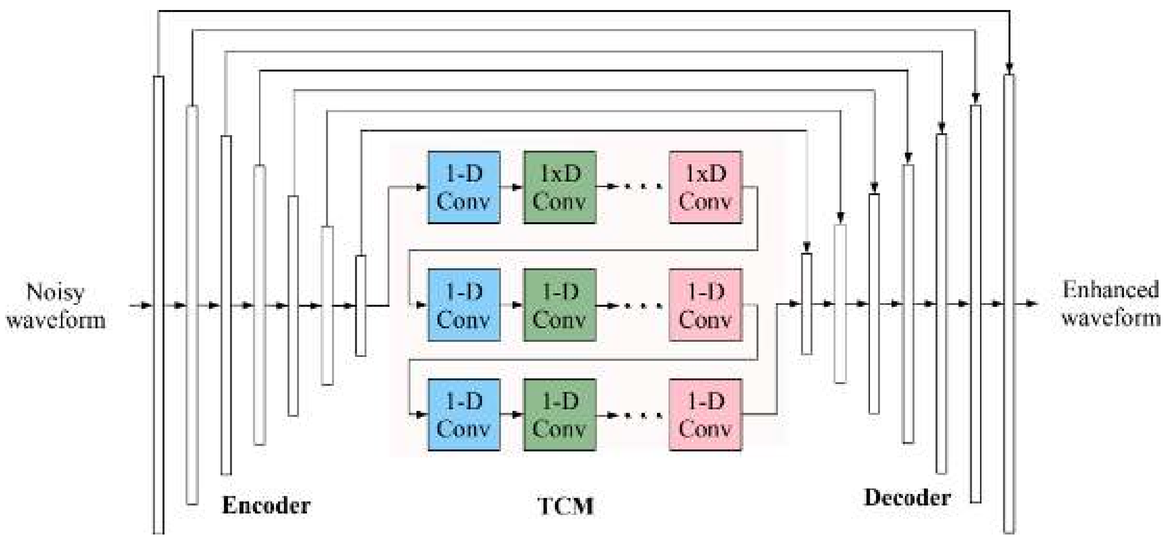


Figure 7.4: TCNN architecture. Encoder and decoder consists of convolutional layers. The input of a layer in the decoder consist of the previous layer's output concatenated with the respective encoder input. Each dilation block within a residual block uses exponentially increased dilation rate. Taken from [60].

Previous models used MFCCs to perform speech enhancement in the frequency domain. TCNN follows a speech enhancement framework for training models in the time domain [58]. It can therefore take advantage of phase information, which can lead to more accurate speech reconstruction [57].

The input and the output of the network consist of a speech signal in the time domain[6]. During the training phase, the network output is used to compute STFT magnitude, which is used to calculate MSE loss. Previously mentioned models used the network outputs, MFFC features, to compute MSE loss. We used a learning rate of $5 \cdot 10^{-5}$, with batch size set at 12. For optimization, we used AdamW. The rest of the parameters are the same as those described in [60].

7.6.3 Results

Model	Noise Environment					
	BUS	PED	CAF	STR	Avg.	RWERR
None	**41.27**	**17.48**	**27.09**	**24.97**	**27.70**	**-**
Baseline	55.14	24.65	31.22	29.86	35.22	-27.15

Table 7.15: ASR WER (%) performance of a baseline model using TCNN architecture.

Table 7.15 shows the results of TCNN baseline. The model has performed poorly. There were many factors that could contribute to these results. Since the baseline model performed much worse when compared to unenhanced data, we decided to not explore the effects of cycle-consistency constraint. The architecture consists of a completely different set of techniques, therefore it is likely that an improperly implemented component could be the reason for those results.

[6]The frames were extracted using a rectangular window of size 20 ms and an overlap of 10 ms

7.7 Summary

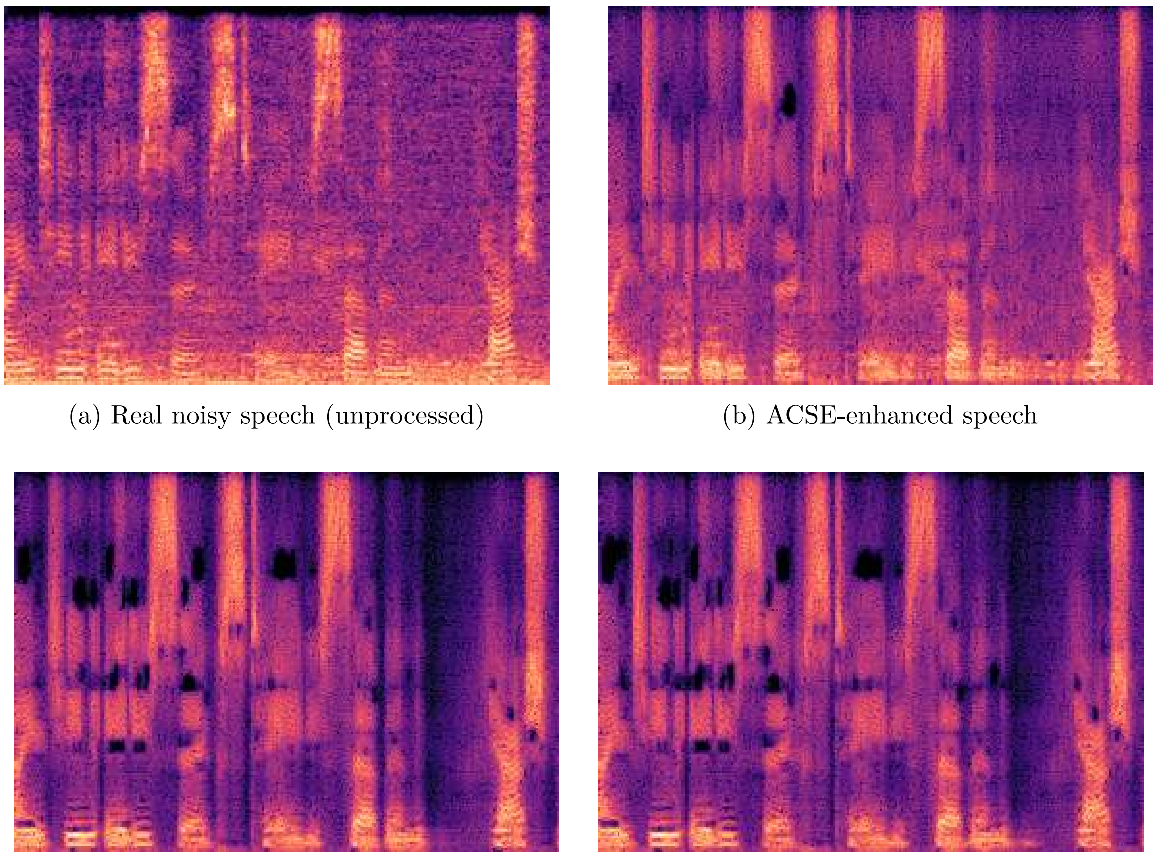
In this work, we performed a large number of experiments with variations of CSE and ACSE. This Section summarizes our findings. We discuss the impact of cycle-consistency, identity loss and adversarial training.

Cycle-Consistency

The cycle-consistency constraint has shown to boost the model performance. Without changing the model structure, we improved RWERR from baseline's 17.40 % to 20.97 %. We can therefore confirm the conclusion from [49] that cycle-consistency improves model performances for speech enhancement tasks.

Adversarial Training

Without using paired data, our ACSE has reached 12.71 % RWERR, which is a significant improvement over [49]'s 6.69 %. By using additional GAN training tricks, we were able to reach such performance. Performing adversarial training using paired data with CSE has shown to further increase RWERR from 20.97 % to 22.92 %. Figure 7.5 shows that CSE and paired ACSE was able to effectively remove noise from the recordings. In ACSE, the phonemes are smeared in time.


(a) Real noisy speech (unprocessed) (b) ACSE-enhanced speech

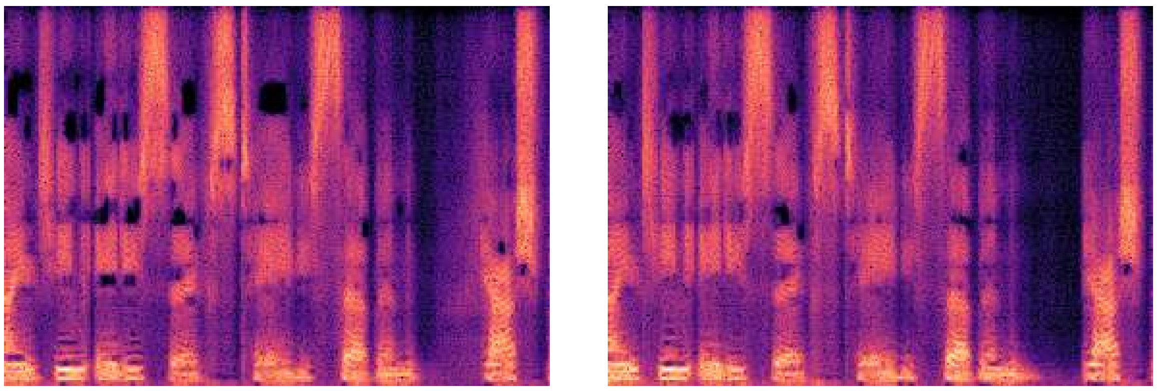
(c) CSE-enhanced speech (d) Paired ACSE-enhanced speech

Figure 7.5: Spectrograms of a noisy utterance enhanced by various models.

Re-training the acoustic model has brought much larger performance jump in models trained using unpaired data. While adversarial CSE with the re-trained acoustic model has reached 33.50 % RWERR, ACSE had reached 46.86 % RWERR. The adversarial training has shown to improve performance when utilizing the re-trained acoustic model. However, the presence of noise in AM training data has the biggest impact on the overall performance. It is also possible that the significant noise suppression caused the acoustic model to improperly represent individual phonemes.

Identity Loss

We further explored the impact of identity loss. Identity loss was originally used in ACSE to regularize the generator networks to keep their respective outputs close to the identity-mapping functions. The addition of identity loss has brought improvement not only to the performance of ACSE but also to the baseline and CSE performance, as well. The addition of identity loss to baseline and CSE improved their respective RWERR by 1.50 % and 1.95 %, and by 4.55 % and 0.69 % when using re-trained AM. Figure 7.6 shows spectrograms of a noisy speech signal enhanced by baseline and baseline with identity loss.



(a) Speech enhanced by baseline (b) Speech enhanced with baseline (+ identity)

Figure 7.6: Spectrograms of a real utterance enhanced by baseline and baseline with identity loss.

Chapter 8

Conclusion

8.1 Summary of the Performed Work

The goal of this thesis was to experiment with neural networks using cycle-consistency constraints for speech enhancement. This thesis is based on a publication that evaluates cycle-consistent neural networks for speech enhancement [49]. Cycle-consistent neural networks use a second neural network for training the network, which performs an opposite task. In our case, it was noise insertion into a clean speech signal. The networks are then jointly trained using an extended objective function. Using two networks that are dual to each other, the original noisy or clean speech signal can be reconstructed. The reconstruction loss is then used in the final objective function.

However, training data consisting of input-target pairs (paired data) is not always available. We utilize generative adversarial networks (GANs) with cycle-consistency to train a network using only unpaired data. In generative adversarial networks, two neural networks are pitted against each other, each attempting to reach its objective, which is "adversary" to the other network. We combine GANs with cycle-consistency to construct CycleGAN, which was originally proposed in [84]. CycleGAN consists of four neural networks in total — two discriminators and two generators.

It was demonstrated that the presence of cycle-consistency constraint drastically improves the network ASR performance, outperforming the models from the reference publication. Cycle-consistent network trained with paired data (CSE) achieves 20.97 % relative WER reduction (RWERR) over noisy data. Backward cycle-consistency was shown to provide a significant improvement when used in combination with forward cycle-consistency. Using unpaired data, our implementation of CycleGAN (ACSE) achieves 12.71 % RWERR, which is a large improvement over [49]'s ACSE, which reached 6.69 % relative WER reduction. Using various GAN training tricks helped improve ACSE performance. To obtain better results, we re-trained the DNN acoustic model (AM) used in Kaldi using data enhanced by the trained models. Using re-trained AM, the RWERR performance of CSE and ACSE further increased to 33.50 % and 46.86 %, respectively.

Lastly, we investigated the impact of adversarial training and identity loss on ASR performance. The addition of identity loss improved the performance of both CSE and baseline models. Using both adversarial training and identity loss in CSE led to additional significant RWERR improvement to 36.10 % with re-trained AM.

8.2 Future Work

Cycle-consistency is a relatively new concept in the deep learning field. The principle on its own is quite simple, however, there are many ways the use of cycle-consistency can be pushed to its limits.

8.2.1 Short-Term Prospects

While we experimented with various hyperparameters to train the best possible model, we have not modified the λ weight coefficients of the respective loss functions. Fine-tuning the coefficients could lead to significant performance improvements in both CSE and ACSE.

8.2.2 Long-Term Prospects

Neural networks successfully trained in this work perform speech enhancement in the frequency domain. While our TCNN implementation did not work well, performing the enhancement within the time domain has recently shown to be successful [62, 58, 59]. Below are described possible architecture replacements for LSTMs which were used in this work.

Transformer Neural Networks

Recently, **transformer neural networks** (TNNs) [75] have demonstrated state-of-the-art performance on natural language processing (NLP) tasks. *Self-attention* is a core building block of a TNN, that allows computing symbol-by-symbol correlations over an entire input sequence in parallel. The self-attention mechanism in TNNs traverses a fixed amount of time steps, before computing attention weights, which has shown to benefit many NLP tasks. Kim et al. have proposed TNNs with Gaussian-weighted self-attention mechanism for speech enhancement [34].

Dual-Path RNNs

Initially proposed for speaker separation task, **dual-path RNNs** (DPRNNs) [47] could be coupled with cycle-consistency. DPRNN splits the sequential input into shorter overlapping chunks. A DPRNN layer consists of two interleaved bi-directional LSTMs (BLSTMs) [71]. One network locally processes chunks independently, while the other network aggregates the information from all the chunks to process the utterance as a whole. Replacing depthwise convolutions with DPRNNs in TasNet [48] led to state-of-the-art performance.

Teacher-Student Learning

Instead of combining cycle-consistency with adversarial training, a **teacher-student** (T-S) learning [43] approach can be used for speech enhancement via unsupervised domain adaptation[1] [42]. In T-S learning, a teacher network is used to teach a student network to make the same predictions as the teacher.

[1]Domain adaptation attempts to transfer the knowledge obtained from the source domain to the target domain.

Bibliography

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A. et al. Tensorflow: A system for large-scale machine learning. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16).* 2016, p. 265–283.

[2] Allen, J. B. and Rabiner, L. R. A unified approach to short-time Fourier analysis and synthesis. *Proceedings of the IEEE.* IEEE. 1977, vol. 65, no. 11, p. 1558–1564.

[3] Avargel, Y. and Cohen, I. System identification in the short-time Fourier transform domain with crossband filtering. *IEEE Transactions on Audio, Speech, and Language Processing.* IEEE. 2007, vol. 15, no. 4, p. 1305–1319.

[4] Bai, S., Kolter, J. Z. and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv preprint arXiv:1803.01271.* 2018.

[5] Barker, J., Marxer, R., Vincent, E. and Watanabe, S. The third 'CHiME'speech separation and recognition challenge: Dataset, task and baselines. In: IEEE. *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU).* 2015, p. 504–511.

[6] Barker, J., Watanabe, S., Vincent, E. and Trmal, J. The fifth'CHiME'Speech Separation and Recognition Challenge: Dataset, task and baselines. *ArXiv preprint arXiv:1803.10609.* 2018.

[7] Baydin, A. G., Pearlmutter, B. A., Radul, A. A. and Siskind, J. M. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research.* JMLR. org. 2017, vol. 18, no. 1, p. 5595–5637.

[8] Bengio, Y., Simard, P., Frasconi, P. et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks.* 1994, vol. 5, no. 2, p. 157–166.

[9] Bishop, C. M. *Pattern recognition and machine learning.* Springer, 2006.

[10] Boll, S. Suppression of acoustic noise in speech using spectral subtraction. *IEEE Transactions on acoustics, speech, and signal processing.* IEEE. 1979, vol. 27, no. 2, p. 113–120.

[11] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *ArXiv preprint arXiv:1406.1078.* 2014.

[12] Chollet, F. Xception: Deep learning with depthwise separable convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, p. 1251–1258.

[13] Dahl, G. E., Yu, D., Deng, L. and Acero, A. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing.* IEEE. 2011, vol. 20, no. 1, p. 30–42.

[14] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics.* Springer. 1980, vol. 36, no. 4, p. 193–202.

[15] Gales, M. J. Maximum likelihood linear transformations for HMM-based speech recognition. *Computer speech & language.* Elsevier. 1998, vol. 12, no. 2, p. 75–98.

[16] Ganin, Y. and Lempitsky, V. Unsupervised domain adaptation by backpropagation. *ArXiv preprint arXiv:1409.7495.* 2014.

[17] Garofalo, J., Graff, D., Paul, D. and Pallett, D. CSR-I (WSJ0) Complete. *Linguistic Data Consortium, Philadelphia.* 2007.

[18] Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics.* 2010, p. 249–256.

[19] Glorot, X., Bordes, A. and Bengio, Y. Deep sparse rectifier neural networks. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics.* 2011, p. 315–323.

[20] Goodfellow, I., Bengio, Y. and Courville, A. *Deep learning.* MIT press, 2016.

[21] Goodfellow, I., Pouget Abadie, J., Mirza, M., Xu, B., Warde Farley, D. et al. Generative adversarial nets. In: *Advances in neural information processing systems.* 2014, p. 2672–2680.

[22] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R. and Schmidhuber, J. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems.* IEEE. 2016, vol. 28, no. 10, p. 2222–2232.

[23] Griffin, D. and Lim, J. Signal estimation from modified short-time Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing.* IEEE. 1984, vol. 32, no. 2, p. 236–243.

[24] Harley, A., Wei, S.-E., Saragih, J. and Fragkiadaki, K. Image disentanglement and uncooperative re-entanglement for high-fidelity image-to-image translation. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops.* 2019, p. 0–0.

[25] He, D., Xia, Y., Qin, T., Wang, L., Yu, N. et al. Dual learning for machine translation. In: *Advances in Neural Information Processing Systems.* 2016, p. 820–828.

[26] He, K., Zhang, X., Ren, S. and Sun, J. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, p. 770–778.

[27] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r. et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine.* IEEE. 2012, vol. 29, no. 6, p. 82–97.

[28] Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation.* MIT Press. 1997, vol. 9, no. 8, p. 1735–1780.

[29] Hoffer, E., Hubara, I. and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In: *Advances in Neural Information Processing Systems.* 2017, p. 1731–1741.

[30] Jason, A. *DeOldify.* 2019 [Online; visited 30. 12. 2019]. Available at: https://github.com/jantic/DeOldify.

[31] Jozefowicz, R., Zaremba, W. and Sutskever, I. An empirical exploration of recurrent network architectures. In: *International Conference on Machine Learning.* 2015, p. 2342–2350.

[32] Kalal, Z., Mikolajczyk, K. and Matas, J. Forward-backward error: Automatic detection of tracking failures. In: IEEE. *2010 20th International Conference on Pattern Recognition.* 2010, p. 2756–2759.

[33] Karras, T., Laine, S. and Aila, T. A style-based generator architecture for generative adversarial networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2019, p. 4401–4410.

[34] Kim, J., El Khamy, M. and Lee, J. T-GSA: Transformer with Gaussian-Weighted Self-Attention for Speech Enhancement. In: IEEE. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2020, p. 6649–6653.

[35] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *ArXiv preprint arXiv:1412.6980.* 2014.

[36] Kinoshita, K., Delcroix, M., Gannot, S., Habets, E. A., Haeb Umbach, R. et al. A summary of the REVERB challenge: state-of-the-art and remaining challenges in reverberant speech processing research. *EURASIP Journal on Advances in Signal Processing.* Nature Publishing Group. 2016, vol. 2016, no. 1, p. 7.

[37] Krizhevsky, A., Sutskever, I. and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems.* 2012, p. 1097–1105.

[38] LeCun, Y., Bengio, Y. and Hinton, G. Deep learning. *Nature.* Nature Publishing Group. 2015, vol. 521, no. 7553, p. 436–444.

[39] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE.* Ieee. 1998, vol. 86, no. 11, p. 2278–2324.

[40] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A. et al. Photo-realistic single image super-resolution using a generative adversarial network. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, p. 4681–4690.

[41] Li, J., Deng, L., Gong, Y. and Haeb Umbach, R. An overview of noise-robust automatic speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing.* IEEE. 2014, vol. 22, no. 4, p. 745–777.

[42] Li, J., Seltzer, M. L., Wang, X., Zhao, R. and Gong, Y. Large-scale domain adaptation via teacher-student learning. *ArXiv preprint arXiv:1708.05466.* 2017.

[43] Li, J., Zhao, R., Huang, J.-T. and Gong, Y. Learning small-size DNN with output-distribution-based criteria. In: *Fifteenth annual conference of the international speech communication association.* 2014.

[44] Loizou, P. C. *Speech enhancement: theory and practice.* CRC press, 2013.

[45] Long, J., Shelhamer, E. and Darrell, T. Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, p. 3431–3440.

[46] Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *ArXiv preprint arXiv:1711.05101.* 2017.

[47] Luo, Y., Chen, Z. and Yoshioka, T. Dual-path rnn: efficient long sequence modeling for time-domain single-channel speech separation. In: IEEE. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2020, p. 46–50.

[48] Luo, Y. and Mesgarani, N. Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM transactions on audio, speech, and language processing.* IEEE. 2019, vol. 27, no. 8, p. 1256–1266.

[49] Meng, Z., Li, J., Gong, Y. et al. Cycle-consistent speech enhancement. *ArXiv preprint arXiv:1809.02253.* 2018.

[50] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems.* 2013, p. 3111–3119.

[51] Mimura, M., Sakai, S. and Kawahara, T. Cross-domain speech recognition using nonparallel corpora with cycle-consistent adversarial networks. In: IEEE. *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU).* 2017, p. 134–140.

[52] Mirza, M. and Osindero, S. Conditional generative adversarial nets. *ArXiv preprint arXiv:1411.1784.* 2014.

[53] Mohri, M., Pereira, F. and Riley, M. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*. Elsevier. 2002, vol. 16, no. 1, p. 69–88.

[54] Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, p. 807–814.

[55] Olah, C. *Colah's blog*. Aug 2015 [Online; visited 25. 12. 2019]. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[56] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O. et al. Wavenet: A generative model for raw audio. *ArXiv preprint arXiv:1609.03499*. 2016.

[57] Paliwal, K., Wójcicki, K. and Shannon, B. The importance of phase in speech enhancement. *Speech communication*. Elsevier. 2011, vol. 53, no. 4, p. 465–494.

[58] Pandey, A. and Wang, D. A New Framework for Supervised Speech Enhancement in the Time Domain. In: *Interspeech*. 2018, p. 1136–1140.

[59] Pandey, A. and Wang, D. A new framework for CNN-based speech enhancement in the time domain. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. IEEE. 2019, vol. 27, no. 7, p. 1179–1188.

[60] Pandey, A. and Wang, D. TCNN: Temporal convolutional neural network for real-time speech enhancement in the time domain. In: IEEE. *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, p. 6875–6879.

[61] Pascanu, R., Mikolov, T. and Bengio, Y. On the difficulty of training recurrent neural networks. In: *International conference on machine learning*. 2013, p. 1310–1318.

[62] Pascual, S., Bonafonte, A. and Serra, J. SEGAN: Speech enhancement generative adversarial network. *ArXiv preprint arXiv:1703.09452*. 2017.

[63] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché Buc, F. d', Fox, E. et al., ed. *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, p. 8024–8035.

[64] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O. et al. The Kaldi speech recognition toolkit. In: IEEE Signal Processing Society. *IEEE 2011 workshop on automatic speech recognition and understanding*. 2011.

[65] Radford, A., Metz, L. and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *ArXiv preprint arXiv:1511.06434*. 2015.

[66] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. Learning representations by back-propagating errors. *Nature*. Nature Publishing Group. 1986, vol. 323, no. 6088, p. 533–536.

[67] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S. et al. Imagenet large scale visual recognition challenge. *International journal of computer vision.* Springer. 2015, vol. 115, no. 3, p. 211–252.

[68] Sainath, T. N., Weiss, R. J., Senior, A., Wilson, K. W. and Vinyals, O. Learning the speech front-end with raw waveform CLDNNs. In: *Sixteenth Annual Conference of the International Speech Communication Association.* 2015.

[69] Sak, H., Senior, A. W. and Beaufays, F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. 2014.

[70] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. et al. Improved techniques for training gans. In: *Advances in neural information processing systems.* 2016, p. 2234–2242.

[71] Schuster, M. and Paliwal, K. K. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing.* Ieee. 1997, vol. 45, no. 11, p. 2673–2681.

[72] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W. et al. Learning from simulated and unsupervised images through adversarial training. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, p. 2107–2116.

[73] Smith, S. L., Kindermans, P.-J., Ying, C. and Le, Q. V. Don't decay the learning rate, increase the batch size. *ArXiv preprint arXiv:1711.00489.* 2017.

[74] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. Rethinking the inception architecture for computer vision. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, p. 2818–2826.

[75] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. Attention is all you need. In: *Advances in neural information processing systems.* 2017, p. 5998–6008.

[76] Wang, D. and Lim, J. The unimportance of phase in speech enhancement. *IEEE Transactions on Acoustics, Speech, and Signal Processing.* IEEE. 1982, vol. 30, no. 4, p. 679–681.

[77] Wang, K., Zhang, J., Sun, S., Wang, Y., Xiang, F. et al. Investigating generative adversarial networks based speech dereverberation for robust speech recognition. *ArXiv preprint arXiv:1803.10132.* 2018.

[78] Watanabe, S., Delcroix, M., Metze, F. and Hershey, J. R. *New era for robust speech recognition: exploiting deep learning.* Springer, 2017.

[79] Wilson, A. C., Roelofs, R., Stern, M., Srebro, N. and Recht, B. The marginal value of adaptive gradient methods in machine learning. In: *Advances in Neural Information Processing Systems.* 2017, p. 4148–4158.

[80] Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M. et al. Achieving human parity in conversational speech recognition. *ArXiv preprint arXiv:1610.05256.* 2016.

[81] Yoshioka, T., Sehr, A., Delcroix, M., Kinoshita, K., Maas, R. et al. Making machines understand us in reverberant rooms: Robustness against reverberation for automatic speech recognition. *IEEE Signal Processing Magazine.* IEEE. 2012, vol. 29, no. 6, p. 114–126.

[82] Zhang, Z., Geiger, J., Pohjalainen, J., Mousa, A. E.-D., Jin, W. et al. Deep learning for environmentally robust speech recognition: An overview of recent developments. *ACM Transactions on Intelligent Systems and Technology (TIST).* ACM. 2018, vol. 9, no. 5, p. 49.

[83] Zhou, Y.-T. and Chellappa, R. Computation of optical flow using a neural network. In: *IEEE International Conference on Neural Networks.* 1988, p. 71–78.

[84] Zhu, J.-Y., Park, T., Isola, P. and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In: *Proceedings of the IEEE international conference on computer vision.* 2017, p. 2223–2232.

Appendix A

CD Content

The attached CD contains following items:

- **sources** folder, which contains the source files and task scripts
- **examples** folder, which contains example noisy and enhanced speech utterances
- **trained_models** folder contains several trained network models
- the text of this thesis in **DT_xkarli05.pdf**
- **text** folder, which contains LaTeX source files
- **README** file, which contains a detailed description of the contents of the CD