Tampere University

Shayan Gharib

UNSUPERVISED DOMAIN ADAPTATION FOR AUDIO CLASSIFICATION

Master of Science Thesis
Faculty of Information Technology and Communication Sciences (ITC)
Examiners: Prof. Tuomas Virtanen
Dr. Konstantinos Drossos
May 2020

ABSTRACT

Shayan Gharib: Unsupervised Domain Adaptation for Audio Classification
Master of Science Thesis
Tampere University
Audio-Visual Signal Processing
May 2020

Machine learning algorithms have achieved the state-of-the-art results by utilizing deep neural networks (DNNs) across different tasks in recent years. However, the performance of DNNs suffers from mismatched conditions between training and test datasets. This is a general machine learning problem in all applications such as machine vision, natural language processing, and audio processing. For instance, the usage of different recording devices and ambient noises can be referred to as some of the causing factors for mismatched conditions between training and test datasets in audio classification tasks. Due to mismatched conditions, a well-performed DNN model in training phase encounters a decrease in performance when evaluated on unseen data. To compensate the reduction of performance caused by this issue, domain adaptation methods have been employed to adapt DNNs to conditions in test dataset.

The objective of this thesis is to study unsupervised domain adaptation using adversarial training for acoustic scene classification. We first pre-train a model using data from one set of conditions. subsequently, we retrain the model using data with another set of conditions in order to adapt the model such that the output of the model is condition-invariant representations of inputs. More specifically, we place a discriminator against the model. The aim of the discriminator is to distinguish between data coming from different conditions, while the goal of the model is to confuse the discriminator such that it is not able to differentiate between data with different conditions. The data that we use to optimize our models on, e.g. training data, have been recorded using different recording devices than the ones utilized in the dataset used to evaluate the model, e.g. test dataset. The training data is recorded using a high quality recording device, while the audio recordings in the test set have been collected using two handheld consumer devices with mediocre quality. This introduces a mismatched condition which negatively affects the performance of optimized DNN. In this thesis, we simulate a scenario in which we do not have access to the annotations of the dataset that we try to adapt our model to. Therefore, our method can be used as an unsupervised domain adaptation technique. In addition, we present our results using two different DNN architectures, namely Kaggle and DCASE models, to show that our method is model agnostic, and works regardless of the used models. The results show a significant improvement for the adapted Kaggle and DCASE models compared to the non-adapted ones by an approximate increase of 11% and 6% respectively in the performance for unseen test data while maintaining the same performance on unseen samples from training data.

Keywords: domain adaptation, acoustic scene classification, unsupervised domain adaptation, adversarial training, audio classification

PREFACE

The implementation of this thesis was undertaken in 2018 at the Audio Research Group (ARG) of Tampere University (previously known as Tampere University of Technology (TUT)).

First and foremost, I would like to express my sincere gratitude to Professor Tuomas Virtanen as my first supervisor in this thesis who believed in me, and gave me the opportunity to work as a research assistant at ARG. I would like to thank him for giving me encouragement and freedom to explore my research aspirations, and for always finding time to meet and dedicate effort during the course of this thesis despite his very busy schedule.

I would like to greatly appreciate Dr. Konstantinos Drossos as my second supervisor for his kindness in welcoming and guiding me by holding regular meetings with me to discuss the obstacles in the implementation of this thesis. I would like to thank him for his intuition on directing the work to the right path, and for his advice and tireless supervision in this endeavor.

I would like to thank Dr. Emre Cakir who helped me to quickly integrate into the group, and to become familiar with each and every group member. I would like to extend my thanks to all members of ARG for creating a positive and friendly working environment during this journey.

Last but not least, I would like to give my most special thanks to my family. To my father and my mother who have been my most influential teachers throughout my life and wholeheartedly supported me from the first steps in my education with the very best of their ability.

Tampere, 13th May 2020

Shayan Gharib

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial intelligence
ANN	Artificial neural network
ASC	Acoustic scene classification
ASR	Automatic speech recognition
CPU	Central processing unit
CQT	Constant-Q transform
DCASE	Detection and classification of acoustic scenes and events
DCT	Discrete cosine transform
DFT	Discrete Fourier transform
DNN	Deep neural network
GAN	Generative adversarial training
GMM	Gaussian mixture model
GPU	Graphics processing unit
GRL	Gradient reversal layer
HMM	Hidden Markov model
Hz	Hertz
MFCC	Mel-frequency cepstrum
MLP	Multilayer perceptron
ML	Machine learning
ms	Milliseconds
NMF	Non-negative matrix factorization
RBF	Radial basis function
ReLU	Rectified linear unit
SED	Sound event detection
SGD	Stochastic gradient descent
STFT	Short-time Fourier transform
SVM	Support vector machine
TanH	Hyperbolic tangent
TAU	Tampere University

TUNI	Tampere Universities
URL	Uniform Resource Locator

1 INTRODUCTION

Sounds carry information by which it is possible to recognize and perceive environments and the events happening in them [2, 3]. Unique characteristics of each event and environment making them distinguishable for humans' auditory system [2]. As a result, we are easily able to recognize the sound events such as footsteps, bird singing, speech, and music when we listen to our surrounding environment on a daily basis routine. On top of that, we are capable of understanding and classifying the environment and background of a sound only by listening to an audio file, and without being present. Although this task may seem trivial in the everyday life of humans, it has been a complex task to mathematically formulate how the auditory system of humans works. *Computational acoustic scene analysis* (CASA) has emerged in the audio community to find solutions that can help machines to understand and recognize the source of acoustic scenes and events [2].

One of the research fields that has been derived from CASA community is *acoustic scene classification* (ASC). The goal of ASC is to train machines that are able to classify audio recordings according to the environment – e.g. office, restaurant, street, etc – that they have been recorded in [4]. As the number of wearable and handheld consumer devices is extensively growing, it is of great significance to provide users of these devices with more environment-related information in order to enable devices to operate more efficiently and naturally in interactions with humans [5]. For example, a cellphone, as an handheld consumer device, can automatically adjust its mode based on surrounding environments, more specifically, it can automatically reject a call when its user is in a meeting, or provide local information from the immediate environment [6]. The first set of methods used for ASC were Gaussian mixture models (GMMs), hidden Markov models (HMMs), support vector machine (SVM), and Non-negative matrix factorization (NMF) [7, 8, 9]. Thanks to the recent advent of large-scaled dataset, the recent state-of-the-art results have been achieved by the employment of deep neural networks (DNNs) [10, 11].

In spite of this flourishing results, DNN approaches do not work properly when it comes to real-life applications [12]. There are different reasons that cause the current methods deficient in validity of their performance. All causing factors normally affect the conditions in which data have been collected [13]. In general, conventional machine learning algorithms presume that the conditions in which training and test data have been collected are similar, more formally, the marginal distribution of training and test data are similar [14]. This rarely occurs in a more realistic scenario where the possibility of data

collection for the development and evaluation at the same time is very low. Therefore, different conditions such as various recording devices and recorder positions reduce the performance of learning algorithms in the evaluation phase [12]. This problem is referred to as *dataset shift* in various literature [13]. Domain adaptation addresses the problem of dataset shift in the context of machine learning [14]. Using domain adaptation methods, we try to compensate the reduction of performance in the evaluation phase and adapt DNNs to the conditions in test data in order to perform as similar as possible to the obtained results in the development phase.

The structure of this thesis is organized as follows. Chapter 2 describes the ASC task and introduce common acoustic features that are used in this task. This chapter also review the background theory of artificial neural networks used in this thesis. Lastly, it presents the theory behind learning from different dataset and reviews recent domain adaptation works in order to address this problem. Chapter 3 contains the actual methodology used to implement the work in this thesis. In Chapter 4, we discuss our experimental setup and evaluation results during the implementation of our method. Eventually, Chapter 5 contains our conclusions, comments regarding our observations of the methods, and possible research that can lead to improvements over the results acquired using the method introduced in this thesis.

A part of the research and obtained results leading to the fulfilment of this thesis have been published in the Workshop on Detection and Classification of Acoustic Scenes and Events 2018 (DCASE 2018).

2 BACKGROUND

2.1 Acoustic Scene Classification

Information about our environment is transported through sound to our auditory system. Sound contains information of *sound events* and *sound scenes*. A sound event is a description that is used to identify and distinguish various auditory events originated from physical objects within an environment such as the sound of a car passing by, a bird singing, a car horn, etc [15]. On the other hand, sound scenes are descriptors of the environment from which a sound is originated, e.g. park, street, home, etc. [2].

Computational analysis of sound events and scenes aims at employing computational methods to learn a higher representation of sounds in order to address a task in different applications. These tasks are categorized into classification and detection [2]. In audio, detection refers to the process of locating the temporal position for active sound events. However, classification refers to the process in which an input data (e.g. audio) is categorized into one or multiple classes (i.e. single-label or multi-label classification). The focus of this thesis is on the classification of acoustic scenes (Figure 2.1).

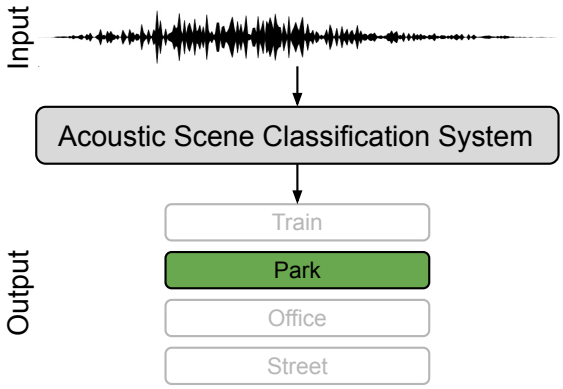


Figure 2.1. *A schematic of acoustic scene classification system. A segment of audio signal is used as an input to the system and an output is predicted as a semantic label for the corresponding audio signal. The figure is adopted from [12].*

An ASC system consists of 3 main steps: audio signal processing, learning, and recognition [2]. Audio signal processing step is typically implemented through two phases. First phase is the preprocessing of audio data in which audio segments are unified to contain the same properties such as sampling rate. In addition, audio segments are one-dimensional representation of sounds, and do not provide distinctive information for

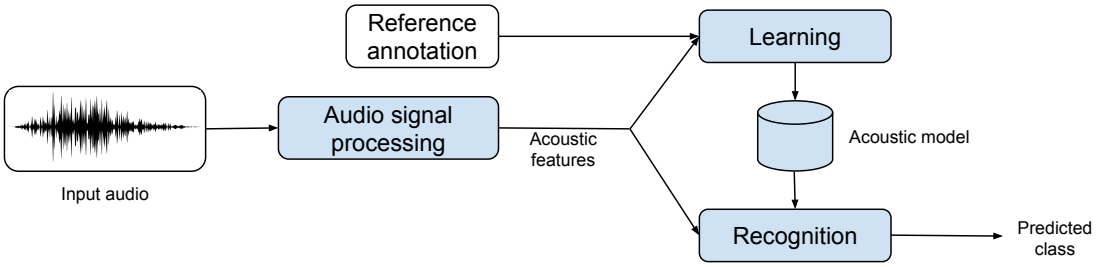


Figure 2.2. *The component of a typical acoustic scene classification system. The figure is adopted from [2].*

classification purposes. Therefore, the feature extraction phase prepares the audio segments in order to be used as inputs for learning and recognition steps (discussed in section 2.1.1). In learning step, we aim at developing an acoustic model for ASC. Subsequently, in the recognition step, we employ the acoustic model to make predictions for each unseen audio segment in an evaluation process.

2.1.1 Audio Signal Processing

Audio signal processing is the first step of an ASC system. It is usually organized in two phases: preprocessing and feature extraction. In this subsection, we explain the reason for using this two stages before applying machine learning algorithms on audio data.

Preprocessing

Datasets often contain samples collected from different sources resulting in inconsistent properties of audio segments [2]. For example, audio segments are recorded using various recording devices and microphones may contain different number of channels, noises, and sampling rates. Preprocessing is done prior to the feature extraction phase in order to unify the characteristic of audio segments and to amplify certain target sounds [2]. In other words, it allows to fix the number of channels by down-mixing, and to adopt a uniform sampling rate for all available audio segments. Additionally, we may suppress some ambient noises using noise suppression techniques in order to improve the performance, and remove some overlapping sounds using source separation techniques to reduce interference of sounds [2].

Feature Extraction

In contrast to image classification in which an image is directly employed as an input to a classification system [16], in audio, there is a need to extract acoustic features from raw audio. Acoustic features are used to reduce redundancy and achieve a compact and expressive representation of audio data which are efficient in terms of memory usage and the required computational power of classification systems [2]. Moreover, feature

extraction can lead to the improvement on the performance of classification systems by boosting the intra-class similarity, and simultaneously increase the inter-class variability in extracted acoustic features [2].

Characteristics (e.g. amplitude) of a sound in its temporal waveform does not provide adequate information for computational analysis of an audio signal in tasks such as ASC. In the presence of additive noise, this representation is easily distorted resulting in an altered waveform. In addition, the representation of audio signals in time domain introduces redundancy to classification systems. To avoid this problem, spectral (i.e. frequency-based) representation – that is a higher representation – of audio signals have shown to be more robust features for audio classification and detection tasks[2]. Frequency domain representation, obtained by Fourier transform, contains the magnitude and phase of each frequency (in audio the phase is discarded). In spite of that, the sheer representation of signals in frequency domain lacks to depict a time-varying representation of the frequencies spectrum. Since environmental sounds generally vary based on time, the time-frequency representation, namely spectrogram, of an audio signal has been adopted in the most recent state-of-the-art methods for ASC [10, 11]. To obtain the frequency domain representation of an audio signal, we use short time Fourier transform (STFT) in order to determine the constituent sinusoidal frequencies of short segments of audio signals. In addition to frequency domain representation, this transformation allows us to align the frequency information with time as the audio signals vary constantly [17]. Below, we describe some of the recent and common feature extraction techniques that are utilized for the purpose of audio classification:

Short-time Fourier transform (STFT): The most prevalent method to transform a signal into a sum of sinusoidal waveforms is the Fourier transform. In other words, the Fourier transform decomposes a signal into its constituent frequencies (in Hertz). The discrete Fourier transform (DFT) is one of the variations of Fourier transform used in many practical applications for finite and discrete signals. According to [18] for a discrete signal x with the length of N, the DFT is formally defined as below:

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}} \tag{2.1}$$

Where $X(k)$ is the calculated DFT at frequency k. In general, audio signals vary over time and therefore are non-stationary. To apply the DFT on digital audio signals, we consider only a short frame of an audio in which the audio signal is assumed to be stationary [2]. To do so, we first segment the audio into successive short frames – in the range of tens of milliseconds. Subsequently, a windowing function , e.g. Hamming window, is applied to each frame, and finally, the DFT is applied to each windowed frame. This process is called *short-time Fourier transform* (STFT). In practice, the successive frames overlap with each other in order to avoid drastic changes at the edges of each frame, and to prevent losing information. In addition, there is an opposite relation for time and frequency resolution in the STFT. More specifically, a wide window results in losing fast changes in time, but it

creates a high resolution in frequency. In contrast, a narrow window is able to detect fast changes in time at the cost of an inferior resolution in frequency [19].

Log mel energies: Psychoacoustic studies [20] proved that humans perceive the frequency on a nonlinear scale. Inspired by this, mel – a logarithmic and subjective – scale was introduced to convert linear frequencies into a different scale in which various frequencies of sounds, that are perceived the same by humans, are placed in one band. In other words, all frequencies are perceptually equal in one mel band. This is to simulate the fact that the auditory system of humans is not capable of discerning any difference between adjacent frequencies, and this effect becomes more noticeable as frequencies increase [21]. The simulation of this effect can be seen in Figure 2.3.

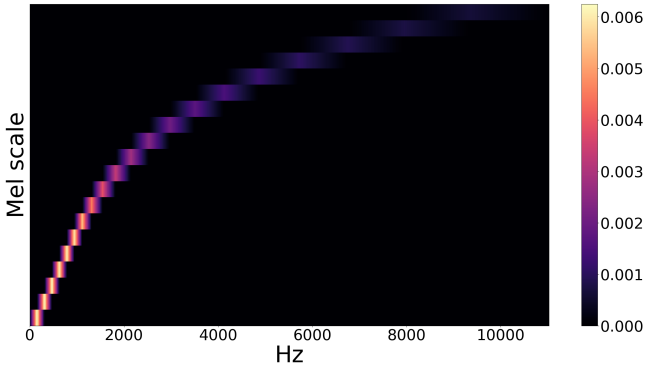


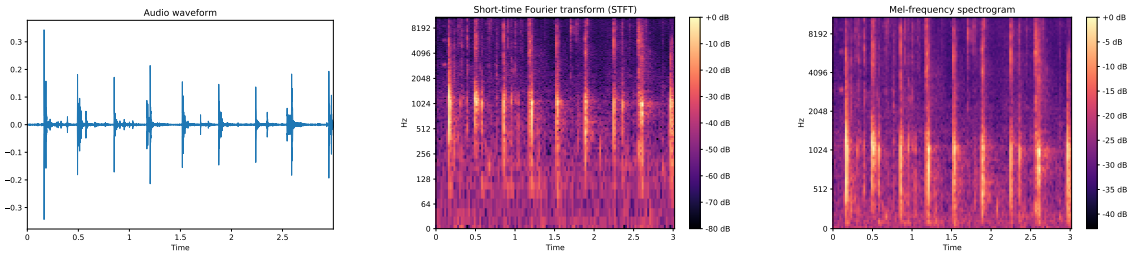
Figure 2.3. *The illustration of 20 mel filters (i.e. filter bank) to convert linear frequencies (Hz) to mel scale. The figure is plotted using [22].*

To obtain mel energies of an audio signal, we first compute the magnitude of the DFT for each windowed frame (i.e. STFT) of the audio. This results in a time-frequency representation in which each windowed frame contains the magnitude of the computed frequencies. In addition, a linear frequency f (in Hertz) can be converted to mel scale using following formula [23]:

$$m = 2595 \log_{10} \left(1 + \frac{\mathrm{f}}{700}\right) \tag{2.2}$$

Then each coefficient in the magnitude of STFT is multiplied by the corresponding gain in the mel filters. The results are accumulated to obtain the value of each bin in the calculated mel energies. The number of mel band in the spectrogram is equal to the number of used mel filters. Finally, an element-wise logarithm operation is applied resulting in *log mel energies*. This simulates the way humans react to changes in the loudness of a sound. In other words, the auditory system of humans does not perceive the energies of a sound on a linear scale.

Mel-frequency cepstral coefficients (MFCCs): One of the most popular acoustic features for the automatic speech recognition (ASR) task is mel-frequency cepstral coefficients (MFCCs) which represent the spectral envelop of a signal [24]. To obtain the MFCCs, it is required to decorrelate the log mel energies. Since adjacent mel filters are



(a) *Audio waveform* ***(b)*** *STFT spectrogram* ***(c)*** *Mel spectrogram*

Figure 2.4. *Time and time-frequency domain representations of a 3-second audio recording of a horse walking.*

overlapping, the log mel spectral vectors of each frame are highly correlated with one another. Therefore, after computing the log mel spectrum, a transformation which is called the discrete cosine transform (DCT) is used to decorrelate the components of log mel spectral vectors [25]. In addition, this results in reduced number of coefficients, since the higher coefficients are discarded. More specifically, higher coefficients of MFCCs are indicators of finer spectral details – e.g. pitch – which do not provide discriminative information for ASR [26]. This has been shown to benefit simpler classifiers such as HMMs and GMMs [24, 27].

2.1.2 Machine Learning for Acoustic Scene Classification

The ASC task investigates the computational analysis of sound scenes. Sound scenes refer to the acoustic environments in which audio segments are captured such as home, office, and street, to name but a few. The goal in ASC is to assign one of the predefined class descriptors to each audio segment which correctly describes the environment where the audio is recorded.

According to [28], machine learning algorithms learn from examples through experience. In various applications, there are challenging tasks for which making a human-engineered indicator cannot be efficiently implemented. On the other hand, machine learning algorithms are to automatically build a mathematical model in order to find a solution for a given task. Similarly in ASC, It is difficult, if not infeasible, to engineer a model by conventional algorithms. However, machine learning methods have been shown as potential algorithms to map the acoustic features into target class labels without being explicitly programmed [29]. In other words, the goal of machine learning algorithms for ASC is to learn a parametric model that maps a given acoustic feature into mutually exclusive categories.

Developing of acoustic models using machine learning algorithms for ASC heavily rely on data [2]. In practice, given the variability of acoustic scenes in real-life, there is a need for extensive amount of audio to be used for training an acoustic model. For the ASC task, each audio segment is accompanied by a reference annotation which refers to the environment in which the corresponding audio is recorded. As mentioned in Section 2.1.1, in

order to employ audio recordings in an machine learning algorithm, acoustic features are extracted from raw audio segment. More specifically, in the learning step of an acoustic model (Figure 2.2), the acoustic features are given as $X \in \mathbb{R}^{F \times T}$ where F represents the number of acoustic features in each frame and T represents the number of frames used in the STFT computation (explained in Section 2.1.1). In addition, the corresponding reference annotation $\mathbf{y}$ for each X (i.e. audio segment) is shown by a binary vector where the length of the vector C is equivalent to the number of target classes, i.e. $\mathbf{y} \in \{0,1\}^C$, in which only relevant class is represented by 1, while the rest of classes are represented by 0. In this step, the pairs of X and $\mathbf{y}$ are used to train the acoustic model in which for each X, model predicts an estimated output vector $\hat{\mathbf{y}} \in [0,1]^C$ which represents probability values for each target class. The class with highest probability value is considered as the predicted class label. Lastly, an error is measured between $\mathbf{y}$ and $\hat{\mathbf{y}}$ in order to adjust the parameters of the acoustic model. Once the learning step is completed, the learned acoustic model is deployed in the recognition step (Figure 2.2) to be evaluated for its performance on unseen X [29].

In general, any machine learning algorithm can be implemented in three different ways of learning: Supervised, semi-supervised, and unsupervised. In supervised learning, each sample of training data (i.e. data used in the learning step) contains a corresponding reference annotation. On the other hand, in semi-supervised learning some and in unsupervised learning none of the training data comes with reference annotations, or rather the reference annotations of training data are not used. Although semi-supervised and unsupervised learning can be applied as a way of learning for acoustic models in ASC, as explained above and according to [2], ASC is generally considered to be performed in supervised manner. In addition, most of the advances in ASC are made in supervised learning.

The earlier machine learning algorithm applied to ASC involved the usage of HMMs and GMMs which were inspired by the intensive usage of them in ASR. In [4], the authors developed a GMM model for each target acoustic scene with 32 components. This acoustic model was trained using expectation–maximization algorithm, and it extracted MFCCs from raw audio segments. In [30], the acoustic model is a SVM classifier using a radial basis function (RBF) as the kernel. In addition, Bisot et al. in [8] employ an NMF as the ASC system to reduce the cost in ASC. However, most of the recent approaches in addition to state-of-the-art results are based on the usage of artificial neural networks, specifically convolutional neural networks [10, 11].

2.2 Artificial Neural Networks

This section reviews artificial neural network (ANN) and its various architectures with a focus on convolutional neural networks and deep learning.

Recognition of a face or detection of a sound is an intuitive and routine task for most of the human beings, however, these tasks had not been successful for machines until

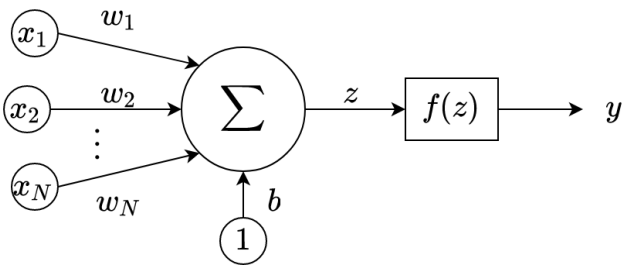


Figure 2.5. *The structure of one perceptron.*

recent years by progress in ANNs [15, 31, 32]. For many years, scientists searched for ways to make intelligent computers to think and predict instead of being hard-coded [33]. The first attempt which paved the way for the current state of ANNs was implemented by Warren McCulloch and Walter Pitts [34]. They published a paper in which they introduced a mathematical model of neural networks. In 1958, Inspired by biological neural networks, Frank Rosenblatt created perceptron (i.e. artificial neuron) with mathematical notations [35]. ANNs have been defined as a system of interconnected computing nodes by which they process the information using their dynamic state response to external inputs [36]. The architecture of a simple perceptron (Figure 2.5) is made up of a node, inputs, weights, and an output which simulate biological neuron components: cell nucleus, dendrites, synapse, and axon respectively [37]. A perceptron is formally defined as:

$$z = (\sum_{i=1}^{N} w_i x_i) + b\,, \text{ and} \tag{2.3a}$$

$$y = f(z) \tag{2.3b}$$

where for N number of inputs, the contribution of each input x_i to the output y is determined by a weight w_i. b is a bias term, and f is a function applied to the output of the Equation 2.3a. In ANNs, perceptrons are divided into separate groups called **layer**. The perceptrons of each layer receive the input form all perceptrons in the preceding layer, and their output is connected to all perceptrons in the succeeding layer. Figure 2.6 depicts a simple ANN consists of one input layer, one hidden layer, and one output layer [38]. The size of ANNs can be extended by augmenting the number of hidden layers or the number of perceptrons.

Equation 2.3a illustrates a linear operation for the computation of z. Most of real-world problems are not linearly separable, and without nonlinear functionality, ANNs are not able to solve real-world problems. Therefore, ANNs typically consist of nonlinear operations which are implemented using nonlinear functions at the output of perceptrons (i.e. f in Equation 2.3b). These nonlinear functions are usually called activation functions which enable ANNs with the capability of untangling and representing the existed nonlinearity patterns inside a dataset. They also provide ANNs with the potential for creating nonlinear decision boundaries. Below, commonly used activation functions are reviewed:

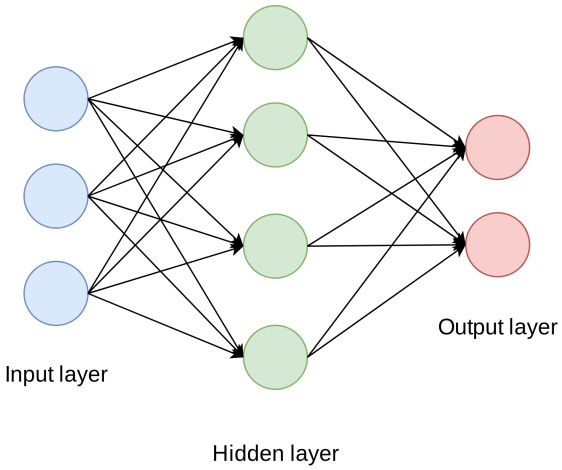


Figure 2.6. *The topology of an ANN with one hidden layer.*

Logistic function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

This activation function takes real-valued inputs and outputs values in the range of $(0, 1)$. Apart from non-linear characteristic of the logistic function (S–shaped curve in Figure 2.7), this function is monotonic and continuously differentiable. However, there are drawbacks which cause this function fall out of interest in recent ANN models. For example, the output of this function is not zero centered which makes the optimization of ANNs harder. More importantly, logistic function saturates in large negative or positive values meaning it responds very little to changes in large positive or negative inputs, resulting in a very slow optimization process. Logistic function is also called *sigmoid function*.

Hyperbolic tangent (TanH):

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.5}$$

Tanh function is nonlinear, and it is similar to sigmoid function. It can be counted as a rescaled version of the sigmoid function, $tanh(z) = 2\sigma(2z) - 1$. It squashes any real-valued number in the range of $(-1, 1)$, and in contrast to sigmoid, its output is centered at zero ($tanh(0) = 0$) which makes the optimization of an ANN model easier.

Rectified linear unit (ReLU):

$$ReLU(z) = max(0, z) = \begin{cases} 0 & \text{for } z \leq 0 \\ z & \text{for } z \geq 0 \end{cases} \tag{2.6}$$

ReLU has been the most popular nonlinear activation function in recent ANN models [39]. For negative values, ReLU outputs zero which inactivates its perceptron and increases

the sparse representation inside an ANN model. Whenever z (input) is positive, it acts as a linear unit meaning it outputs the value itself.

Softmax:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \text{ for } i = 1, \ldots, K \tag{2.7}$$

Softmax is generally used at the output layer of ANNs, specially for classification tasks. The above formula shows the calculation of softmax value for class i out of K target classes for vector z [40]. It normalizes the value for each class between 0 and 1 such that the sum of values for all classes is 1. In other words, the output of this function is interpreted as the probability distribution over all target classes [1].

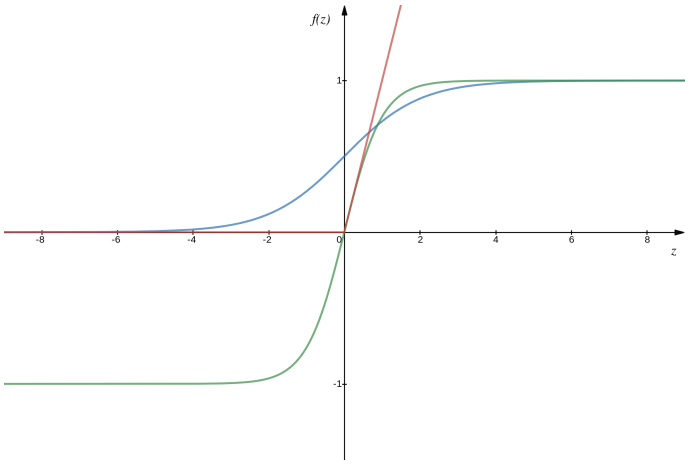


Figure 2.7. *Illustration of sigmoid (blue), TanH (green), and ReLU (red) activation functions. It is clearly visible that the sigmoid and TanH saturate for large negative and positive values. However, ReLU is linearly increasing for positive values in an unbounded fashion.*

2.2.1 Training Process of ANNs

Thus far, we have reviewed the basic architecture of ANNs and the means of expanding them. In this section, we study the way ANNs are trained in a supervised classification task.

The first step in training an ANN model is the initialization of its parameters (i.e. θ in Equation 2.8). This is done either by random values or using initialization methods such as He [41] and Xavier [42]. In addition, any classification task in supervised learning needs annotated dataset where each and every data sample is accompanied with one corresponding target annotation in case of single-label or multiple target annotations in case of multi-label classification tasks. The components of a sample of data are applied as the input layer (i.e. the first layer) of an ANN, and it is passed to the next layers as it is processed – meaning the output of each perceptron is computed as shown in Equation 2.3. Therefore, output and hidden layers of the ANN receive the data from their preceding layer in the network. The aim of this phase is to calculate the output values, i.e. prediction

probabilities, of each target class. This procedure of traversing the information from input to output in an ANN model is called **forward pass** [43].

Once the forward pass is completed, an error is computed using a comparison between reference annotations and predicted outputs by the ANN model. In other words, the error measures how far the predicted outputs are from the corresponding target annotations. The error function is usually addressed as objective function, loss function, or cost function [1]. A general formulation of a cost function is illustrated in Equation 2.8:

$$g\left(\theta\right) = \sum_{i=1}^{N} L(f(x_i;\theta), y_i) \tag{2.8}$$

where L is per-example cost function, $f(x_i;\theta)$ is the predicted output (i.e. $\hat{y}_i$) of the model for the input x_i, y_i is the corresponding target annotation for input x_i, and N is the number of samples used in the computation of the cost function g. Assuming the total number of training samples is M, the computation of g can be implemented using one sample (i.e. $N = 1$), a mini-batch of samples (i.e. $N < M$), or based on entire training data (i.e. $N = M$).

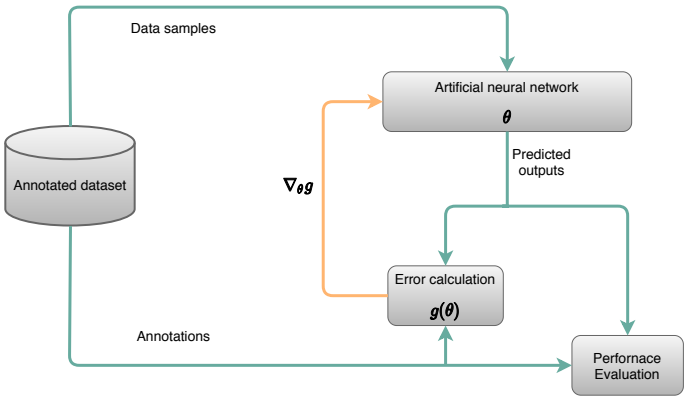


Figure 2.8. *The block diagram of training process for ANNs.*

One of the most commonly used cost functions for classification tasks is cross entropy [38]. It is a similarity measure of two probability distributions. In other words, cross entropy measures on average how many bits are needed to encode an event from an estimated distribution q by a model (e.g. ANN) instead of using the true distribution p [1].

$$H(p,q) = -\sum_{i} p_i \log(q_i) \tag{2.9}$$

In the context of ANNs, p can be replaced with target annotations, i.e. y, and q can be replaced with the predicted outputs of a model, i.e. $\hat{y}$. For example, in a one-hot encoded setting for a single-label multi-class classification task (with K number of target classes) where the output layer of an ANN involves a softmax activation function, the value of p is one for a single target class o, and zero for other classes. Therefore, the cross entropy

cost function for a single sample can be translated into

$$H = -\log(\frac{e^{\hat{y}_o}}{\sum_{j=1}^{K} e^{\hat{y}_j}}). \tag{2.10}$$

In order to improve the prediction power (i.e. performance) of an ANN model, it is required to minimize the error calculated by the cost function. To do so, the gradient of the cost function is calculated with respect to each parameter and propagated backwards using chain rule algorithm [44]. This phase is called **backward pass**. The backward pass in Figure 2.8 is indicated with yellow color. The completion of one forward and backward pass in called one *iteration*.

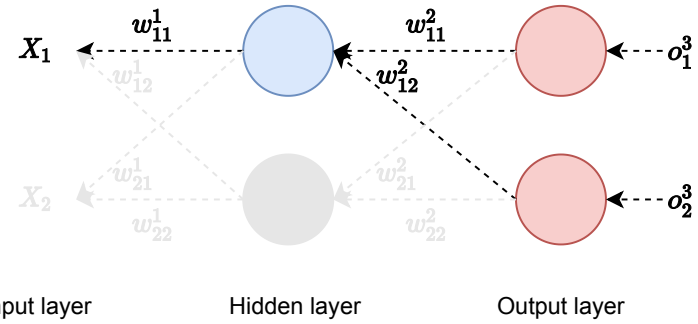


Figure 2.9. *The illustration for the backpropagation of error g with respect to parameter w_{11}^1. The bias terms are omitted for simplicity.*

Figure 2.9 is an example to present the computation of the error, i.e. g, backpropagation with respect to parameter w_{11}^1 in a simple ANN using chain rule algorithm. The parameters are denoted by w_{ij}^l formation where each w connects the ith node (or perceptron) of layer l to the jth perceptron of layer $l+1$. Considering Equation 2.3b, we define the output of activation function f for perceptron k from layer l as:

$$f(z_k^l) = o_k^l \tag{2.11}$$

As it is shown by Figure 2.9, the irrelevant paths from output to any point other than X_1 through the perceptron shown by the blue color in the hidden layer is inactive because they do not contribute to the backwards propagated value of the error with respect to parameter w_{11}^1. The value of this error backpropagation is associated with the error coming from two paths in the ANN. Each path is originated from one of the perceptrons in the output layer. Considering output o_1^3 and discarding the bias terms for simplicity, we calculate the chain rule algorithm backwards as:

$$\varphi_1 = \frac{\partial g}{\partial o_1^3}\frac{\partial o_1^3}{\partial z_1^3}\frac{\partial z_1^3}{\partial o_1^2}\frac{\partial o_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{11}^1}. \tag{2.12}$$

Doing the same for the path originated from output o_2^3 backwards to w_{11}^1, we obtain:

$$\varphi_2 = \frac{\partial g}{\partial o_2^3}\frac{\partial o_2^3}{\partial z_2^3}\frac{\partial z_2^3}{\partial o_1^2}\frac{\partial o_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{11}^1}. \tag{2.13}$$

Now that the backwards propagated error from both paths are calculated, we can derive the total error backpropagation of g with respect to parameter w_{11}^1 as:

$$\frac{\partial g}{\partial w_{11}^1} = \varphi_1 + \varphi_2. \tag{2.14}$$

The calculation of error propagation in backwards for other parameters in ANN shown in Figure 2.9 is calculated using the same concept presented by chain rule. The only difference is the paths involved for computation of each parameter differ from other parameters. In general, the same computational algorithm is used even for more complex ANNs regardless of the number of layers and perceptrons.

Once the gradients are estimated, the value of each parameter is updated using an optimization method in order to decrease the value of the error. In contrast to conventional optimization approaches, optimization is performed indirectly for ANNs. This means that in order to increase the performance of ANN models, it is required to reduce the value of cost function during the training process. In other words, we pay attention to the minimization of cost function with respect to indirectly maximizing the performance of the model [1]. In addition, due to the importance of the generalization performance in ANNs, the convergence criteria of an ANN model cannot be solely based upon the training set. Instead, early stopping technique is used to cease the training process whenever the performance of the model cannot be improved any further over a validation set of data[45]. In general, the problem of optimization for ANN models is unconstrained problem and non-convex unless some regularization techniques have been applied to the cost function as constraints [1].

One of the most utilized optimization algorithms for ANN models is **gradient descent**. This algorithm is an iterative and first-order optimization method meaning it updates the parameters of a model using only their first derivative. The three main variants of the gradient descent algorithm are batch, stochastic, and mini-batch [1].

Batch gradient descent computes the gradient of a cost function with respect to the parameters of the model based on entire training set. Although it computes the exact gradients, however, it can be very expensive and slow. It needs to fit all the training samples to a memory which is not always possible. In addition, it does not allow an online update of parameters when a new example added to the training set. In contrast, *stochastic gradient descent* (SGD) computes the gradient of a cost function based on each sample in the training set. Updating the parameters based on one sample, SGD allows us to update in an online manner. However, this method suffers from high variance for parameters update leading to a noisy cost function and keeps overshooting and making the

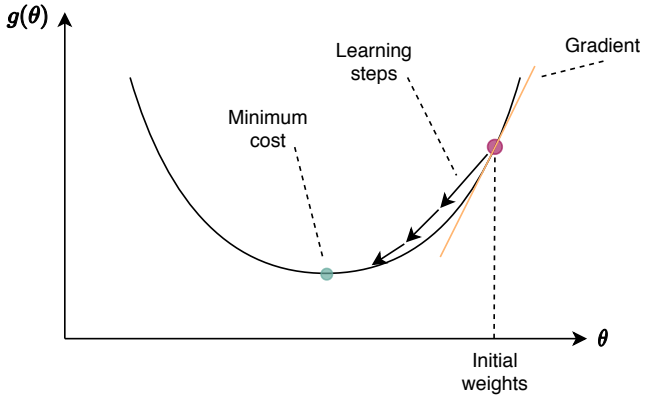


Figure 2.10. *An exemplar of gradient descent algorithm depicting how the algorithm utilizes the first derivative of the cost function to move towards the minimum cost.*

convergence to the exact minimum point very difficult. Therefore, stability is an issue of this gradient descent variant.

Minibatch gradient descent uses the best of two previous techniques and updates the parameters based on small batches of training samples (i.e. minibatch) each time. By doing so, it increases the stability of convergence due to the less variance of parameter updates. However, the computed gradient of cost function g using this technique is an estimation of the exact gradient calculated with batch gradient descent. This is because, in minibatch gradient descent, the gradient of cost function g is computed using a small number of samples (i.e. a minibatch) instead of entire training dataset. This integrates a noise into the optimization process which is present even at the resulted local minimum [1]. Moreover, the samples in each minibatch should be selected randomly and independently from each other to prevent the aggravation of integrated noise – by sampling only from a few target classes instead of all – in the estimated gradient value [1]. Considering the cost function $g(\theta)$ introduced in Equation 2.8, minibatch gradient descent updates the parameters (i.e. θ) of ANNs by the following formula:

$$\theta \leftarrow \theta - \eta_k \nabla_\theta g \tag{2.15}$$

where $\nabla_\theta g$ is the gradient of cost function g with respect to parameters θ, and η_k is a learning rate at iteration k.

As it is shown in the Equation 2.15, learning rate η is a hyperparameter and a function of the iteration variable k. The adopted value of this parameter plays an important role in the optimization of ANNs. If it is set too high (e.g. > 1.0), we may diverge and overshoot the optimum value (desired local minimum). However, if it is too low (e.g. $< 1e{-}6$), the time of convergence will unnecessarily take long, and it may not converge to a local minimum [1]. We practically call for decaying the learning rate over time in minibatch gradient descent, and it is why we consider learning rate at iteration k despite the possibility for retaining a fixed learning rate in batch gradient descent. According to [1], choosing the right value for the learning rate should be based on try and error, and it is common practice to decrease

initial learning rate η_0 until some iteration τ (η_τ is the value of learning rate at iteration τ), and after that fix it to a constant value:

$$\eta_k = (1-\alpha)\eta_0 + \alpha\eta_\tau, \text{ and} \tag{2.16}$$

$$\alpha = \frac{k}{\tau} \tag{2.17}$$

As mentioned above, calculation of gradient based on a mini-batch of data (and not all training samples) is an approximate of the exact gradient and, therefore, noisy [1, 46]. This slows down the optimization process. **Momentum** is an enhancing term integrated into the basic minibatch gradient descent algorithm in order to help the problem of slow-moving pace in optimization procedure [44]. Momentum accelerates the optimization process of ANNs by steering the gradient in the right direction and alleviating oscillations in irrelevant directions. In other words, it leads to faster and stable convergence and reduces the oscillations of gradient descent algorithm due to noises in the data. To do so, momentum uses an exponentially moving average of past gradients to keep track of their directions [47]. Doing so, more number of training samples are involved in the calculation of momentum and subsequently updating parameters. In comparison to the update rule of minibatch gradient descent in Equation 2.15, There is one extra step for updating the momentum [1]:

$$\upsilon \leftarrow \alpha\upsilon - \eta\nabla_\theta\Big(\frac{1}{N}\sum_{i=1}^{N} L(f(x_i;\theta), y_i)\Big), \text{ and} \tag{2.18}$$

$$\theta \leftarrow \theta + \upsilon \tag{2.19}$$

Where υ is the velocity vector (i.e. momentum) that keeps track of previous gradients and α is a hyperparameter ($\alpha \in [0, 1)$) which shows the degree of impact of previous gradients on the direction of the current one. In contrast to physics which defines the momentum as a multiplication of mass and velocity, in learning algorithms, mass is considered to be unit. Therefore, momentum is equivalent to velocity [1].

Optimization Algorithms with Adaptive Learning Rates

Finding the optimal value for the learning rate is not a simple task. The value of this hyperparameter can considerably affect the performance of ANN models. In addition, the cost of optimization is not equally sensitive for all directions in the parameter space. Therefore, it reasonably suggests to update each parameter with a different learning rate. Despite the fact that momentum reduces the oscillation of gradient towards irrelevant directions to a certain degree, it introduces another hyperparameter to the optimization algorithm. The advent of optimization algorithms with adaptive learning rates is to solve this issue [1].

Adam optimizer: This algorithm [48] is one of the most popular optimization methods with adaptive learning rate. This method employs the moving average of past gradients

and squared gradients. They can be referred to as the first and second order moments of past gradients which indicate the emergence of the name Adam which is derived from adaptive moments. Adam also solves the issue of initialization bias in an earlier version of optimization methods with adaptive learning rates [49]. As shown in Algorithm 1, the first and second moment variables are initialized by vectors of zero. In addition, initial values of decay rates (i.e. ρ_1 and ρ_2) are close to 1 for the moment estimates in the initial steps of optimization process which lead to biased estimates of moment variables towards zero, and force for larger initial steps in optimization [47, 48]. This is resolved in Adam by correcting the bias in the values of moment estimates before updating them (Algorithm 1).

Algorithm 1 The Adam algorithm. The algorithm is adopted from [1].

Require: Learning rate η
Require: Initialize ρ_1 and ρ_2 in $[0, 1)$.
Require: Small constant δ used for numerical stabilization.
Require: Initialize:
  Parameters θ
  1st and 2nd moment variables: $s = 0, r = 0$
  Time step $t = 0$
  while stopping criterion not met **do**
    $\{(x^1, y^1), (x^2, y^2), \ldots, (x^N, y^N)\}$ ▷ Sample a minibatch from the training set
    $g' \leftarrow \frac{1}{N} \nabla_\theta \Sigma_i L(f(x_i; \theta), y_i)$ ▷ Compute gradient estimate
    $t \leftarrow t + 1$
    $s \leftarrow \rho_1 s + (1 - \rho_1) g'$ ▷ Update biased first moment
    $r \leftarrow \rho_2 r + (1 - \rho_2) g' \odot g'$ ▷ Update biased second moment
    $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$ ▷ Correct bias in first moment
    $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$ ▷ Correct bias in second moment
    $\theta \leftarrow \theta - \eta \frac{\hat{s}}{\sqrt{\hat{r} + \delta}}$ ▷ Update parameters
  end while

Challenges in the optimization of ANNs

The optimization of ANNs is a very challenging task regardless of the used optimization algorithm. An objective function which is a convex function of ANN parameters provides a global minimum since the local minimum of the function is considered the global minimum [1]. However, a non-convex function of ANN parameters can contain a large number of local minima with large values, and settling in these points may prevent the optimization process from further reduction of the error which results in an underfitted ANN model. Despite this issue, Goodfellow et al. in [1] state that for adequately large ANNs, pragmatically there are many local minima with low cost, Therefore, this cannot be a practical problem in the optimization of ANNs. However, settling in saddle points can hinder the optimization algorithms from further reduction of the error. Saddle points are critical points placed on the surface of a function where its derivative is zero but it is counted neither a local minimum nor a local maximum [50]. Goodfellow et al. express that saddle points are more problematic for optimization algorithms since these points are extensively ubiquitous in higher dimensional spaces compared to local minima. Lastly, the problem of

exploding gradients is another challenge in the optimization of ANNs. This issue occurs when the update value for parameters changes drastically and becomes very large. This can be a result of multiple multiplications of parameters with large values. This issue can be solved using gradient clipping such that any large gradient update which leads to the update of parameters with large amount is diminished to a predefined value [1].

2.2.2 Deep Neural Networks

Frank Rosenblatt created the perceptron decades ago [35], however, ANNs would not gain as the popularity of current time for many years thereafter [51]. One of the breakthroughs which can be counted as a milestone for the popularity of ANNs is the deployment of backpropagation method for adjusting the parameters of ANN models [51]. Although the concept of backpropagation was first introduced in control theory [52], it was employed for the usage in training ANN models in 1986 by Rumelhart et al. [44]. Apart from this advancement, There was a need for faster processors than central processing units (CPUs). Therefore, the utilization of graphics processing units (GPUs) played an important role in the progress of ANNs as well [1]. In addition, ANNs are data-driven models which heavily rely on data to learn the true representation of their inputs. The lack of annotated data in many tasks as well as hardware limitations, i.e. low capacity memories, hindered the progress of ANNs [1, 16]. Solving these issues opened a new era in the field of artificial intelligence (AI). Since then ANN models have solved previously impossible tasks for machines such as image classification and speech recognition. The more complicated a task has been, the more complex ANN models have become [1]. This has been achieved by increasing the number of hidden layers, and the number of perceptrons in each layer. These complex models have been referred to as *deep neural network* (DNN) or *deep learning*.

As the number of hidden layers increases in DNNs, their potentiality to outperform previous shallow networks escalates as well. However, this causes DNNs susceptible to overfitting to the training data [53, 54]. As a result of overfitting, DNNs perfectly perform on the training set, however, they are not able to generalize well for unseen data, hence it raises the generalization error. This is due to the large number of parameters in DNNs and their exceptional ability to perfectly fit to a set of data.

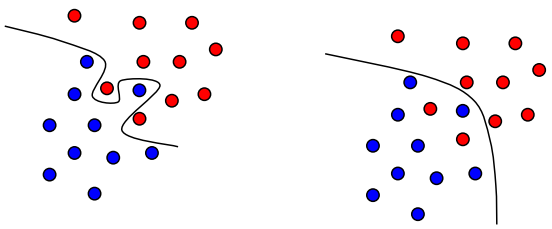


Figure 2.11. *the left side image shows a complex model which perfectly separates two classes of data in training set, while the right side model is much simpler in terms of complexity and shows misclassification of few points in a training set. However, the right side model may generalize better for unseen data. The figure is adopted from [55].*

Regularization plays an important key in the training of DNNs to prevent this issue [1]. Regularization is a technique applied to learning algorithms in order to decrease the generalization error. To compensate for overfitting, some regularization techniques have been introduced to cope with the overfitting problem, and to assist DNNs in training phase to capture more generalized features of data in order to motivate the performance of DNNs for unseen data as well.

In large neural networks where all their parameters are learned together, it is common for the parameters of DNNs to present an asymmetric pattern in terms of predictive capability. This means that some of the parameters have more predictive power compared to other ones. After training DNNs for large number of iterations, this phenomenon becomes more prominent such that only a small number of parameters are trained and the rest are disregarded and left with less predictive power. This problem, which is called Co-adaptation, limits the predictive capability of DNNs. One of the well-studied techniques to prevent this problem is **dropout** [56]. Dropout randomly eliminates perceptrons and their connections which results in various lighter networks for each iteration. This prevents training only with a fraction of parameters, and stimulates the usage of all available parameters in DNNs in order to boost the predictive capacity of DNNs.

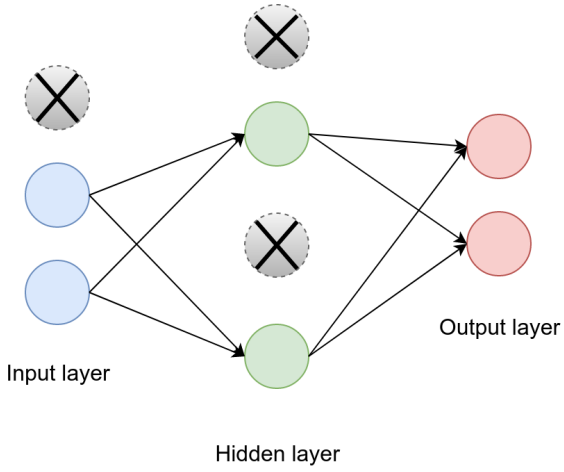


Figure 2.12. *An illustration of dropout technique applied to a simple ANN shown in the Figure 2.6. The marked perceptrons with "X" are the eliminated perceptrons, and they are disconnected from the perceptrons in preceding and succeeding layers.*

Another method that assists DNNs in easier and faster optimization is **batch normalization** [57]. It is a common practice in machine learning to normalize the input data if the values of different features are at contrasting scales, because it increases the speed of convergence in optimization algorithms for DNNs during training process [58]. The batch normalization addresses the same issue inside DNNs which is called *internal covariate shift* (covariate shift will be discussed in details in Section 2.3.1). After each iteration and updating the parameters of a DNN, its parameters change accordingly. The change in the parameters of an internal layer causes the distribution of inputs to the next layer changes as well. This results in a slower optimization and significantly harder training process. To

solve this issue, batch normalization introduces the concept of normalization to inputs of each internal layer. To do so, instead of hardcoding the normalization so that the values are normalized between [0, 1], it inserts two normalization parameters, i.e. γ and β, into the set of original parameters of DNNs to scale and shift the normalized output values for each mini-batch of data. Doing so, it allows the parameters to be learned by back-propagation process. It has been shown to speed up the training process much faster. In addition, it allows for the choice of higher learning rates during optimization process [57].

2.2.3 Feedforward Neural Networks

Feedforward neural network (FNN) is a network of perceptrons in which there is no internal feedback, and the flow of the information signal is headed from perceptrons in the input to perceptrons in the output layer [59, 60]. Similar to other ANNs architecture the goal of the FNNs is to estimate a function that maps inputs to an desirable output using a set of parameters (θ).

The simplest version of FNNs is *single-layer perceptron* where the input and output layers are the same. The early design of these versions used a linear threshold unit as the activation function. In 1969, Minsky and Papert [61] showed that a single-layer perceptron is not capable of learning XOR function. However, they did not deny this possibility for a multi-layer perceptron network.

The more recent version of FNNs is the multi-layer perceptron (MLP) (illustrated in Figure 2.6). In contrast to a single-layer perceptron, an MLP with a hidden layer is capable of approximating any bounded measurable function to any arbitrary degree [62]. In practice, not one perceptron, but the stacked version of them in different number of layers shapes the final MLPs. Except from the input and output layers, the number of hidden layers can be from one to as many as desired [62]. Any given perceptron in a layer is connected to all perceptrons of the next layer. The number of perceptrons in the output layer depends on the number of categories of interest in the classification task. In general, the number of hidden layers in ANNs can be an indicator of capability of the network to model complex representations [62]. The more number of hidden layers is equivalent to greater ability of the network in expressing higher representation of input data [54]. This ability of the network for extracting higher and more complex representation of input data eases the need for preprocessing and feature engineering techniques, and allows the network to use lower-level representation of data as inputs.

2.2.4 Convolutional Neural Networks

Convolutional neural network (CNN) has been one of the significant milestones in the development and deployment of DNNs in different applications. Having been introduced decades ago to the field of machine learning [63, 64], CNN is still a constituent element of the networks used in recent state-of-the-art results for challenging tasks specially image

classification [16, 54]. The topology of CNNs allow for receiving inputs of 1D to 3D dimensions [1]. As it can be denoted by the name of these networks, the main operation of CNNs is a linear transformation, namely **convolution**. Multiple characteristics of CNNs have made them a better practical solution compared to MLP networks:

- The output of CNNs is dependent on the input size, kernel size, zero padding, and strides. Whereas, the output of MLPs is independent of the input size [65].
- In contrast to MLPs, CNNs leverage the ability of utilizing spatially correlated features of small regions in inputs, this means that every point in the output of a CNN is affected by a small region of the input called *receptive field*.
- Due to the shared parameters of a kernel that slide on input, CNNs are capable of cutting down many unnecessary parameters while keeping the relevant information for the task of interest. This also reduces the memory usage due to the storage of less number of parameters. Subsequently, pooling layers (will be discussed later in this section) are advantageous to reduction in the number of linear transformations in convolutional layers by shrinking the size of the data.
- In contrast to the restriction in MLPs in which inputs are shuffled, flattened, and the order of input shape has to be changed, CNNs are able to receive inputs in their original shape and dimension.

The discrete convolutional operation of 2D CNNs consisting of K_o kernels $K \in \mathbb{R}^{K_c \times K_h \times K_w}$ – where K_c is the number of input channels, and subscripts h and w respectively refer to the height and width – for an input $I \in \mathbb{R}^{K_c \times I_h \times I_w}$, and with unit stride and zero padding (will be discussed later in this section) computes output $O \in \mathbb{R}^{K_o \times O_h \times O_w}$ for each kernel K as:

$$O_{k_o, o_h - K_h, o_w - K_w} = \sum_{k_c=1}^{K_c} \sum_{k_h=1}^{K_h} \sum_{k_w=1}^{K_w} I_{k_c, i_h - k_h, i_w - k_w} K_{k_o, k_h, k_w} \tag{2.20}$$

In CNNs, channel refers to a third dimension in inputs which corresponds to depth, and due to the popularity of CNNs in computer vision applications, the term channels has been adopted by the majority in this field. For example, a black and white image consists of one channel. Whereas, the pixels of an RGB image contain values for three channels of red, green, and blue.

According to Equation 2.20, the points (i.e. pixels) of a small and localized region of the input feature map (i.e. I) are multiplied by the points of the kernel (i.e. K), also known as filter. The size of kernel is typically smaller than the input size. Therefore, we slide the kernel over the input in order to calculate the dot products of the kernel and the receptive field of the input which is covered by the kernel. To clarify the calculation in convolutional layers, we provide an example of convolutional operation in CNNs. Figure 2.13 demonstrates the values of an input feature map and kernel. In this example, since the number of channels is assumed to be one, the input feature map and the kernel are

of 2D dimensional shape.

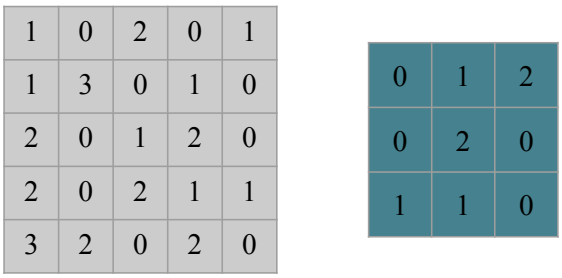


Figure 2.13. *The left table (grey color) shows an input feature map of size* (5×5)*, and the right table presents the values of a* (3×3) *kernel.*

In Figure 2.14, the green highlighted region shows the elements of receptive field on the input feature map which are multiplied element-wise by the kernel values. In order to compute the output feature map (i.e. O) of convolution, the results of multiplications are summed up to compute the final value in the output feature map at each location which is illustrated with red color. In practice, after summation, a bias value will be added to find the final output value which is omitted in this example for simplicity.

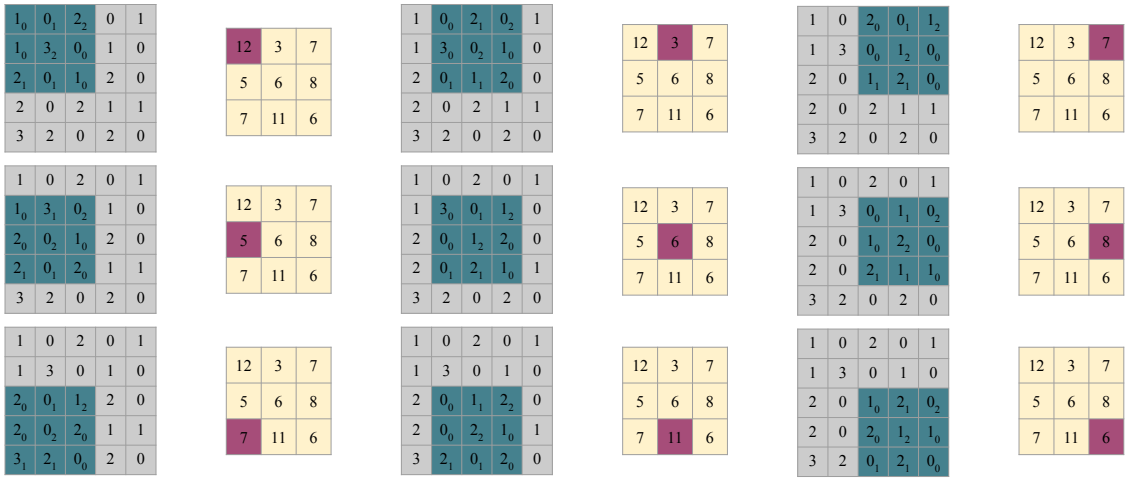


Figure 2.14. *Computing the output values of a discrete convolution between the input and the kernel presented in Figure 2.13*

In practice, The number of output feature maps of one convolutional layer is equivalent to the number of used kernels. In addition, the number of channels of each kernel equals to the number of channels in input feature maps. Therefore, in Figure 2.14, using one kernel results in one output feature map (i.e. yellow window).

Strides can be viewed as a method of subsampling in convolutional layers. Strides refer to the distance between two consecutive positions of a kernel along each axis [65]. We can skip some of the positions of input feature maps when performing convolution using strides greater than one. This shrinks the number of performed computations, at the expense of loosing some details at the representation of output feature maps. Figure 2.14 illustrates 1×1 strides in convolutional operation.

Zero padding is the process of adding zeros at the boundaries of an input feature map along an axis before convolutional operation. As depicted in Figure 2.14, convolutional operation without any zero padding shrink the size of output feature map. This shrinkage occurs at each convolutional layer put a constraint over the demonstrative ability of CNNs. In other words, the number of convolutional layers, which could be utilized in an CNN, is automatically limited. This occurs because adding convolutional layers shrinks the dimension of output feature maps, and at some point, it will be of dimension 1×1 on which convolutional operation is not considered logical. In addition, zero padding can prevent losing information in the boundaries of input feature maps. Mathematically, each convolutional layer without zero padding as well as unit stride shrinks the dimension of output feature maps by:

$$W_O = W_I - W_K + 1 \tag{2.21}$$

where W_O, W_I, and W_K refers to the width of an output, input, and kernel respectively. Equation 2.21 implies that this limitation in capacity of CNNs is more perceptible as the size of kernels gets larger. This type of convolution without zero padding is well-known as ***valid*** convolution. To solve this problem, another method for implementation of convolutional layer is used in which the input feature map is padded with a number of zeros in each axis in order to obtain an output size of the same size as in the input feature map. This convolution is referred to as ***same*** convolution. However, in this convolution, the points (i.e. pixels) near to the edges of inputs are underrepresented in the output feature map compared to the central points in the input feature map. The third form is called ***full*** convolution. This type of convolution addresses the problem noted in same convolution. In full convolution, each point in the input feature map is visited by kernel the same number of times to represent the same potency of points irrespective to their position in the input feature map [1, 65]. A downside to this method is that learning a kernel which perfectly extracts meaningful features at all positions of the input is not trivial [2].

Pooling is a practical technique used as a building block in CNN to make the output of convolutional layers robust to small changes in inputs [66]. A pooling layer is typically placed between two consecutive convolutional layers. Similar to a convolutional layer, it employs a small window to slide over a localized rectangular neighbourhood of its input, but instead of using a linear combination to compute the output value in each window [67], it calculates either the average or max value of that rectangular region. Average and max pooling are the most common used methods in the implementation of pooling layers. In some tasks, the pooling layer performs only on one axis due to the necessity of transferring the information on one axis of feature maps to the output [68]. An example of this can be found in sound event detection task (SED) in which pooling cannot be performed on time axis of input feature maps otherwise the information regarding onset and offset of sound events happening in a sound clip is discarded. The main leverage of using pooling layer is to make the output of convolutional layers invariant to small changes (e.g. noises) in input [1]. In other words, pooling layers pay attention to extracting certain features

and ignoring the exact position of them in feature maps. As a consequence of pooling operation, the size of output feature maps are shrunk each time causing the number of computations needed in the following layers of a CNN less cumbersome resulting in the enhancement of efficiency in memory usage for storing parameters of the network.

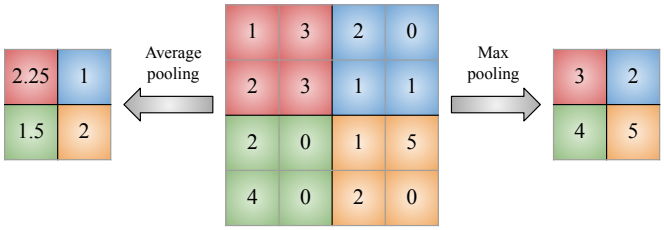


Figure 2.15. *An example of two popular variants of pooling operations, i.e. max and average pooling, with input size of* (4×4)*, kernel size of* (2×2)*, and strides of size 2 on both width and height axes.*

A deep CNN is typically built of stacking blocks of simple operational layers. The simplest convolutional block is mainly comprised of three operational layers: convolution, non-linearity, pooling layers.

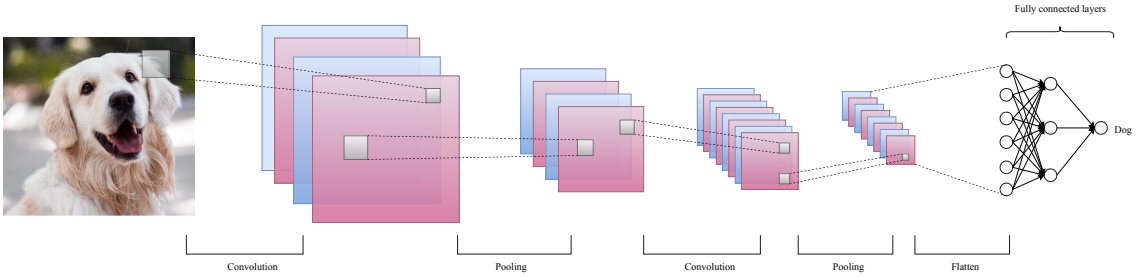


Figure 2.16. *A convolutional neural network followed by fully connected layers for image classification task.*

Figure 2.16 presents an example of a straightforward CNN. The output of a convolutional layer is typically passed through a non-linearity function such as ReLU in hidden layers and softmax in the output layer (This stage is skipped in Figure 2.16 for simplicity). It is also possible to use less number of pooling layers compared to the number of convolutional layers in a network. In practice for a general classification task, we usually add fully connected layers at the end of convolutional blocks to calculate the probability involved with each class given the input data.

2.3 Domain Discrepancy

Domain discrepancy is one of the main concerns of machine learning algorithms. This section first introduces the causing factors for domain discrepancy, i.e. dataset shift. Then, it presents the domain adaptation methods to cope with the dataset shift problem by reviewing main recent works in this field.

2.3.1 Dataset Shift

Having learned a representation for a set of data samples, a model is able to automatically predict desirable variables in a given task. Considering the performance of the model in the development phase, we expect the model to perform relatively similar in the evaluation phase. This is the presumption in designing conventional machine learning algorithms. Although it could be correct when the training and test datasets are sampled from the same underlying distribution, it does not examine a more quintessential scenario of a real-world problem where there is no guarantee for such assumptions [14]. In a real-case application of ASC, we generally deal with situations in which there is a large variability in the acoustic environments, recording devices, position of recorders, etc. These various conditions affect the data that we access to in the training stage and the data we encounter in the testing stage. In addition, this problem is not limited to the ASC task, and it influences the performance of machine learning algorithms in different applications. For example, in image classification task, different camera angles and lightening intensities can be counted as the variable conditions causing reduction of performance in this application. Consequently, this problem results in a performance gap between training and test stages so that the evaluation performance of a model on a test set is substantially lower than that in the training stage. It is not often possible to know all the various conditions which might be faced with during the testing phase. In addition, annotating enough samples from the distribution of test set can be too costly. *Domain adaptation* refers to the methods by which we try to propose a systematic solution for this problem. According to [14], a domain is formally defined as a pair of distribution $\mathcal{D}$, from which input data X are sampled, as well as a labeling function f [14]. In fact, domain adaptation focuses more on two domains of data, and on boosting the predictive performance of a model in one domain given the data in another one [13]. In the context of domain adaptation, the data which is originally used to optimize the model is called **source domain** which is equivalent to training data, while the data used in order for the model to adapt to is called **target domain**, and is analogous to test data.

Table 2.1. *Connection between the settings used in conventional machine learning versus domain adaptation. The table is adopted from [69].*

Learning Paradigms	Source and Target Domains	Source and Target Tasks
Conventional machine learning	The same	The same
Domain adaptation	Different but related	The same

As mentioned above, dataset shift refers to the situation when the joint distributions of inputs and outputs for training and test sets differ. Formally, for any given sample $x \in X$ and the corresponding class label $f(x) = y$:

$$p_{tr}(y, x) \neq p_{te}(y, x) \tag{2.22}$$

However, this change can be categorized differently based on the type of shift among datasets. Followings are three of the most common types of dataset shift in machine learning problems:

Covariate shift has been first defined as the case in which only the distribution of input attributes x in training and test sets are different, i.e. $p_{tr}(x) \neq p_{te}(x)$ [70]. Despite the fact that this definition has been also reiterated in different works such as in [71, 72, 73, 74], other works such as [75] define covariate shift problem as a change of the distribution of class labels between training and test datasets. Therefore, machine learning community has yet to find a consensus and a general agreement on the standard definition used for this problem.

Prior probability shift is the inverse problem of covariate shift where the distribution of class label variables $p(y)$ changes, rather than $p(x)$, between training and test datasets [74, 75, 76]. In other words, there is a change in the prevalence of class labels distribution between training and test sets [77].

Concept shift is a more complex problem compared to the above two problems. Concept shift is also referred to as concept drift in literature [78]. As a by-product of non-stationary and dynamically changing environments, this phenomenon happens when the relationship between input and output variables changes over time [79]. One common example of this problem can be seen in non-stationary time series analysis where the relationship between inputs and predicted variables changes according to time. In other words, training and test data have been collected in different time. According to [73], this problem is formally defined as:

$$\text{if } x \rightarrow y,\ p_{tr}(y \mid x) \neq p_{te}(y \mid x) \tag{2.23}$$

$$\text{if } y \rightarrow x,\ p_{tr}(x \mid y) \neq p_{te}(x \mid y) \tag{2.24}$$

Aforementioned types of dataset shift occur due to various factors. There are multiple causing factors of such variability among datasets, but the most two common factors are: *sample selection bias* and *non-stationary environments*. Sample selection bias is one of the most prevalent factors causing the dataset shift problem. It can be created in data collection phase where various reasons can involve to prevent an uniform sampling of a population for training samples. As a result of that, a model, trained by a biased dataset, does not fairly model all different involved categories existed in the population. A survey can be a simple instance to show this issue where accessibility to some of the people is easier than others, and ending in the over-representation of those people in the result of survey, while the opinions of other peoples have been under-represented in the result [73, 80]. Moreover, non-stationary environments can also lead to different types of dataset shift problem. In real-world applications, this problem causes the advent of new operating conditions or shift in the process of data generation which results in temporal or spatial shift of data between the development and evaluation stages of a

model. As a result, models, which are trained under the assumption of stationary property of environment and fixed probabilistic properties of data, will become obsolete and are not able to optimally perform for new captured data [81, 82].

2.3.2 Domain Adaptation Methods

This section presents the advantages of using domain adaptation techniques. Then, it provides the formula for upper boundary generalization error in the presence of dataset shift. Lastly, it reviews different types of domain adaptation techniques in addition to main recent works.

As a result of dataset shift, conventional learning algorithms optimised on source domain data, are not able to perform well on the target domain. Consequently, there is a gap between the evaluation results of these two domains. In order to alleviate this issue and reduce the gap in evaluation results, domain adaptation methods employ different techniques to adapt the already optimized model on source domain to data from target domain. Furthermore, as the size of datasets collected in the real world is extensively increasing, annotating such datasets is time-consuming and expensive work, and in some applications impossible to do. However, creating synthetic data for most of applications not only is easier, but also the annotation of such data can be done automatically with far less amount of efforts and dedication of time in comparison to a real-world dataset. Learning algorithms can benefit from synthetic data in order to learn the inherent patterns inside the data for a specific task. Despite that, synthetic data often lack diversity and noises that a real-world sample may represent. Therefore, this causes a dataset shift if a learned model by synthetic data is tested on real-world datasets. One of the significance of domain adaptation is to solve this problem to make the usage of synthetic data possible as a leverage in learning process.

Ben-david et al. formalized the problem of domain adaptation using the following upper boundary generalization error on target domain [14]. For a binary classification task with a source domain $< D_S, f_S >$ and target domain $< D_T, f_T >$ – where D refers to the distribution of data and f to a labeling function – the upper boundary generalization error is defined as:

$$\varepsilon_T(h) \leq \varepsilon_S(h) + d(D_S, D_T) + min\{\mathbb{E}_{D_S}[\mid f_S - f_T \mid], \mathbb{E}_{D_T}[\mid f_S - f_T \mid]\} \tag{2.25}$$

According to the equation 2.25, for a hypothesis h, the error on the target domain $\varepsilon_T(h)$ is bounded with the sum of three terms. the first term is the risk of h on the source domain data, i.e. $\varepsilon_S(h)$. The second term signifies the distance between the marginal distributions of the source and target domain data. Lastly, the third term measures the difference between the labeling functions of the source and target domains. There has been a common belief in domain adaptation community that given a minimal labeling function differences between the source and target domain, minimizing the risk on the

source domain together with the minimization of distance between marginal distributions by a learnt invariant representation is sufficient to guarantee adaptation performance on the target domain, however, Zhao et al. in [83] showed the insufficiency of this assumption by an counterexample, and exhibited that we need to take into consideration the change of the conditional distributions between the source and target domains.

The settings adopted for domain adaptation methods mainly depend on the application of interest and availability of various data sources. In general, we usually have large number of labeled samples from the source domain, while there is no or few labeled samples from the target domain. In this regard, domain adaptation approaches are considered either semi-supervised or unsupervised. Unsupervised domain adaptation addresses the problem of dataset shift when no labeled data from target domain is available [84, 85]. On the other hand, in semi-supervised domain adaptation, few labeled data is available from target domain [86, 87]. Domain adaptation can be also performed when the source domain is comprised of multiple datasets, named *multi-source domain adaptation*. In this setting, the aim is to combine the learned hypotheses for each source to obtain a hypothesis with small error on target domain [88, 89].

Although the domain adaptation techniques have been categorized with different names in recent works [90, 91, 92, 93], we review the most well-known categories including three general forms of **discrepancy-based**, **adversarial-based**, and **reconstruction-based** approaches in this thesis.

Discrepancy-based approaches focus on fine tuning of DNNs using the data from the source or the target domain in order to reduce the dataset shift. This can be implemented in various ways. One is done using a pre-trained DNN and only training some layers of the network while freezing the first n layers. The second set of approaches assigns pseudo-label targets to samples of target domain in order to incorporate these samples in the learning process alongside the data from source domain. However, more works in this category focus on finding an invariant-domain representation of data by reducing the discrepancy between source and target distributions [94, 95, 96]. In this regards, maximum mean discrepancy (MMD) and Kullback–Leibler (KL) divergence have been extensively used as a measure of discrepancy. For example, Tzeng et al. [97] incorporate the MMD as a domain confusion cost function into the optimization process of DNNs to investigate the effect of MMD on latent space representations produced by DNNs. Formally, the MMD distance with respect to the DNNs higher representations, i.e. the latent representations of preceding output layer $\phi(.)$, is defined as

$$d_{MMD}(X^S, X^T) = \left\| \frac{1}{M} \sum_{m=1}^{M} \phi(x_m^S) - \frac{1}{N} \sum_{n=1}^{N} \phi(x_n^T) \right\|, \tag{2.26}$$

where x^S belongs to the source and x^T belongs to the target domain. They employ two CNN-based architectures with shared parameters. One is fed by the source domain data (X^S, Y^S), and the input to the other one is the data from the target domain. The

former network is optimized in a supervised manner using the data and corresponding annotations from the source domain. They use an adaptation bottleneck layer before the final layer of the classifier in order to regularize the output of this layer using domain-invariant loss function. In addition, to make the latent features of DNNs semantically separable for actual classification task, the network features a classification loss, i.e. $\mathcal{L}_C$. Therefore, the optimization is done based on the final error calculated using Equation 2.27 to improve the performance of image classification task over the target domain data:

$$\mathcal{L} = \mathcal{L}_C + \lambda \, d^2_{MMD} \tag{2.27}$$

Inspired by generative adversarial nets (GANs) [98], the aim of adversarial-based methods is to maximize the domain confusion (i.e. source and target) for a domain discriminator by achieving an invariant representation of the source and target domain data [84, 99, 100]. In general, different adversarial-based methods vary in terms of three components. First, the implementation of the base model can be done in generative or discriminative manner. Secondly, adversarial loss function is a minimax function in which the previous methods tried to reduce its value by optimizing the discriminator while maximize it by optimizing the generator (or a model in discriminative problem). The direct implementation of this loss function creates vanishing gradient problems in early stages of optimization as the discriminator tends to converge far quicker than the generator, and this damages the learning process. To overcome this issue, recent works tried to implement the optimization using two different loss functions. One of them is dedicated to the optimization of discriminator, while the other one is to optimize the generator. The third difference is the usage of shared, unshared, or partially shared weights for models producing feature mappings from the source and target domains during learning process. Ganin et al. proposed one of the first domain adaptation methods using adversarial-based training [84]. In this work which is called domain-adversarial neural network (DANN), they use a common feature extractor together with two classifiers. One classifier is to do the actual classification task based upon the latent representations of the source domain data provided by a feature extractor. The other classifier plays a role similar to the discriminator in GANs. In other words, this classifier classifies whether the data comes from the source or target domain. The aim of the classifier is to minimize the domain confusion, while there is a gradient reversal layer (GRL) which is responsible for the alignment of latent feature mappings of feature extractor, hence the alignment for the distributions of the source and target domains.

In reconstruction-based approaches, the invariant feature space is obtained by reconstructing the features of the source and target datasets. This method has been usually implemented using an adversarial or encoder-decoder training process. In adversarial approach, the aim is to create mappings using a generator to produce an indistinguishable representation of a domain compared to another one using an adversarial loss and discriminator [101, 102]. In addition, the framework of encoding and decoding of representations for domain adaptation is inspired by autoencoders [103, 104]. This approach

adds a supplementary task to the main task in order to assure a shared subspace between the source and target domain. The encoding part tries to obtain an robust and invariant higher-level representation of both source and target domains from which it is possible to reconstruct the original input data using a decoder network [105]. This can be done by one step or two-level reconstruction. Chen et al. achieved a robust reconstruction of input data in a closed form solution and without the need to gradient descent optimization [106]. Bousmalis et al. proposed an encoding-decoding based solution using DNNs [107]. This method consists of one shared encoder between domains, two domain exclusive encoders, one shared decoder, and one classifier. The shared encoder learns an invariant representations for the samples of both domains, while domain exclusive encoders are to capture the unique representations for data of each domain. The shared decoder learns to reconstruct a sample using the representation of both shared and domain-specific encoders. The aim is to enhance the similarity of the feature mappings of shared encoder for the source and target domains, while stimulate a orthogonality between the feature mappings coming from the shared and domain-exclusive encoders. At the same time, the classifier implements the classification task on data from the source domain coming from the shared encoder to ensure that the representations are also useful for the main classification task as well.

3 METHOD

In this chapter, we propose a method to address domain adaptation problem for ASC. We first glance over the method before discussing each part of the method in details. Lastly, we introduce the architecture of the models used in this project.

3.1 System Overview

Inspired by [99], our method is an adversarial-based adaptation approach to deal with mismatch conditions. The method in adaptation process does not need the reference annotations of target domain for the classification task. Therefore, this method is applicable to the problems in which no class labels are available from target domain. Figure 3.1 shows an overview of our method. It is comprised of three main steps: pre-training, adversarial adaptation, and testing.

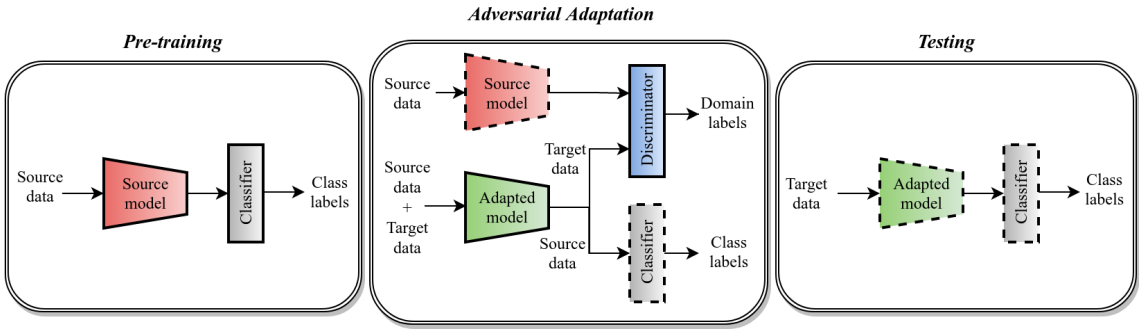


Figure 3.1. *An overview of the proposed method.*

3.1.1 Pre-training Step

Pre-training step is the first step of our method in which we try to implement an ASC task in supervised learning. In fact, this step simulates the development phase for a network that is optimized only for a specific domain of data: source domain. To clarify each part of the used network, from now on in this work, We use *model* for the first part of our network, and represent it using M. The second part is called *classifier* which is shown using C. In this step, since our model is optimized only on data from the source domain, we call it **source model**, and it is represented by M_S. We train the network using the data from the source domain $\mathbf{X}^S = \{X_1^S, X_2^S, \ \ldots, X_{N_S}^S\}$ together with its corresponding ground truth labels of acoustic scenes $\mathbf{Y}^S = \{\mathbf{y}_1^S, \mathbf{y}_2^S, \ \ldots, \mathbf{y}_{N_S}^S\}$ where the number of

data from the source domain is denoted by N_S. In order to optimize the network, the error $\mathcal{L}_S$ is minimized with respect to M_S and C:

$$\mathcal{L}_S = -\sum_{n=1}^{N_S} \mathbf{y}_n^S (log(C(M_S(X_n^S)))) \tag{3.1}$$

M_S is responsible to extract high-level representations of input data and classifier, as it can be indicated by its name, is to classify the feature mappings of M_S (i.e. the outputs of M_S) into pre-defined classes.

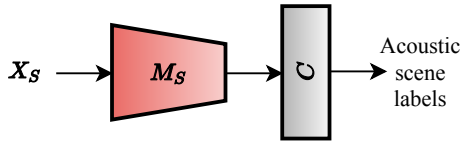


Figure 3.2. *The scheme used in the pre-training step.*

3.1.2 Adversarial Adaptation Step

Once the pre-training is completed, the second step is adversarial adaptation which is the main core of our method where the pre-trained model (i.e. M_S) is adapted to the target domain data. As denoted by its name, adversarial adaptation uses an adversarial training scheme to implement the adaptation process.

To the best of our knowledge, in the context of machine learning, The term *adversary* was introduced by Dalvi et al. [108] in order to show the fragility of linear classifiers encountering with inputs designed to fool the classifiers [109]. However, Szegedy et al. later defined *adversarial examples* as the imperceptibly perturbed examples which are able to affect the stability of neural networks [110]. Later, Goodfellow et al. introduced the employment of adversarial process to estimate generative models, i.e. GANs, [98]. One of the GANs capabilities is generating realistic data –e.g. images– of real and virtual objects such as human faces, and human poses, to name a few [111, 112]. In general, to create similar images to samples of a real image dataset, as shown in Figure 3.3, GANs pits a generator and a discriminator against each other. Generator tries to create images similar to samples in real dataset in order to fool the discriminator that these samples are drawn from the real dataset, while the discriminator is to distinguish between data coming from real dataset and fake data, i.e. produced by the generator.

Similarly, we employ the adversarial training method in this work. However, we implement this method in an discriminative manner. To do so, instead of using a generator to produce the fake data, we use data from the target domain in the place of fake samples and data from the source domain as the real samples. In our work, two main tasks are performed: *domain classification*, *ASC*. The origin of each sample of data is known in this step, meaning the data are annotated for the domain they come from. However, the ASC task is performed only using the data from the source domain because the data from target

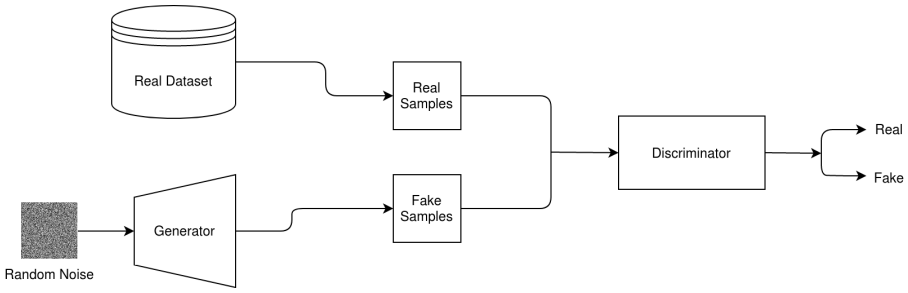


Figure 3.3. *The generic sceheme used in GANs.*

domain is considered to be non-annotated for acoustic scenes.

In addition to the M_S and the C from previous step, we introduce two new neural networks to complete adversarial adaptation scheme of our work: adapted model (M_A) and domain discriminator (D). Figure 3.4 depicts the structural implementation of this step. The solid lines used in the illustration of D and M_A denote that optimization is done only on these two networks, while dashed lines used in the representation of C and M_S is to show that these two networks are used in the evaluation mode meaning their parameters are not optimized during adaptation process. The architecture of M_A is exactly the same as the one used in M_S. Additionally, the parameters of M_A are initialized by the values of parameters in M_S. M_S receives the samples of source domain as inputs, while the input to M_A is data from the both domains, i.e. source and target. Since data from target domain is considered to be non-annotated, the output of M_A only for the data from the source domain is fed to C to do ASC. Moreover, the output of M_A for data from the target domain together with the output of M_S is given as inputs to D to perform domain classification task(Figure 3.4).

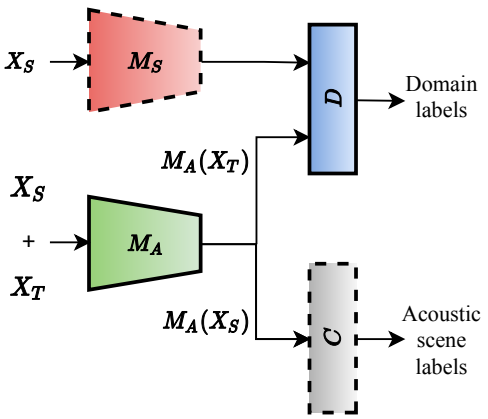


Figure 3.4. *The scheme used in the adversarial adaptation step. This is the main step of our method where adapted model is adapted to data from the target domain.*

The goal of this step is to learn a set of parameters for M_A that enables it to produce invariant latent representations of the data from the source and target domains in order to fool domain discriminator D which tries to distinguish between the domain of latent representations. The input data from the target domain is defined as $\mathbf{X}^T = \{X_1^T, X_1^T, \ \ldots, X_{N_T}^T\}$ where N_T is the total number of samples from the target domain,

and this data is considered without acoustic scene labels. In addition, D performs a binary classification as an auxiliary task in our method to classify the domains of data. We use the ground truth labels for the domain of data, i.e. domain labels, and predicted labels produced by D to minimize the error $\mathcal{L}_{\mathcal{D}}$ with respect to D, which is given as

$$\mathcal{L}_D = -\sum_{n=1}^{N_S}(log(D(M_S(X_n^S))) + log(1 - D(M_A(X_n^T)))). \tag{3.2}$$

On the other hand, in order to impose similarity between the distribution of $M_A(\mathbf{X}^T)$ and $M_S(\mathbf{X}^S)$, D and M_A are optimized jointly. In this regard, the error $\mathcal{L}_{M_A}$ given as

$$\mathcal{L}_{M_A} = -\sum_{n=1}^{N_S}(log(D(M_A(X_n^T))) + \mathbf{y}_n^S log(C(M_A(X_n^S)))) \tag{3.3}$$

is minimized with respect to M_A.

In a nutshell, the adversarial adaptation targets to boost the performance of M_A compared to the results achieved by M_S for the ASC task on the data from the target domain.

3.1.3 Testing Step

After the adversarial adaptation step, we need to evaluate the performance of M_A for unseen samples from target domain, and compare it to that of M_S to find if there is any achieved improvement over the results for target domain. To do so, we use the optimized M_A from the previous step as well as C which was optimized in the pre-training step. As shown in Figure 3.5, both of the networks are illustrated using dashed line to emphasize that they are used only in a forward pass, i.e. evaluation mode, to output predicted labels for unseen data from the target domain. In the test phase, the evaluation criterion is prediction accuracy: the number of correctly classified samples to the total number of available samples [4].

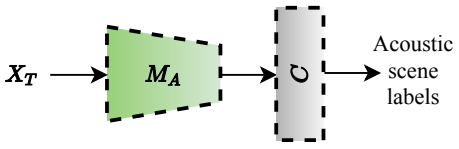


Figure 3.5. *The scheme used in the test step.*

3.2 DNN Architectures

In this section, we introduce the architecture of the DNNs employed in this work. For each network (i.e. model, classifier, and discriminator), we use two different sets of architecture, and each set has been utilized as a separate experiment. Figure 3.6 shows the first set of used DNNs architecture. The architecture of the model and classifier in this set

has been adopted from the baseline system of DCASE challenge 2018, and the first task of this challenge named acoustic scene classification [12]. We refer to this model as the **DCASE model**. The original proposed model by DCASE challenge organizers includes two convolutional layers with 32 and 64 filters respectively. Both layers employ a kernel size of (7, 7) and are followed by batch normalization, ReLU activation, and max pooling layers. The pooling layers use a kernel size of {(5, 5), (4, 100)}. This model has a slight difference with the architecture that we use as the DCASE model in our work. Since we use higher number of log-mel band energies in our input acoustic features compared to that in the development of the DCASE baseline model, we had to alter the first max pooling layer as well as the padding of the second convolutional layer in order to fully match the shape of our model output with the one of the DCASE baseline model. More specifically, we use the pooling layer with a kernel of size (8, 4). In addition, the second convolutional layer utilizes a padding of size (3, 0). When we use this model, it is employed together with a discriminator and a classifier. The discriminator first flattens the output of the model, and uses one linear layer to classify samples as the source or target domain. The classifier also uses two linear layers. The first layer is followed by a ReLU non-linearity and a dropout of 30%. Finally, the output of the second linear layer is followed by a softmax non-linearity to output the probability for each class.

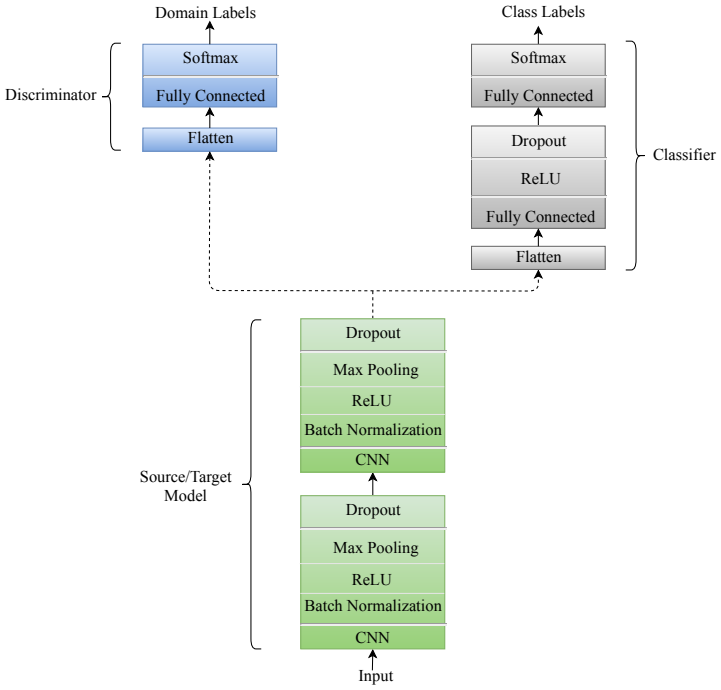


Figure 3.6. *The architecture of the DCASE model alongside with the used classifier and discriminator when the DCASE model is employed in our work. The model can be used as the source and adapted model dependent on the step of the method. The dashed arrows are to only show that the output of the model goes as input to the classifier and discriminator.*

The second set of DNNs employed in our work is shown in Figure 3.7. The architecture of the model and classifier are adopted from the winner of an ASC contest on Kaggle platform [113] which is inspired by AlexNet [16]. We refer to the model as the **Kaggle**

model. This model is a CNN-based architecture with 5 convolutional layers. The layers feature the kernels of size {(11, 11), (5, 5), (3, 3), (3, 3), (3, 3)} together with {48, 128, 192, 192, 128} as the number of used filters. In the first two convolutional layers, we use the strides of size (2, 3) while the rest use the strides of size (1, 1). The first two convolutional layers as well as the last one are CNN blocks in which each convolutional layer is followed by ReLU as the non-linearity, max pooling layer, and a batch normalization layer. However, the rest of convolutional layers are followed only by ReLU. In the experiment that we use the Kaggle model, it is always employed together with a specific discriminator and classifier as well. The discriminator is also a CNN-based architecture, and it consists of three convolutional layers with a kernel of size (3, 3) and {64, 32, 16} as the number of filters. All the layers are followed by ReLU as non-linearity, and a batch normalization layer. Then, it uses a flatten layer to make a vector of latent representations in order to input it to the final layer which is a fully connected layer to distinguish samples coming from source or target. The classifier, on the other hand, first flattens the output of the model. Then, it feeds this vector into two blocks of fully connected layers where each layer is followed by a ReLU and dropout of 25%. The final layer is a fully connected layer together with a Softmax non-linearity to predict the probability of each sample over different target classes.

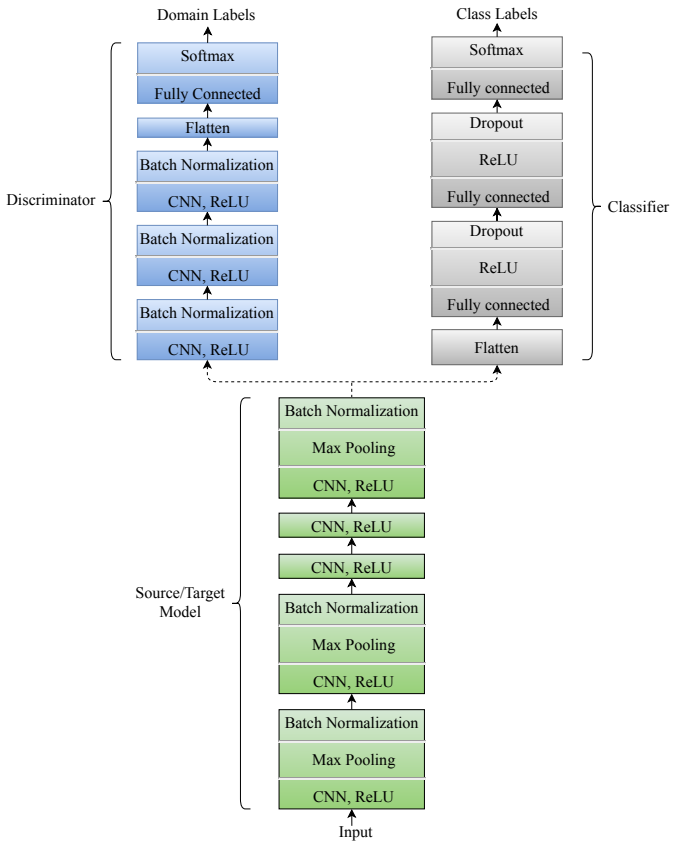


Figure 3.7. *The architecture of Kaggle model alongside with the used classifier and discriminator when the Kaggle model is employed in our work. The model can be used as the source and adapted model dependent on the step of the method. The dashed arrows are to only show that the output of the model goes as the input to the classifier and discriminator.*

We choose two different architecture sets, i.e. DCASE and Kaggle models, in order to show that the performance of the adaptation method is not tied to a specific network. In other words, the adaptation method is model agnostic.

4 EVALUATION

This chapter introduces the used dataset in this work, and details the experimental setups. Subsequently, it presents the achieved results by the adapted model, and compare it with the results obtained by the source model.

4.1 Dataset

One of the advantages that detection and classification of acoustic scenes and events (DCASE) challenge has brought to the audio community is the availability of large-scale datasets in different classification (e.g. ASC), detection (e.g. SED), and audio tagging tasks. The dataset used in this work has been first published in the DCASE challenge for the ASC task, subtask B, in 2018 which is called TUT urban acoustic scenes 2018 mobile, development dataset [12]. We use only the development set of this dataset since the evaluation set was not publicly published at the time of implementation of this work. In addition, according to the rules of the challenge, the annotations of the evaluation set cannot be published.

This dataset has been collected for ASC. More specifically, this dataset has been assembled to simulate a scenario in which the collected data for the training set is recorded with a different recording device than that used for the test set. The usage of different recording devices introduces audio recording with different quality to the dataset. As a result, This arises a mismatch condition which leads to the reduction of performance of learning algorithms when optimized using audio samples from one recorder and evaluated with audio samples recorded using devices other than the ones used in training set. Therefore, it motivates to study the methods, e.g. domain adaptation, that bridge the gap between the performance of the ASC systems for training and test sets.

As presented in details in Table 4.1, the development set of TUT urban acoustic scenes 2018 mobile dataset contains 10 different acoustic scenes from three different environments: indoor, outdoor, and transportation. The audio samples are recorded in different locations inside 6 different European cities. The samples of this dataset are recorded using three different devices. The main recording device, which accounts for the majority of audio recordings in this dataset, comprised of a Zoom F8 audio recorder together with an electret binaural microphone: Soundman OKM II Klassik/studio A3. To capture the audio analogous to the auditory system of humans, the microphones are made similar to headphones so they can be worn. The audio recorder uses 48 kHz sampling rate and

24-bit resolution. We refer to this recorder as **device A** in this work. In addition, the development set of this dataset features two other recording devices which are handheld consumer devices such as smartphones, namely Samsung Galaxy S7 and IPhone SE. We are referred to them as **device B** and **device C** respectively.

Table 4.1. *There are 10 acoustic scenes in TUT urban acoustic scenes 2018 mobile, development dataset. The variety of these acoustic scenes includes indoor, outdoor, and traveling while inside vehicle which is called transportation.*

Categories	*Indoor*	*Outdoor*	*Transportation*
Acoustic scenes	Airport Metro station Indoor shopping mall	Urban park Public square Pedestrian street Street with medium level of traffic	Travelling by a bus Travelling by a tram Travelling by an underground metro

The audio recordings are provided in 10-second segments. The number of available segments are 8640, 720, and 720 from device A, B, and C out of which each acoustic scene accounts for 864, 72, and 72 segments from device A, B, and C respectively. In total, this is equivalent to having 24, 2, and 2 hours of audio recorded using device A, B, and C respectively.

The original development set is published by the organizers of the DCASE challenge suggests a proposed training and test split. There are 6122, 540, and 540 segments from device A, B, and C in training while the test set contains 2518, 180, and 180 segments from device A, B, and C respectively. However, since the necessity of a validation set is of great significance, in order to assure the validity of the training process as well as to avoid overfitting, we partition the training split into new training and validation sets. We also hold the proposed test split intact to use as the evaluation set for our work. Figure 4.1 illustrates the details of the partitioning setup used in this work.

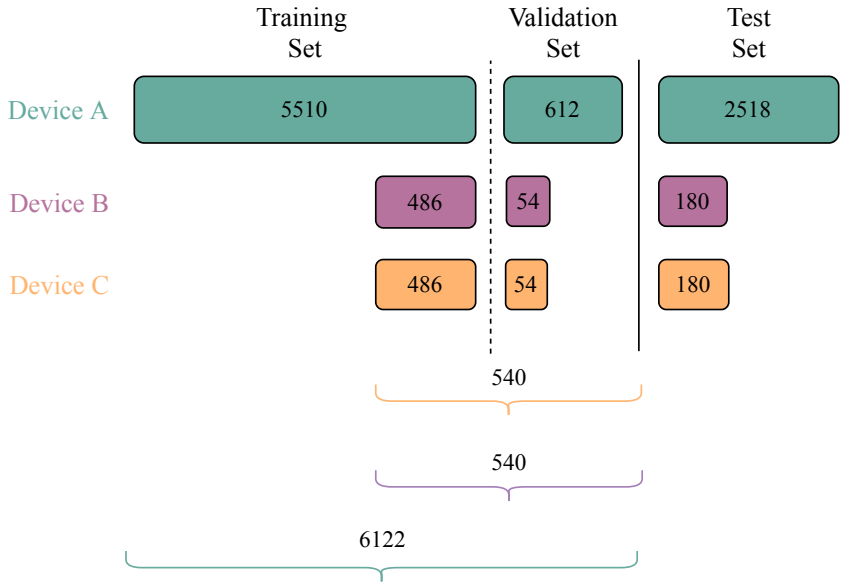


Figure 4.1. *The setup of training, validation, and test splits used in our work. The numbers present the number of 10-second segments for each split and recording device.*

4.2 Experiments

In all experiments, we use the data recorded using device A as the source domain data while the data recorded using device B and C are served as the target domain data. In addition, we use 64 log Mel-band energies as acoustic features extracted using Hamming windows of size 2048 samples (equivalent to 46 ms), and the successive windows are overlapped 50%. Acoustic feature extraction part is implemented using Librosa package [22]. The amount of available data from the source domain (i.e. data from device A) are much higher than the amount of data from the target domain (i.e. data from device B and C), therefore, we oversampled the data from the target domain, approximately 5.6 times, to achieve the same amount of data as in the source domain for the training process of the adaptation step.

The used DNNs in all experiments are trained using the cross entropy loss function and Adam optimizer with the learning rate of $1e-4$. In the pre-training step, the size of minibatches includes 38 samples which are selected only from the source domain. In the adaptation process, the number of samples in each minibatch is 16 out of which 10 are selected from the source domain, while the other 6 are selected from the target domain data: 3 samples are selected from each of B and C devices. In pre-training step, the update of parameters using backpropagation of loss is done after each iteration, while in the adaptation step and based on our observations during the training process, we found that the best results are achieved by updating the model M_A after each iteration while updating the discriminator D after 10 iterations. The reason is that the discriminator tends to converge very quickly and this leads to a gradient vanishing problem when updating the parameters of the model M_A. In addition, the model M_A is updated using the backpropagation of the sum of two losses calculated at the output of the Discriminator D and the classifier C. Since the value of the two losses are not at the same scale, we used a constant coefficient to bring the both values to the same scale. This is done by multiplying the classification loss value by 10 before adding to the calculated loss at the output of D. We stopped the optimization process after 350 epochs in the pre-training and 300 epochs in the domain adaptation step.

The implementation of the system is done using Pytorch package [114]. In addition, all the DNNs are trained using Tesla P100 GPUs on the Taito super cluster platform of CSC – IT Center for Science Ltd.

4.3 Results

This section presents the result achieved by the proposed domain adaptation method in this work. All the representative values in this section are calculated using prediction accuracy as the evaluation criterion. Prediction accuracy measures the number of corrected predictions samples to the total number of predictions. We report the results on the test set of data from both source and target domains using the source and adapted model to

show that the method is being able to improve under mismatch conditions.

Figure 4.2 illustrates the results for the test data from the source domain using the source and adapted models (i.e. M_S and M_A). In both experiments using Kaggle and DCASE models, the results show that the adapted models (i.e. green color) perform the same on the source domain data after adaptation. This means that regardless of adapting to a new domain of data, adapted models do not forget the transferred knowledge from the source models. This is the result of integrating the classifier C into the adaptation step.

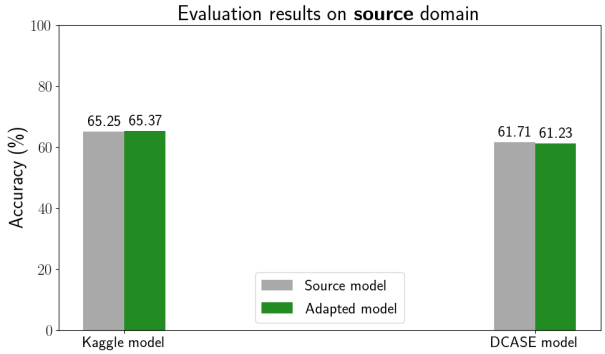


Figure 4.2. *The results of evaluation for Kaggle and DCASE models on the test set from the* ***source domain****. Grey color shows the source models in both experiments, while the green color represents the adapted ones.*

Figure 4.3 demonstrates the improvements of the results on the test data from the target domain using adapted model compared to the results achieved by the source model. In the experiment using the Kaggle model, the source model shows 20.28% correct predictions on the target domain, while the adapted model surpasses this results, and presents the accuracy of 31.67%. Similarly, the experiment on the DCASE model gives the same pattern of results where the source model shows an accuracy of 19.17%, and the adapted model outperforms the source model by achieving the accuracy of 25.28%.

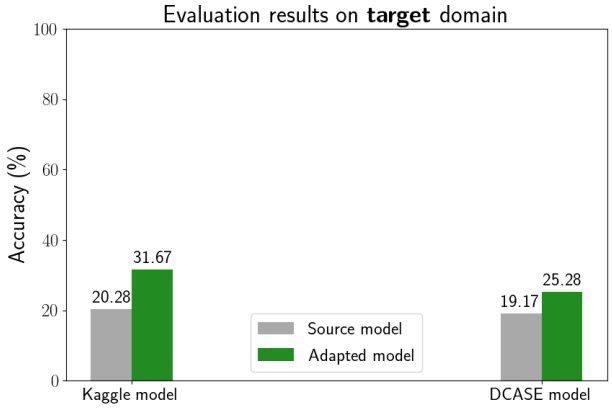
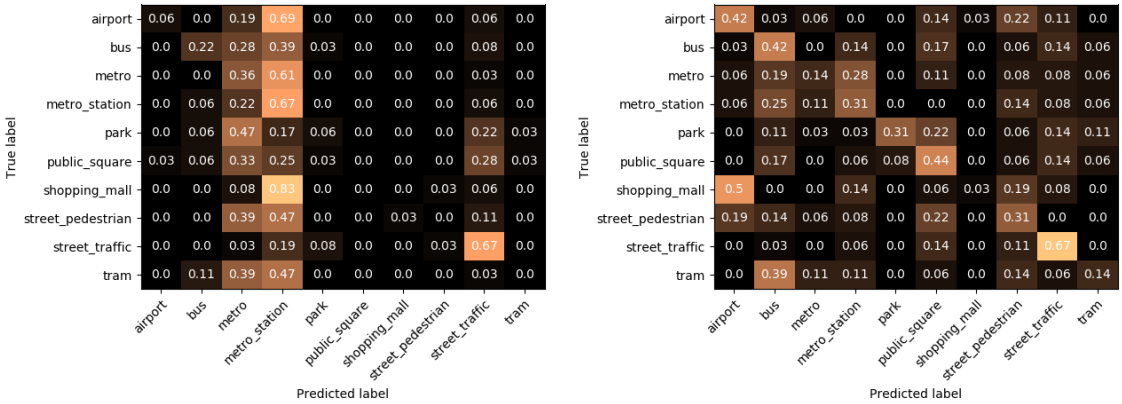


Figure 4.3. *The results of evaluation for the Kaggle and DCASE models on the test data from the* ***target domain****. Grey color shows the source models in both experiments, while the green color represents the adapted ones.*

To obtain a better understanding of how the mismatch conditions, caused by different

recording devices in the source and target domain, lead to the failure of a model in the evaluation of unseen data from a new domain, we present the confusion matrices of the source and adapted versions of the Kaggle model obtained in the evaluation of the test data from the target domain. under ideal circumstances, we expect to see the brightest colors to appear on the main diagonal of the confusion matrix. According to Figure 4.4, the source model is unable of correct classification of most of the classes from the target domain data (the left image). However, the partiality of the source model to only two of the target classes, i.e. metro and metro stations, is diminished in the confusion matrix of the adapted model (the right image) presenting a more diagonal representation of the brighter colors.



(a) *Confusion matrix for the source model* ***(b)*** *Confusion matrix for the adapted model*

Figure 4.4. *Confusion matrices of the source (a) and the adapted (b) versions of the Kaggle model for the target domain. The values are normalized according to the amount of examples in each class. Brighter colors denote higher values. The figure is adopted from [115].*

5 CONCLUSION

This thesis presented an unsupervised domain adaptation method using adversarial training as a solution to the problem of mismatch recording devices for ASC. This method employs an discriminator in an adversarial-based framework together with source and adapted models to achieve an invariant representation for the data coming from source and target domains. In addition, we presented two different architectures of DNNs in order to show that the method is model agnostic. We have evaluated our method using TUT urban acoustic scenes 2018 mobile dataset, development dataset. This dataset consists of samples recorded using different recording devices so that it creates mismatch conditions. Our method not only shows an improvement of results for the data from target domain by 11% and 6% on the adapted Kaggle and DCASE model respectively, but also it is capable of retaining the same performance on data from the source domain thanks to the employment of the pre-trained classifier to perform the ASC task for the source domain data alongside adapting the two domains in the adversarial adaptation step. Therefore, the results indicate that our adversarial-based domain adaptation method can be successfully adopted in order to address the mismatch conditions for the ASC task. Last but not least, we also noticed that it is difficult for the adapted model to converge in the adversarial adaptation step in our experiments. Therefore, we motivate the usage of a different GAN loss in future research works in order to examine the stability issue in the convergence of the adapted model in this step. As an example, One recent approach which investigated this problem can be found in [116].

REFERENCES

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

[2] Virtanen, T., Plumbley, M. D. and Ellis, D. *Computational Analysis of Sound Scenes and Events*. Springer, 2018.

[3] Drossos, K., Floros, A., Giannakoulopoulos, A. and Kanellopoulos, N. Investigating the Impact of Sound Angular Position on the Listener Affective State. *IEEE Transactions on Affective Computing* 6.1 (2015), 27–42.

[4] Mesaros, A., Heittola, T. and Virtanen, T. TUT Database for Acoustic Scene Classification and Sound Event Detection. *2016 24th European Signal Processing Conference (EUSIPCO)*. IEEE. 2016, 1128–1132.

[5] Mantyjarvi, J., Huuskonen, P. and Himberg, J. Collaborative Context Determination to Support Mobile Terminal Applications. *IEEE Wireless Communications* 9.5 (Oct. 2002), 39–45. ISSN: 1558-0687. DOI: 10.1109/MWC.2002.1043852.

[6] Eronen, A. J., Peltonen, V. T., Tuomi, J. T., Klapuri, A. P., Fagerlund, S., Sorsa, T., Lorho, G. and Huopaniemi, J. Audio-based Context Recognition. *IEEE Transactions on Audio, Speech, and Language Processing* 14.1 (2006), 321–329.

[7] Nogueira, W., Roma, G. and Herrera, P. Sound Scene Identification Based on MFCC, Binaural Features and a Support Vector Machine Classifier. *IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events* (2013).

[8] Bisot, V., Serizel, R., Essid, S. and Richard, G. Supervised nonnegative matrix factorization for acoustic scene classification. *IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events (DCASE)* (2016), 62–69.

[9] Yun, S., Kim, S., Moon, S., Cho, J. and Kim, T. Discriminative training of GMM parameters for audio scene classification and audio tagging. *IEEE AASP Challenge Detect. Classification Acoust. Scenes Events* (2016).

[10] Sakashita, Y. and Aono, M. *Acoustic Scene Classification by Ensemble of Spectrograms Based on Adaptive Temporal Divisions*. Tech. rep. DCASE2018 Challenge, Sept. 2018.

[11] Chen, H., Liu, Z., Liu, Z., Zhang, P. and Yan, Y. *Integrating the Data Augmentation Scheme with Various Classifiers for Acoustic Scene Modeling*. Tech. rep. DCASE2019 Challenge, June 2019.

[12] Mesaros, A., Heittola, T. and Virtanen, T. A Multi-device Dataset for Urban Acoustic Scene Classification. *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2018 Workshop (DCASE2018)*. Nov. 2018, 9–13. URL: https://arxiv.org/abs/1807.09840.

[13] Quionero-Candela, J., Sugiyama, M., Schwaighofer, A. and Lawrence, N. D. *Dataset Shift in Machine Learning*. The MIT Press, 2009. ISBN: 0262170051.

[14] Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F. and Vaughan, J. W. A Theory of Learning from Different Domains. *Machine Learning* 79.1-2 (2010), 151–175.

[15] Heittola, T., Mesaros, A., Eronen, A. and Virtanen, T. Context-dependent Sound Event Detection. *EURASIP Journal on Audio, Speech, and Music Processing* 2013.1 (2013), 1.

[16] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger. Curran Associates, Inc., 2012, 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[17] Kehtarnavaz, N. CHAPTER 7 - Frequency Domain Processing. *Digital Signal Processing System Design (Second Edition)*. Ed. by N. Kehtarnavaz. Second Edition. Burlington: Academic Press, 2008, 175–196. DOI: https://doi.org/10.1016/B978-0-12-374490-6.00007-6.

[18] Oppenheim, A. V. and Schafer, R. W. *Discrete-Time Signal Processing*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0131988425.

[19] Stankovic, L., Stankovic, S. and Dakovic, M. From the STFT to the Wigner Distribution [Lecture Notes]. *IEEE Signal Processing Magazine* 31.3 (2014), 163–174.

[20] Stevens, S. S., Volkmann, J. and Newman, E. B. A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America* 8.3 (1937), 185–190. URL: https://doi.org/10.1121/1.1915893.

[21] Gelfand, S. *Essentials of Audiology (Fourth)*. New York: Thieme Medical Publishers, Inc, 2016.

[22] McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E. and Nieto, O. librosa: Audio and Music Signal Analysis in Python. *Proceedings of the 14th Python in Science Conference*. Ed. by K. Huff and J. Bergstra. 2015, 18–24. DOI: 10.25080/Majora-7b98e3ed-003.

[23] Gin-Der Wu and Chin-Teng Lin. Word Boundary Detection With Mel-scale Frequency Bank in Noisy Environment. *IEEE Transactions on Speech and Audio Processing* 8.5 (2000), 541–554.

[24] Mitrović, D., Zeppelzauer, M. and Breiteneder, C. Features for Content-based Audio Retrieval. *Advances in computers*. Vol. 78. Elsevier, 2010, 71–150.

[25] Molau, S., Pitz, M., Schluter, R. and Ney, H. Computing Mel-frequency Cepstral Coefficients on the Power Spectrum. *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*. Vol. 1. IEEE. 2001, 73–76.

[26] Huang, X., Acero, A., Hon, H.-W. and Reddy, R. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. 1st. USA: Prentice Hall PTR, 2001. ISBN: 0130226165.

[27] Logan, B. et al. Mel Frequency Cepstral Coefficients for Music Modeling. *Ismir*. Vol. 270. 2000, 1–11.

[28] Mitchell, T. M. *Machine Learning*. 1st ed. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077.

[29] Cakir, E. Deep Neural Networks for Sound Event Detection. *Tampere University Dissertations* 12 (2019).

[30] Roma, G., Nogueira, W., Herrera, P. and Boronat, R. de. Recurrence Quantification Analysis Features for Auditory Scene Classification. *IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events* 2 (2013).

[31] Zhao, W., Chellappa, R., Phillips, P. J. and Rosenfeld, A. Face Recognition: A Literature Survey. *ACM Comput. Surv.* 35.4 (Dec. 2003), 399–458. URL: https://doi.org/10.1145/954339.954342.

[32] Chellappa, R., Sinha, P. and Phillips, P. J. Face recognition by Computers and Humans. *Computer* 43.2 (2010), 46–55.

[33] Smith, C., McGuire, B., Huang, T. and Yang, G. The History of Artificial Intelligence. *University of Washington* 27 (2006).

[34] McCulloch, W. S. and Pitts, W. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics* 5.4 (1943), 115–133. URL: https://doi.org/10.1007/BF02478259.

[35] Rosenblatt, F. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review* (1958), 65–386.

[36] Caudill, M. Neural Networks Primer, Part I. *AI Expert* 2.12 (Dec. 1987), 46–52. ISSN: 0888-3785.

[37] Gurney, K. *An Introduction to Neural Networks*. USA: Taylor & Francis, Inc., 1997. ISBN: 1857286731.

[38] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[39] Glorot, X., Bordes, A. and Bengio, Y. Deep Sparse Rectifier Neural Networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson and M. Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html.

[40] Bridle, J. S. Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters. *Advances in Neural Information Processing Systems 2*. Ed. by D. S. Touretzky. Morgan-Kaufmann, 1990, 211–217. URL: http://papers.nips.cc/paper/195-training-stochastic-model-recognition-algorithms-as-networks-can-lead-to-maximum-mutual-information-estimation-of-parameters.pdf.

[41] He, K., Zhang, X., Ren, S. and Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV '15. USA: IEEE Computer Society, 2015, 1026–1034. URL: https://doi.org/10.1109/ICCV.2015.123.

[42] Glorot, X. and Bengio, Y. Understanding the Difficulty of Training Deep Feedforward Neural Networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[43] McClelland, J. L. and Rumelhart, D. E. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*. MIT press, 1989, pp. 130–131.

[44] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. Learning Representations by Back-propagating Errors. *Nature* 323.6088 (Oct. 1986), 533–536. DOI: 10.1038/323533a0.

[45] Prechelt, L. Early Stopping — But When?: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by G. Montavon, G. B. Orr and K.-R. Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, 53–67. URL: https://doi.org/10.1007/978-3-642-35289-8_5.

[46] Polyak, B. T. Some Methods of Speeding Up the Convergence of Iteration Methods. *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), 1–17.

[47] Ruder, S. An Overview of Gradient Descent Optimization Algorithms. *ArXiv preprint arXiv:1609.04747* (2016).

[48] Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[49] Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning* 4.2 (2012), 26–31.

[50] Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S. and Bengio, Y. Identifying and Attacking the Saddle Point Problem in High-dimensional Non-convex Optimization. *Advances in Neural Information Processing Systems*. 2014, 2933–2941.

[51] Schmidhuber, J. Deep Learning in Neural Networks: An Overview. *Neural Networks* 61 (2015), 85–117. DOI: https://doi.org/10.1016/j.neunet.2014.09.003.

[52] KELLEY, H. J. Gradient Theory of Optimal Flight Paths. *ARS Journal* 30.10 (1960), 947–954. URL: https://doi.org/10.2514/8.5282.

[53] Bengio, Y. Learning Deep Architectures for AI. *Found. Trends Mach. Learn.* 2.1 (Jan. 2009), 1–127. URL: https://doi.org/10.1561/2200000006.

[54] Simonyan, K. and Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*. 2015.

[55] Mohri, M., Rostamizadeh, A. and Talwalkar, A. *Foundations of Machine Learning*. Mit Press, 2018, pp. 1–8. ISBN: 9780262018258.

[56] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15.56 (2014), 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[57] Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by F. Bach and D. Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[58] Bishop, C. M. et al. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[59] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE* 86.11 (1998), 2278–2324.

[60] Jansson, P. A. Neural Networks: An Overview. *Analytical Chemistry* 63.6 (1991), 357A–362A.

[61] Minsky, M. and Papert, S. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[62] Hornik, K., Stinchcombe, M. and White, H. Multilayer Feedforward Networks Are Universal Approximators. *Neural Netw.* 2.5 (July 1989), 359–366. ISSN: 0893-6080.

[63] LeCun, Y. and Bengio, Y. Convolutional Networks for Images, Speech, and Time Series. *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, 255–258. ISBN: 0262511029.

[64] Yann Le Cun, Bottou, L. and Bengio, Y. Reading Checks with Multilayer Graph Transformer Networks. *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 1. Apr. 1997, 151–154 vol.1. DOI: 10.1109/ICASSP.1997.599580.

[65] Dumoulin, V. and Visin, F. A guide to Convolution Arithmetic for Deep Learning. *ArXiv e-prints* (Mar. 2016). eprint: 1603.07285.

[66] Zhou, Y. T. and Chellappa, R. Computation of Optical Flow Using a Neural Network. *IEEE 1988 International Conference on Neural Networks* (1988), 71–78 vol.2.

[67] Nagi, J., Ducatelle, F., Di Caro, G. A., Cireşan, D., Meier, U., Giusti, A., Nagi, F., Schmidhuber, J. and Gambardella, L. M. Max-pooling convolutional neural networks for vision-based hand gesture recognition. *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. 2011, 342–347.

[68] Sainath, T., Vinyals, O., Senior, A. and Sak, H. Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks. *ICASSP*. 2015.

[69] Pan, S. J. and Yang, Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22.10 (Oct. 2010), 1345–1359. ISSN: 2326-3865. DOI: 10.1109/TKDE.2009.191.

[70] Shimodaira, H. Improving Predictive Inference under Covariate Shift by Weighting the Log-likelihood Function. *Journal of Statistical Planning and Inference* 90.2 (2000), 227–244. ISSN: 0378-3758. DOI: https://doi.org/10.1016/S0378-3758(00)00115-4.

[71] Yamazaki, K., Kawanabe, M., Watanabe, S., Sugiyama, M. and Müller, K.-R. Asymptotic Bayesian Generalization Error When Training and Test Distributions Are Different. *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, 1079–1086. URL: https://doi.org/10.1145/1273496.1273632.

[72] Bickel, S., Brückner, M. and Scheffer, T. Discriminative Learning for Differing Training and Test Distributions. *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, 81–88. URL: https://doi.org/10.1145/1273496.1273507.

[73] Moreno-Torres, J. G., Raeder, T., Alaiz-RodrıéGuez, R., Chawla, N. V. and Herrera, F. A Unifying View on Dataset Shift in Classification. *Pattern Recogn.* 45.1 (Jan. 2012), 521–530. URL: https://doi.org/10.1016/j.patcog.2011.06.019.

[74] Alaiz-Rodrıéguez, R. and Japkowicz, N. Assessing the Impact of Changing Environments on Classifier Performance. *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer. 2008, 13–24.

[75] Cieslak, D. A. and Chawla, N. V. A Framework for Monitoring Classifiers' Performance: When and Why Failure Occurs?: *Knowledge and Information Systems* 18.1 (2009), 83–108.

[76] Webb, G. I. and Ting, K. M. On the Application of ROC Analysis to Predict Classification Performance under Varying Class Distributions. *Machine learning* 58.1 (2005), 25–32.

[77] Tasche, D. Fisher Consistency for Prior Probability Shift. *The Journal of Machine Learning Research* 18.1 (2017), 3338–3369.

[78] Schlimmer, J. C. and Granger, R. H. Beyond Incremental Processing: Tracking Concept Drift. *AAAI*. 1986, 502–507.

[79] Gama, J., Žliobaitundefined, I., Bifet, A., Pechenizkiy, M. and Bouchachia, A. A Survey on Concept Drift Adaptation. *ACM Comput. Surv.* 46.4 (Mar. 2014). URL: https://doi.org/10.1145/2523813.

[80] Zadrozny, B. Learning and Evaluating Classifiers under Sample Selection Bias. *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML '04. Banff, Alberta, Canada: Association for Computing Machinery, 2004, 114. URL: https://doi.org/10.1145/1015330.1015425.

[81] Ditzler, G., Roveri, M., Alippi, C. and Polikar, R. Learning in Nonstationary Environments: A Survey. *IEEE Computational Intelligence Magazine* 10.4 (2015), 12–25.

[82] Minku, L. L. Transfer Learning in Non-stationary Environments. *Learning from Data Streams in Evolving Environments*. Springer, 2019, 13–37.

[83] Zhao, H., Combes, R. T. d., Zhang, K. and Gordon, G. J. On Learning Invariant Representation for Domain Adaptation. *ArXiv preprint arXiv:1901.09453* (2019).

[84] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., March, M. and Lempitsky, V. Domain-Adversarial Training of Neural Networks. *Journal of Machine Learning Research* 17.59 (2016), 1–35. URL: http://jmlr.org/papers/v17/15-239.html.

[85] Gong, B., Shi, Y., Sha, F. and Grauman, K. Geodesic Flow Kernel for Unsupervised Domain Adaptation. *2012 IEEE Conference on Computer Vision and Pattern Recognition*. June 2012, 2066–2073. DOI: 10.1109/CVPR.2012.6247911.

[86] Donahue, J., Hoffman, J., Rodner, E., Saenko, K. and Darrell, T. Semi-supervised Domain Adaptation with Instance Constraints. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2013.

[87] Yao, T., Pan, Y., Ngo, C.-W., Li, H. and Mei, T. Semi-Supervised Domain Adaptation With Subspace Learning for Visual Recognition. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.

[88] Hoffman, J., Mohri, M. and Zhang, N. Algorithms and Theory for Multiple-source Adaptation. *Advances in Neural Information Processing Systems*. 2018, 8246–8256.

[89] Mansour, Y., Mohri, M. and Rostamizadeh, A. Domain Adaptation With Multiple Sources. *Advances in Neural Information Processing Systems*. 2009, 1041–1048.

[90] Wang, M. and Deng, W. Deep Visual Domain Adaptation: A Survey. *Neurocomputing* 312 (2018), 135–153. URL: http://www.sciencedirect.com/science/article/pii/S0925231218306684.

[91] Zhang, L. Transfer Adaptation Learning: A Decade Survey. *ArXiv* abs/1903.04687 (2019).

[92] Csurka, G. *A Comprehensive Survey on Domain Adaptation for Visual Applications*. Ed. by G. Csurka. Cham: Springer International Publishing, 2017, 1–35. URL: https://doi.org/10.1007/978-3-319-58347-1_1.

[93] Wilson, G. and Cook, D. J. A Survey of Unsupervised Deep Domain Adaptation. *ArXiv preprint arXiv:1812.02849* (2018).

[94] Gebru, T., Hoffman, J. and Fei-Fei, L. Fine-Grained Recognition in the Wild: A Multi-task Domain Adaptation Approach. *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), 1358–1367.

[95] Borgwardt, K. M., Gretton, A., Rasch, M. J., Kriegel, H. P., Schölkopf, B. and Smola, A. J. Integrating Structured Biological Data by Kernel Maximum Mean Discrepancy. *Proc. of Intelligent Systems in Molecular Biology (ISMB)*. Fortaleza, Brazil, 2006, e49–e57.

[96] Saito, K., Ushiku, Y. and Harada, T. Asymmetric Tri-Training for Unsupervised Domain Adaptation. *Proceedings of the 34th International Conference on Machine*

Learning - Volume 70. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, 2988–2997.

[97] Tzeng, E., Hoffman, J., Zhang, N., Saenko, K. and Darrell, T. Deep Domain Confusion: Maximizing for Domain Invariance. *CoRR* abs/1412.3474 (2014). arXiv: 1412.3474. URL: http://arxiv.org/abs/1412.3474.

[98] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. Generative Adversarial Nets. *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence and K. Q. Weinberger. Curran Associates, Inc., 2014, 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[99] Tzeng, E., Hoffman, J., Saenko, K. and Darrell, T. Adversarial Discriminative Domain Adaptation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, 2962–2971. DOI: 10.1109/CVPR.2017.316.

[100] Liu, M.-Y. and Tuzel, O. Coupled Generative Adversarial Networks. *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon and R. Garnett. Curran Associates, Inc., 2016, 469–477. URL: http://papers.nips.cc/paper/6544-coupled-generative-adversarial-networks.pdf.

[101] Zhu, J.-Y., Park, T., Isola, P. and Efros, A. A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017.

[102] Kim, T., Cha, M., Kim, H., Lee, J. K. and Kim, J. Learning to Discover Cross-Domain Relations with Generative Adversarial Networks. *Proceedings of the 34th International Conference on Machine Learning*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, 1857–1865.

[103] Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H. Greedy Layer-wise Training of Deep Networks. *Advances in Neural Information Processing Systems*. 2007, 153–160.

[104] Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.-A. Extracting and Composing Robust Features with Denoising Autoencoders. *Proceedings of the 25th International Conference on Machine Learning*. 2008, 1096–1103.

[105] Glorot, X., Bordes, A. and Bengio, Y. Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach. *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, 513–520. ISBN: 9781450306195.

[106] Chen, M., Xu, Z., Weinberger, K. Q. and Sha, F. Marginalized Denoising Autoencoders for Domain Adaptation. *Proceedings of the 29th International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, 1627–1634. ISBN: 9781450312851.

[107] Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D. and Erhan, D. Domain Separation Networks. *Advances in Neural Information Processing Systems 29*.

Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon and R. Garnett. Curran Associates, Inc., 2016, 343–351. URL: http://papers.nips.cc/paper/6254-domain-separation-networks.pdf.

[108] Dalvi, N., Domingos, P., Mausam, Sanghai, S. and Verma, D. Adversarial Classification. *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. Seattle, WA, USA: Association for Computing Machinery, 2004, 99–108. ISBN: 1581138881. DOI: 10.1145/1014052.1014066. URL: https://doi.org/10.1145/1014052.1014066.

[109] Biggio, B. and Roli, F. Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, 2154–2156. ISBN: 9781450356930. DOI: 10.1145/3243734.3264418. URL: https://doi.org/10.1145/3243734.3264418.

[110] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R. Intriguing Properties of Neural Networks. *International Conference on Learning Representations*. 2014. URL: http://arxiv.org/abs/1312.6199.

[111] Ma, L., Jia, X., Sun, Q., Schiele, B., Tuytelaars, T. and Van Gool, L. Pose Guided Person Image Generation. *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Curran Associates, Inc., 2017, 406–416. URL: http://papers.nips.cc/paper/6644-pose-guided-person-image-generation.pdf.

[112] Radford, A., Metz, L. and Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2016. URL: http://arxiv.org/abs/1511.06434.

[113] Gharib, S., Derrar, H., Niizumi, D., Senttula, T., Tommola, J., Heittola, T., Virtanen, T. and Huttunen, H. ACOUSTIC SCENE CLASSIFICATION: A COMPETITION REVIEW. *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*. Sept. 2018, 1–6. DOI: 10.1109/MLSP.2018.8517000.

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

[115] Gharib, S., Drossos, K., Cakir, E., Serdyuk, D. and Virtanen, T. Unsupervised Adversarial Domain Adaptation for Acoustic Scene Classification. *Proceedings of*

the Detection and Classification of Acoustic Scenes and Events 2018 Workshop (DCASE2018). Nov. 2018, 138–142.

[116] Drossos, K., Magron, P. and Virtanen, T. Unsupervised Adversarial Domain Adaptation Based on the Wasserstein Distance for Acoustic Scene Classification. *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE. 2019, 259–263.