UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

Towards Unsupervised Domain Adaptation for Diabetic Retinopathy Detection in the Tromsø Eye Study

Magnus Størdal

FYS-3900 Master's thesis in physics - 60 ECTS - May 2021

Abstract

Diabetic retinopathy (DR) is an eye disease which affects a third of the diabetic population. It is a preventable disease, but requires early detection for efficient treatment. While there has been increasing interest in applying deep learning techniques for DR detection in order to aid practitioners make more accurate diagnosis, these efforts are mainly focused on datasets that have been collected or created with ML in mind. In this thesis, however, we take a look at two particular datasets that have been collected at the *University Hospital of North-Norway* - UNN.

These datasets have inherent problems that motivate the methodological choices in this work such as a variable number of input images and domain shift.

We therefore contribute a multi-stream model for DR classification. The multi-stream model can model dependency across different images, can take in a variable of input of any size, is general in its detection such that the image processing is equal no matter which stream the image enters, and is compatible with the domain adaptation method ADDA, but we argue the model is compatible with many other methods.
As a remedy for these problems, we propose a multi-stream deep learning architecture that is uniquely tailored to these datasets and illustrate how domain adaptation might be utilized within the framework to learn efficiently in the presence of domain shift.

Our experiments demonstrates the models properties empirically, and shows it can deal with each of the presented problems. The model this paper contributes is a first step towards DR detection from these local datasets and, in the bigger picture, similar datasets worldwide.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

The prevalence of diabetic retinopathy

1.1 The obesity pandemic

Since 1975 the world has seen a massive increase of obese adults. According to a study covering about 19.2 million subjects in 186 countries over the period 1975-2014, we have seen an increase from 105 million (3.2% of the 1975 world population) to 641 million (10.8% of the 2014 world population) obese adults [12]. We have naturally seen a correlated increase in cardiovascular diseases, some types of cancers, and diabetes. According to the WHO's global report on diabetes [93], we have seen a likewise massive increase in reported diabetes cases. In 1980 there were 108 million reported cases (4.7% of the adult population) compared to 422 million in 2014 (8.5% of the adult population). Diabetes brings a lot of health risks and unique problems for the affected, one of these being diabetic retinopathy.

Diabetic retinopathy (DR) is an eye disease which affect about 34.6% (146 million) of the diabetic population according to the WHO [59]. This disease can lead to loss of vision, retinal detachment and glaucoma. The disease is however treatable, quite so in-fact. The key to proper treatment is early detection of the disease. This is not an easy task however, as early symptoms are easily missed by the human eye. Furthermore, according to the *Association of American Medical Colleges* there is an expected shortage of between 21400 - 55200 primary care practitioners, and 33700 - 86700 non-

primary care practitioners by 2033 in the US alone if current trends hold [4]. This is not a US specific trend as global projections seem to come to the conclusion that the demand for healthcare workers will outgrow the number of healthcare workers [46, 44]. Given a reduction in capacity, and an almost guaranteed increase in DR cases, we need to develop tools to lessen the burden on practitioners and increase diagnosis accuracy. Automatic systems and analytical tools are the immediately obvious solution to solve this problem. These kinds of automated systems would be categorised as clinical decision support systems, or CDS for short.

Automatic systems based on machine learning have been used for many years in the medical field [65, 47], and are showing more and more promise recently. The first machine learning based CDS system dates back to the 1970's with the GOFAI[1] models. These systems made use of image processing techniques, and mathematical modelling and applied a simple rule-based decision system. Due to new and better suited hardware, and a lack of significant progress, symbolic approaches to AI where made more irrelevant as models relying on neural networks became prominent [25]. In more recent times such neural network based approached have been quite successful as CDS systems. For example, Richens et.al [65] made a model which placed in the top 25% of general practitioners in the London area when diagnosis realistic patient cases. Others have found that having models assisting professional health workers improves their diagnostically accuracy and reduces under-diagnosing patients [68, 21].

One subfield of machine learning that is emerging in particular in the medical field is deep learning. Deep learning has already had, and will most likely have a large role in our future healthcare systems[42, 61]. There are several benefits to using deep learning models as CDS systems. Deep learning systems have the ability to uncover patterns, correlations, and features that might be either invisible, or incomprehensible to humans. These systems are therefore invaluable when acting as a second pair of eyes for any professional health worker. Compared to most alternative machine learning methods, deep learning methods also avoids the hand crafting of features, and rather relies on the models being able to craft these themselves. There are several benefits to this, one being the process is much more automated versus having

[1]Good old-fashioned artificial intelligence systems

computer scientists, and medical experts handcrafting each potential useful feature.

One particular common data modality in the medical field are images, which is also the focus of this thesis. Within the deep learning field, one method in particular has found extensive use for image data, that being the convolutional neural network (CNN). CNN models are flexible to suit most grid-like data, which most often means image data. Medical diagnostics heavily rely on manual analysis of images, be they retina images, x-ray images, CT images, etc. As such the field is ripe for the application of CNN models and methods. CNNs work by extracting features which are all summarised by a feature vector. These vectors are often robust representations of the original input, and are comparatively easy to classify through simple neural networks. Due to the CNNs ability to extract important features and patterns, they are excellent in detecting lesions such as hemorrhages, ischemic tissue, and other defects. This in turn makes them excellent for DR detection.

There has been an increase in interest in applying deep learning models and methods in DR detection in the last decade [68, 21, 2, 90, 53, 62]. This specific topic has seen, and continues to see a huge diversity of models, and method combinations in an attempt to solve the tedious task of DR classification. Most published models tend to focus on the big public DR datasets; These being Messidor-2, EyePacs, APTOS 2019, etc. These datasets come fully labeled with thousands of images. This, however, does not always represent the practical setting. Labeling images is costly, and single labeled images are often not available. Retina screening is often performed by taking multiple images at different angles through the pupil to get a full image of the eyes interior. While the eye is labeled, the images are not which is an important distinction as labeling the images according to the eye grading would lead to images without visible symptoms often being labeled as diseased. Simply adopting such a simple approach would therefore lead to erroneous labels, leading to unstable training, and degradation in performance.

While there exists a few public DR datasets, the models trained with these datasets are not always directly applicable to real life cases. Ignoring the multiple image per eye problem, there is also something called *domain shift* to take into consideration. Domain shift refers to a shift in the underlying data distribution. A domain shift between two datasets can be

brought about due to the datasets not using the same equipment, camera settings, illumination, pose, image quality and more [96, 92]. In more practical terms, a domain shift between two datasets mean a model trained on the first dataset, will not perform as well, or not at all on the other. This is due to the model using the first datasets distribution, which does not align with the second datasets distribution. The field of domain adaptation (DA) specialize in methods which either reduce, or negates the effects of domain shift. A model which is compatible with domain adaptation could in theory apply its labeled knowledge on any new, and unlabeled dataset. DA tend to differentiate between datasets as *source*, and *target*. Source is often a fully labeled dataset, and target is either unlabeled, or has a few examples from each class to aid in the domain adaptation process. DA is just one of many approaches to classification with unlabeled data, as the field of *unsupervised learning* deals with classification without the need for a ground-truth. This is generally achieved through clustering or similar approaches.

This thesis has been given two realistic diabetic retinopathy datasets from the university hospital of Tromsø, UNN. The details, challenges, and our solution for this dataset will be explained in the sections to come.

Unlike prior studies that mostly consider the standard datasets (Messidor, EyePacs, APTOS, etc), we consider a locally collected dataset. Note that this dataset, as many medical datasets, is not collected with machine learning in mind, leading to several challenges. In the following sections, we will describe the dataset detail the particular challenges and describes our proposed solutions.

1.2 The dataset

The following sections will detail in short how the dataset is built up, and some challenges that comes with the dataset.

This thesis deals with DR image classification on data provided by the *Universitetssykehuset Nord-Norge HF* (UNN for short). The data is split into two datasets. The first dataset was collected in relation to *Tromsø Studien 6*[17], where the first *Tromsø øyestudie 1*[7] took place. The second dataset was collected in relation to *Tromsø Studien 7*[2], where the second *Tromsø øyestudie 2*[3] took place. The latter studies are as of this thesis not yet published. From this point out we'll refer to these two datasets as T6 and T7 respectively.

The datasets are built up of multiple unique eyes. T6 has a total of $n = 13080$ unique graded eyes, while T7 has a total of $n = 14211$ unique graded eyes. Each eye has a number of images attached to it. This number varies, but should in theory be six. These images are taken at different angles through the pupil in order to visualize as much of the eyes interior as possible.

The T6 dataset is taken from a population of adults within the age group of $38 - 87$ years old. The T7 dataset is not yet released, but it is assumed the age group will be relatively similar.

Each unique eye is graded between 0 and 4, or five unique grades. Each grade correlated to how far the degradation of the eye has come. These five classes are as follows

- Grade 0: No DR
- Grade 1: Mild DR
- Grade 2: Moderate DR
- Grade 3: Severe DR
- Grade 4: Proliferate DR

We will go deeper into the symptoms and visual cues each stage exhibit in the materials and methods section (Section III).

[2]The Tromsø Study 7 is a combination of ongoing projects, and a list of publications can be found here: `https://uit.no/research/tromsostudy`

[3]The timeline for the second eye study can be found here: `https://app.cristin.no/projects/show.jsf?id=543079`

There are some significant challenges to these datasets. Multiple images per eye, huge imbalance, and different data gathering methods lead to some interesting problems which will be explained in the next section.

1.2.1 Dataset Challenges

This section aims to outline some of the challenges we faced with when performing DR classification with the given dataset, and some challenges with DR in general.

One hurdle for the provided datasets is how they are structured. Each of the datasets is collected on an eye-to-eye basis. For each eye observed, a number of images are taken. All of these images share a ground truth which corresponds to the diagnosis of the entire eye.

Why is this a problem? Given that each eye consists of several images, all under a single ground truth; It is important to realise that conventional CNN architectures will not aid us here. CNN architectures makes the assumption that each individual image is independent from each other. That is, the images are self containing, and all the information needed to correctly classify an image is all encompassed in a single image. This is not possible with this dataset however. Diabetic retinopathy as a disease slowly degrades the retina, meaning one image which shows a single region in the eye might not display any symptoms. In the case where an image displays what looks like a healthy eye, but the ground-truth is diseased, one would consider this image a noisy image. Training a CNN as normal on this data would thus lead to the model trying to learn both on useful and noisy data. This in turn will make the model unstable at best, and at worst useless.

- **Challenge #1:** The first challenge of these datasets is to make a model which can model dependence between different images which have been taken from the same eye. The idea being that we want to classify the eye, which we have labels for, and not for individual sections of the eye, which we don't have labels for.

One of the main challenges for this thesis is to construct a model which can deal with multiple images representing the same eye, and allow for feature information from all of the eyes to aid in the final classification. By default each eye should consist of six images. This is confirmed by a manual for Tromsø's eye clinic named "Prosedyrebok Øyestasjon Tromsø 6". The

dataset however consists of eyes with varying amounts of images. Some subjects did not want to go on with the screening after a couple of images, while some images where not up to standard, leading to more than six images being taken. How these images are collected exactly will be explained in detail later in the thesis, but sufficient to say a specialized camera is used to take images through the pupil. Unlike normal image classification task, there is no off-the-shelf model which can be used for this task. As no previous model can be directly applied to this problem, it is necessary to build a model which is fit for this task. It is desirable to construct a model which can take in any given number of images, and give a collective classification using useful information from all input images.

- **Challenge #2:** The second challenge of these datasets is a varying amount of images per eye. Our model must be able to process multiple images in parallel, and combine this together with the problem shown in the first challenge.

- **Challenge #3:** The third challenge comes from the fact that the eye images are not in any particular order. This meaning what region of the eye which was loaded first one time, might not be the same region loaded first the next time. Our model must be able to accommodate for this fact.

DR screening is heavily reliant on equipment such as cameras. Just in the UK there are 30 accepted different retina screening cameras approved for use [1]. This is going to lead to some domain shift occurring between datasets, and ours are no different. The provided T6 and T7 datasets are taken with different cameras and settings. Training the same model on both datasets might prove detrimental to performance as the underlying distribution might be rather complex. However, this problem is also a golden opportunity. If the model which is created for these datasets is also compatible with different DA methods, then there is nothing to stop a single baseline model being created, and distributed world wide to be tuned to unlabeled data in the clinic where the model will be applied. This is all to say the constructed model for this thesis must also be compatible with current DA methods.

- **Challenge #4:** The final challenge is a more practically minded one. The model which can deal with the first and second challenge should also be compatible with DA methods. The two datasets are collected in

such a manner that a domain shift has occured. Having the capability of adapting to new data allows for a base model to adapt onto new data from different clinics.

DR data in general struggles heavily with the problem of class imbalance. In the case of the Tromsø Eyestudy data this imbalance comes from the fact that they surveyed a sample of the general population in the Tromsø area[7]. This means that the **No DR** classification contains images from the non-diabetic population as well as parts of the diabetic population. This massively inflates the **No DR** class. This imbalance favours the **No DR** grade so heavily to the point where data for the other four classes simply may not be enough. Figure 1.1 shows the class imbalance found in dataset T6 and T7.

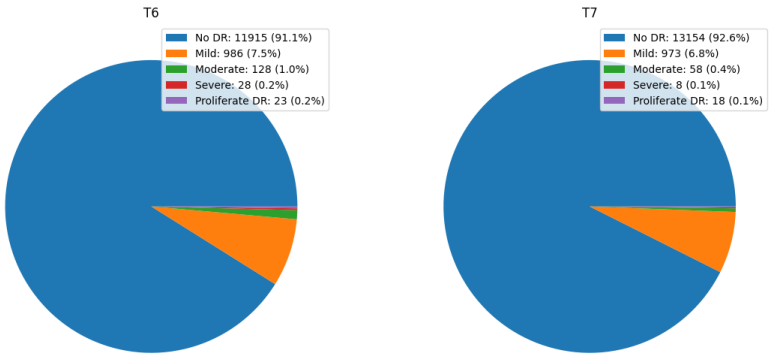


Figure 1.1: Visualisation of the massive class imbalance found in both the T6 and the T7 datasets

There are several ways of addressing this imbalance. This is however outside the scope of While there are several ways of addressing this imbalance, however this a challenge for DR classification in general, and not something we address in this thesis.

1.3 Contributions

This section aims to give the reader an idea of the contributions made in this thesis. The main focus of this thesis is medical image classification, especially diabetic retinopathy classification. Most diabetic retinopathy classification that has been done up to this point has been done on the larger public datasets (EyePacs, Messidor, APTOS 2019, etc) [68, 2, 21]. These datasets consists of thousands of fully labeled retina images. This thesis instead considers data provided by UNN from two local studies, *Tromsø øyestudie 1* [7], and *Tromsø øyestudie 2* (To be released later this year). This data consists of eyes with a single label, but multiple images. In order to address the challenges mentioned in the previous section, we propose a novel multi-stream architecture for the task of DR classification. In particular, we address the challenges as follows:

The contribution of this thesis is a novel specially constructed multi-stream model fit to analyse and classify the provided DR data. The network is capable of processing multiple dependent images in parallel to address **Challenge** #2, and fuse the individual feature vectors into a feature vector representing the eye as a whole, and creating dependency which addresses **Challenge** #1. The only practical limit for this network is the GPU memory, but theoretically the proposed network can take a number of inputs between $[1, \infty]$. The model is also shown to be compatible with DA methods, specifically ADDA which directly addresses **Challenge** #4. We further argue that the model is compatible with other DA methods. The retina region the images are taken is also not provided. We therefore construct a multi-stream where each stream is equally capable of detecting lesion, addressing **Challenge** #3

To the authors knowledge there are no previous publications of multi-stream diabetic retinopathy classification which deals with multiple dependent images and domain adaptation.

In summary, we propose a novel network architecture that is particularly tailored to the challenges inherited in our local dataset.

1.4 Thesis outline

This thesis is split up into multiple parts. These parts being *The introduction* I, *Theory and Related Work* II, *Materials and Methods* III, *Experiments and Results* IV, and *Discussion and future work* V

The following part of the thesis will explain the relevant theory for this thesis, going into detail on the basic concepts of machine learning, CNNs, and domain adaptation. The theory is set up such that anyone with sufficient background in mathematics, or machine learning should be able to follow along. There will be a red thread from the beginning, where each part will motivate the next; accumelating in a complete theory behind CNNs, DA, and multistream structures. Each part will build further upon the preceding sections, all accumulating into the final model used in this thesis which will be explained in detail in the materials and methods section.

The materials and methods section will go through the dataset used in this thesis. This part will describe what diabetic retinopathy is, and the structure of the datasets. This will motivate the presented model for this classification task.

The experiment and result section will describe the experiments done in this thesis, their purpose, and other relevant information needed to understand the results presented in the results chapter within this part. This will illustrate the ability of our model to address the aforementioned challenges.

The thesis ends with the discussion and future work, and conclusion section. The discussion chapter will expand upon some of the ideas in this thesis, and give pointers into how the model could be improved. The thesis will conclude with a summary of the thesis.

Part II

Theory and Related Work

Chapter 2

Notation

This part of the thesis will go in depth into the underlying theory which the experiments, and the thesis builds upon. Starting from the fundamental statistical theory, this part will build into the basic building blocks of a modern neural network, and transition into CNN theory, domain adaptation and multistream neural network theory. At the end the reader should have the theoretical understanding to follow the methodology, experiments and discussion which will follow in the next parts of the thesis.

The notations found in machine learning literature are not consistent. This might be due to the researchers backgrounds, localization etc. It's therefore important that for clarity we define what each symbol means to reduce misinterpretations. This thesis will be using the same notations found in the book "Deep Learning"[27], and the used notations will be reiterated in this section.

General symbols

a	Any scalar
$\mathbf{a}$	Any vector
a_i	i-th element of vector $\mathbf{a}$
$\mathbf{A}$	Any matrix
$A_{i,j}$	Element of matrix $\mathbf{A}$ found on the i-th row and j-th column
$\phi(\cdot)$	A mapping function
$p(x)$	A probability density function
$P(A)$	Probability of event A

Specific symbols

$\boldsymbol{\theta}$	The set of parameters for a given function
$\mathcal{L}$	A undefined loss
$\mathcal{D}$	A domain

Chapter 3

What is Machine Learning

Machine learning is a branch of computer science which heavily leans itself upon statistical principles. The field of machine learning is considered a sub-field of AI [85]. The main goal of machine learning can be described as the analysis of data. Machine learning algorithms have shown to be able to detect visual cues, patterns, and trends which humans have not been able to pick up on [18]. This is done by letting the network learn from data. Within the field of machine learning is the field of deep learning [85], and the relevant deep learning techniques will be discussed from section 7. Deep learning networks often have millions of parameters which all contribute to effectively making a complex mapping function into a high dimension. In the ideal case, this mapping will reveal some complex structure in the data which can be discriminated. This is all achieved through the introduction of learning examples, giving the network an update scheme which allows it to reduce the mistakes it makes, and then let the model optimize towards that specific goal. This is the essence of deep learning.

Within the field of machine learning we can define three main learning schemes. They are defined by how the data is annotated, and depending on how much of the data has a ground-truth, we need to adapt different methods for discrimination.

- *Supervised learning*: In the supervised setting all data comes with a ground truth, mostly referred to as a "label". These labels are measured against the networks prediction for its data pair, and through a loss function we can compare the predictions against the labels and update the network to reduce miss classification. The case of supervised

learning is often seen as the best way to get a model which gives solid predictions, but due to the time-consuming task of labeling data it is often not possible to use this form of training scheme.

- *Unsupervised learning*: In the unsupervised setting we have no labels attached to the data points. There are no ground truth to compare the output of the model with, so loss functions which do not rely on a label are used in this setting. It's up to the designer of the algorithm to choose how the model should update itself in this case. Arguably the most common way to analyse unlabeled data is by designing loss functions that encourage clustering of similar examples, where each cluster tends to be a unique classification.

- *Semi-supervised learning*: The semi-supervised setting is a mix of both unsupervised and supervised learning. In this setting we have a couple of data-label pairs from each class whereas the rest of the data is unlabeled. A loss combining both the supervised information (e.g. through a classification loss), and the unsupervised information.

- *Reinforcement learning*: In the reinforcement learning setting an agent learns by trial and error to maximize some reward. By letting the agent interact with an environment it can learn patters which lets it achieve its goal with the most reward. The input for such models is the environment itself, and often the environment itself might try to push the model into a non-ideal/non-rewarding state and the model "pushes" back.

Chapter 4

Risk minimization and Classification

This section aims to give an understanding of the underlying fundamental statistics which describe how classifications in machine learning are made.

While statistics and probability theory plays an important part in machine learning, one of the most important principles is arguably *Risk minimization*. Risk minimization is the most extensively used framework when it comes to design, construction and analysis of machine learning algorithms[18, 32, 70].

Risk and *Bayes classification rule* are heavily intertwined, sand provides a natural starting point for the following discussion. Assuming a binary classification problem, the feature space will contain a region where the first class (C_0) is found, and another region where the second class (C_1) lies. These regions are denoted as R_0 and R_1. The ideal regions would be regions which minimize the classification error. The probability of missclassifying a datapoint is found as

$$P(error) = P(C_0)\int_{R_1} p(\mathbf{x}|C_0)d\mathbf{x} + P(C_1)\int_{R_0} p(\mathbf{x}|C_1)d\mathbf{x} \tag{4.1}$$

$$= \int_{R_1} P(C_0|\mathbf{x})p(\mathbf{x})d\mathbf{x} + \int_{R_0} P(C_1|\mathbf{x})p(\mathbf{x})d\mathbf{x} \tag{4.2}$$

Note that the above expression assumes that error in either class is equal. This however is not always the case. Sometimes one might want less error

for a single class, which comes at the cost of the other classes. This is not uncommon when working with medical data where one might want to have more false-positives rather than false-negatives. It is possible to weight the terms of the risk to adjust the rate of getting either false-positives or false-negatives. With $\lambda_i, i \in [0, 1]$ denoting the individual classification weights, the probability of missclassification now becomes

$$P(error) = \lambda_0 \int_{R_1} P(C_0|\mathbf{x})p(\mathbf{x})d\mathbf{x} + \lambda_1 \int_{R_0} P(C_1|\mathbf{x})p(\mathbf{x})d\mathbf{x} \tag{4.3}$$

Due to the complex nature of most data distributions, it is near impossible to find the risk analytically. Finding it empirically however is much more plausible. This can be done in a few different ways, such as with Monte Carlo Estimation. The estimated risk for a class k, in a classification task with N classes can be written as

$$r_k = \sum_{i=1}^{N} \lambda_k \int_{R_i} p(\mathbf{x}|C_k)d\mathbf{x} \tag{4.4}$$

Where the average risk can be found with

$$r = \sum_{i=1}^{N} r_k P(C_k) = \sum_{i=1}^{N} \int_{R_i} \left(\sum_{i=1}^{N} \lambda_{ki} p(\mathbf{x}|C_k)P(C_k) \right) \tag{4.5}$$

With a knowledge of what risk is, it is now time to find ways of reducing it. Risk is not something anyone wants. No matter what a model does in real life, high risk only brings about miss classifications, and a model with high risk is generally useless. This realization is important as it shows why a model should choose the prediction which minimizes risk. Furthermore, this leads us to a way of defining a classification rule which will be described in the next section.

4.1 Loss functions

Most machine learning and deep learning algorithms attempts to maximize the models ability to provide predictions for the data which the model has never observed. So when the model during training makes a mistake, our model needs to understand how it's wrong, and how to modify itself as to

not make the same mistake later. But to do this the model needs to somehow quantify "wrongness". The term loss is used to quantify the wrongness of a model. A loss function helps penalizing a bad decision made by the model, and minimizing loss equates directly to minimizing risk [18, 70].

Lets tie risk together with loss. For any action α_i which assign a class to some example $\mathbf{x}$, there is an underlying expected risk [18]. The action of assigning class C_i to $\mathbf{x}$ incur some loss λ_{in}. The expected risk can be defined as

$$R(\alpha_i|\mathbf{x}) = \sum_{n}^{N} \lambda_{in} P(C_n|\mathbf{x}) \tag{4.6}$$

Where λ_{in} is defined as the binary value

$$\lambda_{in} = \begin{cases} 0 & if\ i = n \\ 1 & if\ i \neq n \end{cases}$$

The action α_i is generally to assign the example to the highest probable class, that is

$$\alpha_i : \mathbf{x} \rightarrow C_i \quad if\ P(C_i|\mathbf{x}) > P(C_n|\mathbf{x})\ \forall i \neq n$$

λ_{in} makes sure that no risk is incurred for assigning the proper class to an example. When a classification is wrong however we can see that the risk incurred is higher the more confident the model is in its wrong decision. We now have a tangible way of quantifying a models wrongness, though somewhat primitive. Note that while this loss might work, it has its downsides. If say a model is confident in its prediction, say $p(C_1|\mathbf{x}) = 0.9$, it's still slightly uncertain. It's desirable to "punish" the model for not being sure as to make it approximate the underlying distribution better. The following sections will go into detail on methods which incorporate loss for such cases.

The loss function is also known as the objective function [27]. There exists quite a few different losses, where we either try to minimize or maximize said loss value. One common way to use the loss for optimization is to take the derivative with respect to each component of the model to figure out how much each component influenced the error (read. the loss) and the model is updated. This is known as back propagation. The loss is essentially a function that depends on all the weights and variables in a network and we wish to find the set of parameters that minimizes/maximizes the loss. The most well known method for this is by gradient decent.

4.1.1 Cross Entropy Loss

One of the most commonly used loss functions is the cross entropy loss. Cross entropy loss punishes models for not classifying confidentially, but the loss also implicitly punishes the approximated data distribution for not matching the true data distribution.

Cross entropy stems from information theory which is a field that seeks to quantify information in communication [18, 81]. Information is defined as

$$h(x) = -log(p(x)) \tag{4.7}$$

Where $p(x)$ is the probability of some event. Information theory uses different names for the information quantity depending on the log base, e.g. a base-2 log quantifies its information in *bits*, while log-e based information has the units *nats*. The output from this formula will tell us how many bits (or units) are needed to convey the observed result of some event. Another way of measuring information is entropy. Entropy is a measure of the average information a random variable contains. Entropy for a random variable X with n possible states can be calculated as [18, 81, 70]

$$H(X) = -\sum_{n} p(n)log(p(n)) \tag{4.8}$$

Note that the entropy is highest when all classes are equally probable for a given datapoint, and lowest when a single state has a probability of ~ 1.0, making entropy a reasonable loss function in and off itself. Moving on to cross entropy. Cross entropy measures the difference between two probability distributions for a random variable. [18, 70]

$$H(P, Q) = -\sum_{x \in X} P(x)log(Q(x)) \tag{4.9}$$

While not immediately intuitive for most people, cross entropy is a measurement of the average number of bits required to identify an event. This is equivalent to saying "how many questions must you ask on average to find the correct classification?". Asking questions which leads to highly probable classifications first leads of course to the quickest classifications, and if all questions are of equal probability, then the cross entropy is equal to entropy. This relation is easier seen in the reformulation of the cross entropy equation which relates it directly to entropy and the Kullback-Leibler divergence [27].

$$H(P, Q) = H(P) + D_{KL}(P||Q) \tag{4.10}$$

where D_{KL} is the Kullback-Leibler (KL) divergence, which becomes zero when the distributions p, q are equal, that is to say when there is a equal chance for an observation to come from either distribution. KL divergence is the expected difference between two distributions. So how do these concepts fit so well as loss functions.

One attribute which is desired from a loss function is for it to be high when predictions are bad, and low when predictions are good. This is solved mostly by using the negative log likelihood as a loss. Cross entropy is derived from this concept (through entropy), but has the additional term of KL divergence. Assume p is the distribution a model has learned, and q is some assumed true underlying distribution our data follows, then not only is the model punished for guessing wrong. It is also punished for learning dissimilar distributions to the true underlying distribution. For a multiclass problem, the cross entropy loss is written as

$$\mathcal{L}_{CE} = -\sum_{c=1}^{C} y_c log(p_c(x)) \tag{4.11}$$

where

$$y_c = \begin{cases} 1 & if\ x\ belongs\ to\ class\ c \\ 0 & otherwise \end{cases}$$

and $p_c(x)$ is the probability for observation x belonging to class c.

4.1.2 Focal Loss

Diabetic retinopathy is plagued with class imbalance. As seen in the introduction section, some classes are barely represented, while the "No DR" class dominates. While it is not always possible to simply get more data, one can attempt to put the less represented classes into focus. In cross entropy loss this is done by a simple scalar weighing of the class-specific losses. Focal loss modifies the cross entropy loss in such a way that less represented classes are scaled in a way that is proportional to the uncertainty behind the classification. This allows for less confidential predictions to modify the networks the most so their predictions becomes confidential.

Focal loss is a relative recent addition to the machine learning field. Focal loss builds on the popular cross entropy loss, and seeks to address the problem of extreme class imbalances [39]. It's not uncommon to reduce the effects of class imbalance in a dataset by weighing the loss of a class by its inverse

frequency. Using the notation for the paper [39], cross entropy loss for a binary class problem is defined by the following terms

$$p_t = \begin{cases} p & if\ y = 1 \\ 1 - p & otherwise \end{cases}$$

Which makes the cross entropy function look like

$$\mathcal{L}_{CE}(p_t) = -\alpha_t log(p_t) \tag{4.12}$$

Where α_t is a hyper parameter that corresponds to the class weight, if any. α_t is normally set to be the inverse of the class frequency in an attempt to balance out the loss propagated by each class. The problem focal loss seeks to address with this approach to imbalance is the fact that a α_t needs to be defined for each class, and in many cases it's desired to push forward some classes more than the others. The focal loss takes away individual weights for each class, and replaces it with a modulating factor with a single tuneable parameter. The resulting loss for a two class problem is expressed as

$$\mathcal{L}_{FL}(p_t) = (1 - p_t)^{\gamma} log(p_t) \tag{4.13}$$

for all $\gamma \geq 0$. The focal loss will weight examples that are missclassified (read. small p_t) much more than correctly classified examples. When the model correctly classifies an example, and is certain in its prediction, the modulating factor will go towards zero, minimizing the loss. The γ variable is called the focusing parameter. This variable effectively determines how fast the effects of the modulating factor should fall off. At high values of γ the modulating factors scaling effect disappears much earlier than for lower values.

Chapter 5

Gradient Decent

When a loss has been found for a batch of examples, there must be some way of using this quantity to modify the network in a way which reduces the miss classification. This is done by back propagating the loss to find a gradient for each element in the model. How these gradients are calculated will be detailed in later sections, but sufficient to say it is found by calculating $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$. This is done for all elements in $\boldsymbol{\theta}$ by applying the chain rule repeatedly. By changing the parameters in the direction of the negative gradient the model is guided towards a loss minima, which means a minimization of loss is achieved, and by extension a minimization of classification error. This is an iterative method which is fairly simple in its implementation. For a given parameter θ_i we have the following update scheme [18, 70, 27]

$$\theta_{i+1} \leftarrow \theta_i - \alpha \nabla_\theta \mathcal{L}(f(\mathbf{x}|\boldsymbol{\theta}), \mathbf{y})$$

Where α is the step size, $\nabla_\theta \mathcal{L}(\mathbf{x}|\boldsymbol{\theta})$ is the gradient of the parameter and θ_i being the parameter at the i-th step. With enough iterations it is expect of that model to reach a minimum (global or local) where the gradient is zero and the updates come to an end. However there are a few undesired cases where the gradient is also zero. A model might be unlucky enough to initialize a start on a maxima of sorts. The update scheme simply will not work as there is no gradient. The solution to this is fairly simple however, and it's to run the training more than once as too even initialize on a maxima is extremely unlikely and the deeper the network, the more unlikely it becomes due to all the dimensions the gradient has to be zero in. The chances of initializing all parameters on a maxima is effectively zero.

The second case are saddle points. These are more likely to be encountered during training and in rare cases can slow the training down to a crawl or stop it completely. This is mostly just a problem in the most basic of gradient decent techniques and in the smaller networks. Most SotA[1] techniques implements momentum which allows for updates even when there is currently no gradient present. Much like a ball rolling down a hill we don't expect it to immediately stop as the slope flattens.

The last case is local minima. During training we wish to find the global minima of the loss landscape, that is to find the set of parameter values in which the classification error is small as the network allows it to be. This case is more tricky than the other cases as we have no way of knowing if we've reached the global minima or not. Momentum can in some cases help, but it's far from guaranteed. The best way to deal with this case is the rather arduous task of simply training the network over and over again with different initialization, and then choosing the best performing network as the one that reached the global minima.

There are a number of different gradient decent schemes. Here is a short description of the most well known ones.

Momentum: Momentum is not an exact gradient decent implementation, but more of a tool which most gradient decent schemes implement. It is therefore valuable to know the concept before tackling the more advanced gradient decent ideas. Momentum helps the optimization algorithm to get out of local minimas and reduces oscillation when performing gradient decent [18, 70]. Momentum is simply adding a fraction (usually some number around 0.9) of the previous gradient to the current gradient. The update scheme is modified as follows

$$\begin{aligned} v_i &= \alpha \nabla_\theta \mathcal{L}(f(\mathbf{x}|\boldsymbol{\theta}), \mathbf{y}) + \gamma v_{i-1} \\ \theta_{i+1} &\leftarrow \theta_i - v_i \end{aligned} \tag{5.1}$$

where γ is the fraction of the previous gradient that passes onward. Intuitively one might think of a ball rolling down a hill, which does not immediately stop when reaching elevated ground due to its inertia.

SGD: Stochastic Gradient Decent (SGD) is much like the normal gradient decent, except the steps are done batchwise [18]. A batch refers to a subset of the dataset. Gradient decent calculate the gradient for the entire

[1] State of the Art

training dataset, then does a single step. This is extremely inefficient for large datasets. SGD does a step for each batch which massively speeds up training, and while the steps will not be in the optimal gradient direction, we can expect over a number of iterations for it to be reasonably similar. The less optimal optimization is a good trade off for the massively increased training speed.

Nestrov: Nestrov accelerated gradient stands out from the other optimization methods in that is uses its "to be" position to find out if the next step is to great or in some way sub-optimal [8, 27]. The way this is done is to modify the standard momentum term to approximate two updates ahead and adjusting before making an update. The adjusted update scheme for SGD with nestrov accelerated gradient becomes

$$\begin{aligned} v_i &= \gamma v_{i-1} + \alpha \nabla_{\theta_i - \gamma v_{i-1}} \mathcal{L}(f(\mathbf{x}), \mathbf{y}) \\ \theta_{i+1} &\leftarrow \theta_i - v_i \end{aligned} \tag{5.2}$$

This will result in a larger step in the "normal" update direction, and a smaller step in the approximated next step. This allows the Nestrov algorithm to avoid climbing up from steep ravines as can easily happen with the momentum algorithm.

AdaGrad: Adagrad is a gradient optimization scheme in which the learning rate for different parameters is scaled them inversely proportional to the accumilated square gradient [14, 27]. By scaling the learning rate by the inverse of the accumulated square gradient less important features will get larger updates, while the more important features gets smaller updates as they come into play more frequently. The scheme can be written as the following three steps

$$\begin{aligned} \mathbf{g} &\leftarrow \frac{1}{N} \nabla_\theta \mathcal{L}(f(\mathbf{x}|\boldsymbol{\theta}), \mathbf{y}) \\ \mathbf{G} &\leftarrow \mathbf{G} + \mathbf{g} \odot \mathbf{g} \\ \theta_{t+1} &\leftarrow \theta_t - \frac{\alpha}{\epsilon + \sqrt{\mathbf{G}}} \mathbf{g} \end{aligned} \tag{5.3}$$

where $\odot$ is the pairwise multiplication, and ϵ is there to ensure numeric stability. AdaGrad has its uses as in convex optimization it has some desired theoretical properties [27]. However due to the accumulated square gradient becoming larger and larger over the course of training, AdaGrad experiences what is called dimensional death. This can lead to the model not learning anymore as the stepsize will effectively be zero along some dimensions.

Adam: Adaptive Moment Estimation [31, 27] is arguably one of the more popular optimization schemes to date. The Adam optimization algorithm uses two decaying averages. The idea resembles that of AdaGrad, but due to how Adam uses averages in its adaptive parameters, it also avoids the dying dimension problem. The two decaying averages is defined as

$$\begin{aligned} m_i &= \beta_1 m_{i-1} + (1-\beta_1) g_i \\ v_i &= \beta_2 v_{i-1} + (1-\beta_2) g_i^2 \end{aligned} \tag{5.4}$$

where g is calculated in the same way as Adagrad, and (β_1, β_2) is two hyper parameters often set to $(0.9, 0.999)$ respectively by default. m_0 and v_0 are initialized as zero. Due to this initialization, the expressions will be biased towards zero (because of the beta values being close to one). The Adam algorithm deals with this problem by bias correcting the two values as follows

$$\begin{aligned} \hat{m}_i &= \frac{m_i}{1-\beta_1^i} \\ \hat{v}_i &= \frac{v_i}{1-\beta_2^i} \end{aligned} \tag{5.5}$$

With these two values accounted for, the Adam update scheme is calculated as

$$\theta_{i+1} \leftarrow \theta_i - \frac{\alpha}{\sqrt{\hat{v}_i} - \epsilon} \hat{m}_i$$

Where $\hat{m}_i$ and $\hat{v}_i$ will contain the gradients. Adam has shown to work extremely well. Unlike SGD which applies momentum separately, the adam update scheme incorporates it into the update by default.

Chapter 6

Perceptrons

The perceptron is the most basic building block of any neural network. A perceptron consists of a set of weights, $\mathbf{w}$, which are used to weigh each feature in an input example, $\mathbf{x}$. The set of weights also incorporates a bias b. The weights and bias are learnable with the learning schemes which was discussed in the previous section. Figure 6.1 shows visually how a single perceptron works

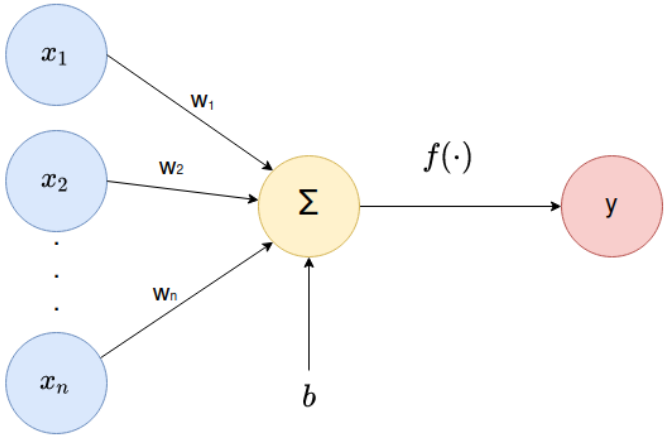


Figure 6.1: A simple visualisation of a basic perceptron

The perception output (also known as potential) is calculated as

$$v = \mathbf{w}^T \mathbf{x} = \sum_i w_i x_i + b \tag{6.1}$$

The potential is denoted as v as this is not the final output of the perceptron, just the weighted sum. In Figure 6.1 there is a function $f(\cdot)$ between the weighted sum of the perceptron and the output. This function is known as the activation function. This is a concept that will be discussed, but sufficient to say this function applies a non-linearity to the output. This ensures the perceptron is more than just a simple linear-remapping of the original input.

Visually a perceptron can be thought of as a single line in some abstract high-dimensional space. The weight vector will correspond to a line/planes normal vector, and the resulting dot-product seen in equation 6.1 will be positive or negative depending on which side of the plane a datapoint $\mathbf{x}$ is relative to the normal vector $\mathbf{w}$. That is, the datapoints on the side of the plane to which the weight vector is pointing will be positive, and on the other side they'll be negative.

6.1 Activation function

Activation functions are a set of functions which sole purpose is to introduce nonlinearity, which is helpful when the perceptron is tied into larger models. These functions are applied to the output of perceptrons or similar operations. As seen in equation 6.1, the output of a single perceptron is calculated as

$$v = \mathbf{w}^T \mathbf{x} \Leftrightarrow v = \sum_i w_i x_i + b \tag{6.2}$$

Note that the output v is defined for all real values, $v \in \mathbb{R}$. The output value not being constrained has several problems. Due to the possible large values, the calculated gradients might follow suit in sheer size. This leads to huge update steps, making the models performance unstable. Large values can also lead to "dominant" values which by themselves can determine the output of the network, making other information irrelevant. While it will first be relevant in the next section, there is also the consideration that this is nothing but a simple identity mapping. This is a problem, as stacking

multiple perceptrons with identity mappings is equivalent to a single layer of perceptrons.

Activation functions as non-linearities works because over multiple layers, the network can approximate any non-trivial continuous function [10]. The non-linearities does not allow the multiple linear layers to be simplified into a single one. This in turn is what allows us to approximate the underlying distribution of any given training data. Consider the sequential layers without non-linearities

$$\hat{y} = \mathbf{w}_l^T(\mathbf{w}_{(l-1)}^T(\mathbf{w}_{(l-2)}^T.....(\mathbf{w}_0^T\mathbf{x}))) \tag{6.3}$$

This can be approximated as

$$\hat{y} = \mathbf{w}_l^T(\mathbf{w}_{(l-1)}^T(\mathbf{w}_{(l-2)}^T.....(\mathbf{w}_0^T\mathbf{x}))) = \mathbf{w'}^T\mathbf{x} \tag{6.4}$$

This simplification cannot be done with the introduction of non-linearities.

The backpropagation also tends to see massive instability if not constrained properly by non-linearities. Activation functions works as a non-linear mappings, often with hard constrains. The functions being non-linear is important for the network to learn high degree polynomials for discrimination[58]. Activation functions also allows us to determine if we want the neuron to "fire", and to what degree it should be allowed to do so[13].

6.1.1 The sigmoid function

One of the most well known activation functions is the sigmoid function. The sigmoid function is fairly simple, and its derivative can be expressed as a product of itself.

$$f(x) = \frac{1}{1+e^{-x}} \tag{6.5}$$

$$\frac{\partial}{\partial x}f(x) = f(x)(1-f(x)) \tag{6.6}$$

Visualising the functions for values between $x \in [-5, 5]$ we get the plots seen in figure 6.2

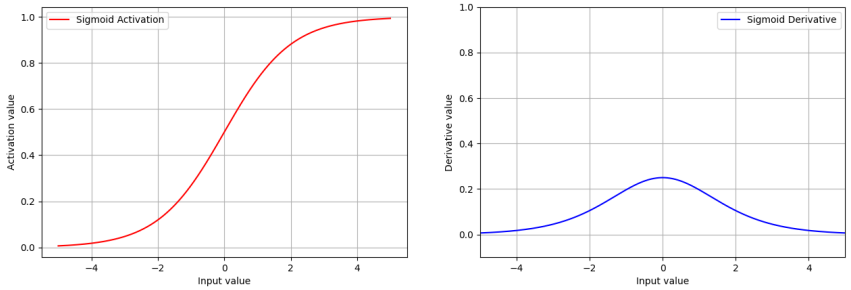


Figure 6.2: The leftmost image shows how the sigmoid output changes based on input value. The visualisation is only for the range $x \in [-5, 5]$. The rightmost image shows the sigmoid derivative value for input values in the same range

Figure 6.2 shows how the sigmoid function squeezes values into the range $y \in [0, 1]$. Higher output values will allow a neuron to fire more strongly than lesser output values. The possible derivative values shown in the Figure 6.2 shows us the sigmoid only has possible values in the range $y \in [0, 0.25]$.

The range of valid values in the sigmoids derivative leads us to a problem known as *vanishing gradients*. Vanishing gradients appear due to how we calculate the gradient. A network with multiple instances of the sigmoid activation function will have to take this activation into account multiple times during backpropagation. Due to the multiplication done in the chain rule the gradient propagated furthest back into the network will start to vanish as sigmoid gradient is at most 0.25, which in turn is multiplied with itself several times.

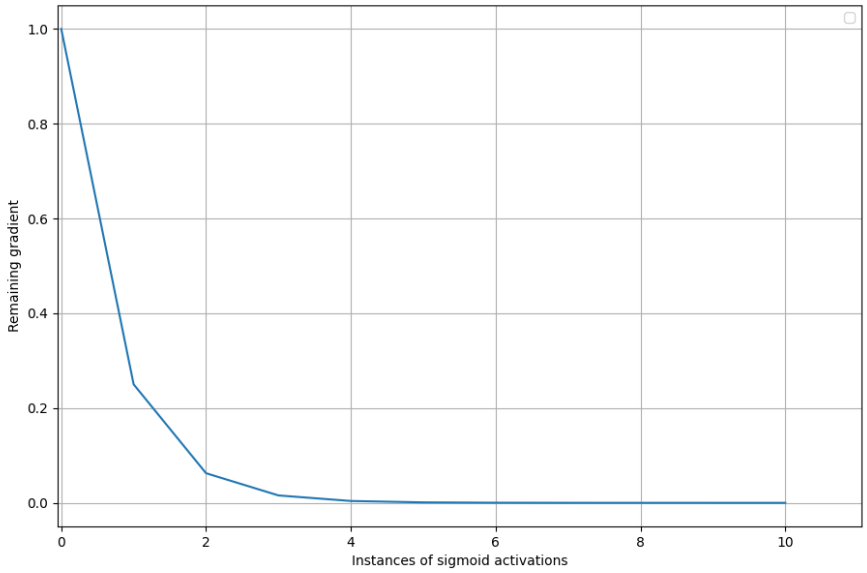


Figure 6.3: Visualisation of vanishing gradient. Here we assume an initial loss of 1. The x-axis shows how the gradient is affected by a number of sigmoid activation functions

Figure 6.3 gives us a good visualisation of how a number of sigmoid functions can affect the gradient during backpropagation. The figure is a representation of how the gradient vanishes over multiple sigmoid activations. This leads us to the question "What if the activations derivative is more than 1?". This case is called *explosive gradient*. Unlike with vanishing gradients where the begining of a network would see barley any gradient for gradient decent, now we see a too large of a gradient. This problem is however much simpler to address than vanishing gradients. Exploding gradients can be addressed by gradient clipping [27]. Gradient clipping works as the gradient does not specify the optimal step, but rather the optimal direction to make a step in.

6.1.2 The ReLU activation function

The problem of vanishing and exploding gradients comes from the simple fact that the derivative of the activation functions has valid values that are not

one. If an activaiton function's derivative have possible values less than one, the gradient vanishes. If the activation functions derivative have possible values of greater than one, the gradient explodes. Ideally we would like an activation function which gives the network the non linearity it needs for optimal learning, and a function which derivative is one. A category of functions which contains possible functions which satisfy these criterions are what's called *non-saturated* functions [97]. A non-saturating function must fulfill the requirement

$$\lim_{x \to \infty} f(x) = +\infty \tag{6.7}$$

Sigmoid is a saturated function as it's limited to finite values. The Rectified Linear Unit (ReLU) is however a non-saturated function which fulfill the desired criterions[56, 97]. The ReLU in its basic form is quite simple. The activation function does not allow for negative values to propagate further by zeroing them out. Positive values are however allowed to propagate further unchanged. The logic of the ReLU is as follows

$$ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0 & otherwise \end{cases} \quad \frac{\partial}{\partial x} ReLU(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$

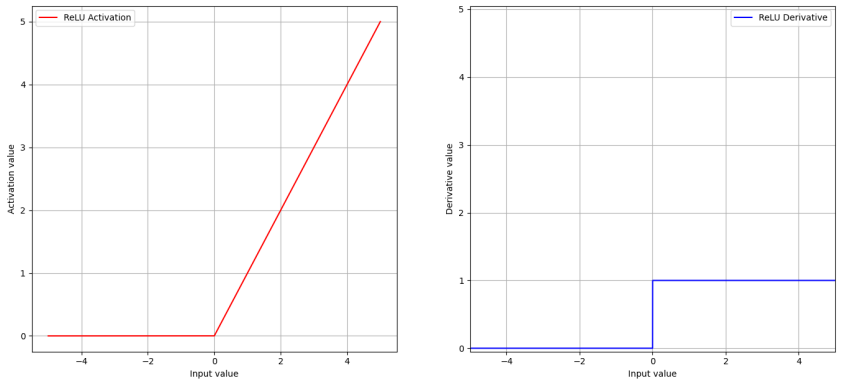


Figure 6.4: Visualisation of the ReLU activation function. The leftmost image shows the activation of any value in the range of $x \in [-5, 5]$. The rightmost image shows the derivative of the ReLU function over the same range of values.

This zeroing out of values makes the activations *sparse*.

$$ReLU\begin{pmatrix}-1\\2\\0\\-5\\4\\7\end{pmatrix} = \begin{pmatrix}0\\2\\0\\0\\4\\7\end{pmatrix} \tag{6.8}$$

Sparsity is desired for a number of reasons. This topic will be tackled more extensively later in the thesis, but its sufficient to know that sparsity is desired as it allows for quicker computation, and works as a form of feature selection mechanism for the network.

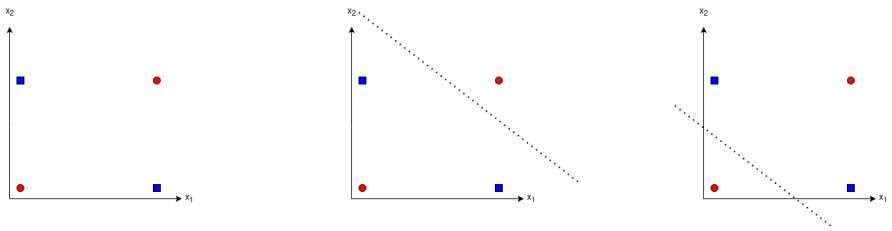
The rectified linear unit has some variants. Leaky ReLU (LReLU) is one of the more well known variants. Introduced in 2013 [51], it proposes that due to the basic ReLU's zeroing of negative activations, neurons which initially fires negative activations might not see any updates through training as no gradient is propagated over the zeroes out activations. Maas et.al [51] rather proposed letting a tiny piece of negative activations to flow onwards, ensuring all the weights found in the network would get some update. The leaky ReLU activation follows the logic

$$LReLU(x) = \begin{cases} x & if\ x > 0 \\ 0.01x & otherwise \end{cases}, \quad \frac{\partial}{\partial x} LReLU(x) = \begin{cases} 1 & if\ x > 0 \\ 0.01 & otherwise \end{cases}$$

Where it should be clear that negative LReLU activations does propagate parts of the gradient backwards.

6.2 Multilayer Perceptrons

A single perceptron has some glaring weaknesses however as was shown as early as 1969 by Minsky and Papert [52]. This weakness can be shown through the fairly simple, intuitive, and well known *XOR problem*. How this problem is classicaly portrayed is shown in figure 6.5a, and two solutions a standard single perceptron will output is found in figure 6.5b and 6.5c



(a) (b) (c)

Figure 6.5: (a) XOR problem visualised. Red and Blue represents two different classes for which we want to find a line which separates the classes perfectly. (b) and (c) shows possible solutions a single perceptron would output. Both with a minimum of one missclassified datapoint.

As mentioned earlier, the perceptron can be interpreted as checking which side of a line/plane a given example finds itself. A single line/plane is not sufficient in more practical cases where data follows complex distributions, and cannot simply be separated by a straight line.

The solution to the XOR problem is fairly simple however. What if a model combined the solutions showed in Figure 6.5b and 6.5c? Then it would be a simple case to check if a datapoint is contained within the region defined by the two lines, or outside it. This can be achieved by combining multiple perceptrons.

Perceptrons can be stacked to map out complex areas such as the one shown in the XOR problem. Figure 6.6 shows a network which is capable of solving the XOR problem. By analysing the output pair (v_1, v_2) the decision function will depend on their combined signs. It's described as follows

v_1 / v_2	Positive	Negative
Positive	Red	Blue
Negative	Blue	Red

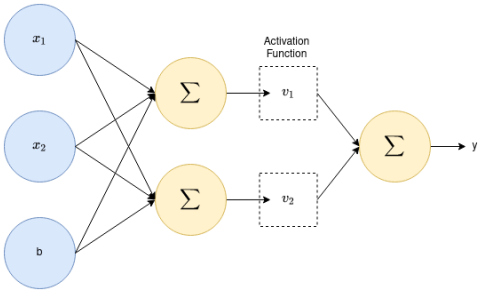


Figure 6.6: Example of a network capable of solving the XOR problem by stacking two perceptrons in a single layer

But following such a decision function is unwanted. A more desired network would spit out a classification, not more data which needs to be interpreted. This can be solved by adding a second layer which would learn to do the classification in the table above. The output would maybe be binary, representing the two classes.

Each layer added onto a perceptron network adds more complexity to the mapping it can do. It also makes the discrimination process more abstract. The forward pass is still quite simple when put into matrix form. Assume a layer l contains i unique perceptrons, or nodes as its more normally refer to. Each node in layer l has its own weight vector, $\mathbf{w}_i^{(l)}$, which put into matrix form makes the $\mathbf{W}^{(l)}$ matrix. The $\mathbf{W}^{(l)}$ is defined as

$$\mathbf{W}_i^{(l)} = \begin{bmatrix} \mathbf{w}_0^{(l)} & \mathbf{w}_1^{(l)} & ... & \mathbf{w}_i^{(l)} \end{bmatrix}$$

Where each $\mathbf{w}_i^{(l)}$ is a columns vector which also contains the nodes respective bias term $b_i^{(l)}$. If it's not clear how a weight vector can incorporate a bias term, it's done by adding a extra feature to both the weight, and activation vector. The weight vectors new feature is the bias term, and the activation vectors new feature is a 1. This is equal to adding the term manually when doing the dot product with unmodified vectors.

$$\mathbf{w}_i = \begin{bmatrix} w_1 \\ w_2 \\ ... \\ w_n \\ b \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ ... \\ z_n \\ 1 \end{bmatrix} \tag{6.9}$$

The forward pass for a layer l is then be calculated as

$$\mathbf{v}^{(l+1)} = \mathbf{W}^{(l)T}\mathbf{z}^{(l)} = \begin{bmatrix} \mathbf{w}_0^{(l)T}\mathbf{z}^{(l)} \\ \mathbf{w}_1^{(l)T}\mathbf{z}^{(l)} \\ ... \\ \mathbf{w}_i^{(l)T}\mathbf{z}^{(l)} \end{bmatrix} = \begin{bmatrix} \sum_j w_{0,j}^{(l)} z_j + b_0^{(l)} \\ \sum_j w_{1,j}^{(l)} z_j + b_1^{(l)} \\ ... \\ \sum_j w_{i,j}^{(l)} z_j + b_i^{(l)} \end{bmatrix} \tag{6.10}$$

where

$$\mathbf{W}^{(l)} \in \mathbb{R}^{j \times i}, \quad \mathbf{z}^{(l)} \in \mathbb{R}^{j \times n}, \quad \mathbf{v}^{(l+1)} \in \mathbb{R}^{i \times n}$$

Note that the input $\mathbf{z}^{(l)}$ can have multiple column vectors as input, each being their own datapoint. The column dimension n defines this, where n is defined for all positive integers $n \in [1, \infty]$. However for the sake of simplicity we'll continue with the vector input example.

All the notation might make this seem messy, but it should be obvious that each node outputs a single value as they are all individual perceptrons. The final layer output is the elementwise application of the activation function $a(\cdot)$

$$z_i^{(l+1)} = a(v_i^{(l+1)}) \tag{6.11}$$

$$\mathbf{z}^{(l+1)} = \begin{bmatrix} a(v_0^{(l+1)}) \\ a(v_1^{(l+1)}) \\ \dots \\ a(v_i^{(l+1)}) \end{bmatrix} \tag{6.12}$$

Note that there are two special values for $\mathbf{z}$. For the first layer we have $\mathbf{z}^0 = \mathbf{x}$, where $\mathbf{x}$ is the input examples, and the final layer $\mathbf{z}^{final} = \hat{\mathbf{y}}$ which is the networks predictions.

To summarise: for any layer l with an arbitrary number of nodes, the layer output is calculated as

$$\mathbf{z}^{(l+1)} = a\left(\mathbf{W}^{(l)T}\mathbf{z}^{(l)}\right) \tag{6.13}$$

6.3 Calculating the MLP gradient

Lets consider a MLP network of l total layers, which takes an input $\mathbf{x}$, and predicts a classification $\mathbf{y}$ making the network effectively mapping the examples $\phi : X \Rightarrow Y$. The predictions are used to calculate some loss $\mathcal{L}$, which we want to propagate backwards into the network, and calculate some change to the parameters. For aesthetic reasons, we'll use $\hat{\mathbf{y}} = \mathbf{z}^{(l)}$. For a weight vector $\mathbf{w}$ in layer k in node i, the gradient can be found by the chain rule as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i^{(k)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial a} \frac{\partial a}{\partial \mathbf{v}^{(l)}} \frac{\partial \mathbf{v}^{(l)}}{\partial \mathbf{z}^{(l-1)}} \frac{\partial \mathbf{z}^{(l-1)}}{\partial a} \dots \frac{\partial \mathbf{v}_i^{(k)}}{\partial \mathbf{w}_i^{(k)}} \tag{6.14}$$

Where the pattern found in the first few factors repeats until the desired layers is reached, in this case layer (k). Analyzing some of these derivations, we find for any layer

$$\mathbf{z} = a(\mathbf{v}) \rightarrow \frac{\partial \mathbf{z}}{\partial a} = \mathbf{1} \tag{6.15}$$

$$\frac{\partial a}{\partial \mathbf{v}} = a'(\mathbf{v}) \tag{6.16}$$

$$\mathbf{v}_i^{(l)} = \mathbf{w}_i^{T(l-1)} \mathbf{z}_i^{(l-1)} \rightarrow \frac{\partial \mathbf{v}_i^{(l)}}{\partial \mathbf{z}_i^{(l-1)}} = \mathbf{w}_i^{(l-1)} \tag{6.17}$$

$$\frac{\partial \mathbf{v}_i^{(l)}}{\partial \mathbf{z}_i^{(l-1)}} = \mathbf{z}_i^{(l-1)} \tag{6.18}$$

Inserting into Equation 6.14, the gradient for a weight vector in layer k is found as follows

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i^{(k)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} a'(\mathbf{v}^{(l)}) \mathbf{w}^{(l-1)} ... \mathbf{z}_i^{(k)} \tag{6.19}$$

Now recall the update equation for gradient decent, Equation 5. For each weight vector in the network the following update will take place

$$\mathbf{w}_{i,t+1}^{(k)} \leftarrow \mathbf{w}_{i,t}^{(k)} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i^{(k)}} \tag{6.20}$$

Chapter 7

Convolutional Neural Networks

Convolutional neural networks, or CNN for short, is a type of neural network which attempts to mimic the multilayer perceptron algorithm, but through the application of the convolution operator [27]. Much like a MLP network, a CNN networks output depends on the preceding input as well as the "nodes" weights. Unlike the MLP network, where each node had a unique weight attached to eac h input feature, the weights of a CNN network is contained solely within the convolution filter, and is applied to the input through the convolution operation. The application of weights onto activations via the convolution operator has some benefits. The number of parameters in a CNN network is massively decreased relative to that of a MLP network. The most commonly used filter in CNN literature tends to be 3×3 filters, whereas a single node in a MLP requires a number of weights equal to the inputsize. The usage of the convolution operator also makes the network architecture excellent at working with grid-like datastructures like images, time series, etc.

A small detail to note is no deep learning engineer/scientist use convolution operations, but rather cross-correlation. This is not because of a conscious choice the engineer or researcher makes, but rather its a choice made for them. Most large machine learning frameworks (Pytorch, TensorFlow) has implemented it in this way. This does not really matter however, as in practice networks achieve the same results weather using cross-correlation or convolution. The weight matrices that are learned will contain the same values, but be flipped relative to the other operation.

This chapter will go into detail of how a convolution is done, how this

can be applied to a neural network setting, how to train such a network, and a CNNs benefits and shortcomings.

7.1 The Convolution Operation

The convolution operator is one of the most effective operations to use when dealing with grid-like data. Convolutions are therefore commonly used in image data, text, video etc. This makes a method based on the convolution operation a natural choice for the work done in this thesis. Mathematically a convolution operation can be interpret as how two functions combined creates a new third function. Consider two matrices. Input matrix $\mathbf{I}$ of size $N_I \times M_I$ and matrix $\mathbf{W}$ of size $N_W \times M_W$. The convolution between matrix $\mathbf{I}$ and $\mathbf{W}$ is written as

$$(\mathbf{I} * \mathbf{W})(x, y) = \sum_i \sum_j \mathbf{I}(x - i, y - j)\mathbf{W}(i, j) \tag{7.1}$$

The $\mathbf{W}$ matrix in this case is what is referred to as the kernel or a filter, and the output is referred to as a feature map. In a convolution neural network this kernel will contain the networks weights. This thesis will from here on out refer to these weight-matrices as kernels. The variables (i, j) is restricted on the size of the kernel. The valid indices for any convolution operation can be said to be indexes where each kernel weight is matched up with an input value. These indices will always be those where all the following are true

$$\begin{aligned} 1 \leq x - i \leq N_I \quad \textit{and} \quad 1 \leq y - j \leq M_I \\ 1 \leq i \leq N_W \quad \textit{and} \quad 1 \leq j \leq M_W \end{aligned}$$

With these restrictions placed on the operator, naturally one might imagine that all convolutions ends with a output that is smaller than the input. While this is technically correct, it is possible to manipulate the input in such a way that the output is either the same size or larger than the input. How the input is manipulated is differentiated between weather one wish for the output convolution to be smaller, equal, or larger than the input in the spatial dimension (height and width). These methods go by the names *valid*, *same*, and *full*.

- **Valid**: The *valid* convolution is the "natural" convolution. The input has not been manipulated and as such the output will be smaller than the input. The output dimension will be of size $(N_I - N_W + 1) \times (M_I - M_W + 1)$ when considering eq. 7.1.

- **Same**: The *same* convolution type adds zero-padding to the input in order to make the number of valid indices equal to the number of input indices. The required zero-padding in each dimension can be written as $p_N = \frac{N_W - 1}{2}$, $p_M = \frac{M_W - 1}{2}$. If the kernel is not odd-sized in each dimension one might experience a small shift in the output. This assumes a stride of one

- **Full**: It is possible to apply the convolution operator to some input to increase the size of the dimensions in the output. When doing a full convolution we zero-pad with $p_N = N_W - 1$, $p_M = M_W - 1$ on all sides. This will in turn increase the dimension of the output relative to the input. This assumes a stride of one.

There are other ways of manipulating the output size resulting from a convolution [15]. These three represents decreasing, same, and increasing the dimension of the input. A much more general formulation of the resulting dimensions from a given convolution operation would be

$$d_{out} = \frac{d_{in} + 2 * padding_d - dilation_d * (kernelsize_d - 1) - 1}{stride_d} + 1 \qquad (7.2)$$

Where d represents the dimension, either height or width, and the subscript denotes the effect in that dimension (e.g. zero-padding height wise might be 2, but in the width dimension it might be 1). It's possible to mix and match padding, dilation, kernelsize and stride to achieve a wide range of output dimension.

CNNs have some desirable properties which was touched on earlier. CNNs leverages three main ideas. Sparse interactions, parameter sharing, and equivariant representation [27], which will be detailed in the following section.

7.1.1 Sparse Interactions

Sparse interactions (also often referred to as sparse weights or sparse connectivity) is an effect of the kernel is smaller than the input image [27]. Sparse

interaction leads to a more localized feature detection, such as edges, lesions or other smaller features one might detect only when looking at a small region of the image, and not the image as a whole. This improves the models statistical efficiency. Sparse interactions also has the advantageous property of reducing the amount of parameters needed for an operation. Having as few weight parameters as possible is hugely beneficial as it saves memory, and makes the output require fewer computations.

7.1.2 Parameter Sharing

Parameter sharing refers to how a convolution kernels parameter is used on the entire input, and is therefore "shared"[27]. Unlike a MLP structure where each input value is associated with a number of weights equal to the number of input features, a kernel is applied to a specific region, then never again. This ensures the trained kernel cannot simply learn to detect for a singular input, but must be more general than what the weights of a MLP structure must. This tend to lead to weights which detect certain features within the small region it's applied to, e.g. edges.

7.1.3 Equivariant Representation

That something is equivariant simply refers to how a change to the input leads to a equal change in the output. This is to say that a function f is equivariant to a function g if $f(g(x)) = g(f(x))$. For a CNN this means a transformation of the data (e.g. apply more contrast to an image) can be done either before or after passing through the CNN network, and both are equivalent.

7.2 Convolution Examples

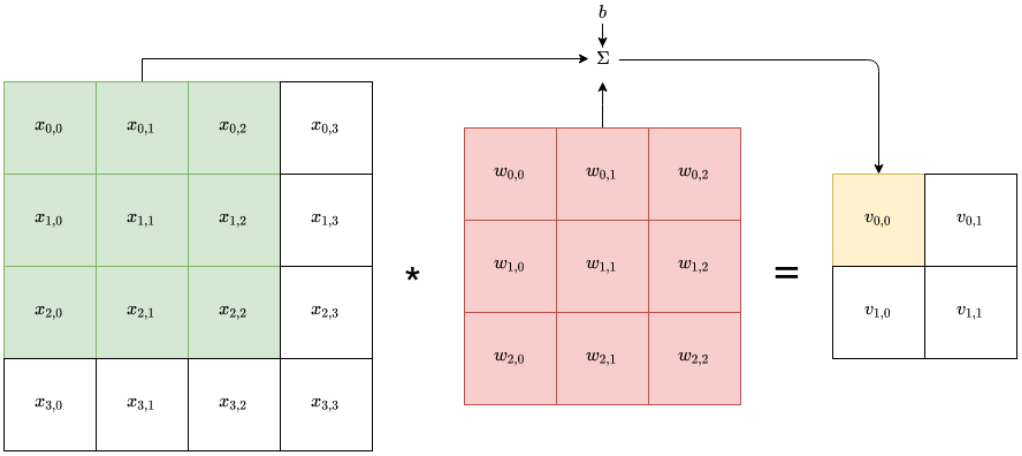


Figure 7.1: Simple convolution example using *valid* convolution

Figure 7.1 shows a simple *valid* type convolution. While formula 7.1 shows us the mathematical formulation of a convolution, it corresponds the operation as sliding the filter over the input matrix. We then multiply the aligning factors and sum to get our output. The output value in figure 7.1 would look like this

$$v_{0,0} = \sum_{i=0}^{2} \sum_{j=0}^{2} w_{i,j} x_{i,j} + b$$

Recall the formulation for calculating the potential for a single perceptron in eq. 6.1. While this sum is over two dimensions, it is effectively the same. Below are similar examples, but for same and full convolutions.

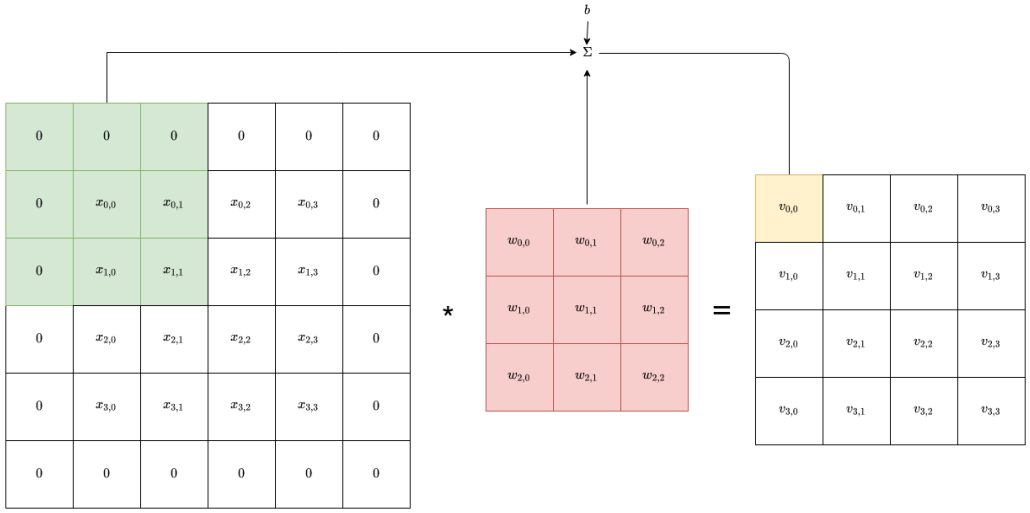


Figure 7.2: A simple convolution example using *same* convolution. The input and output sizes are equal

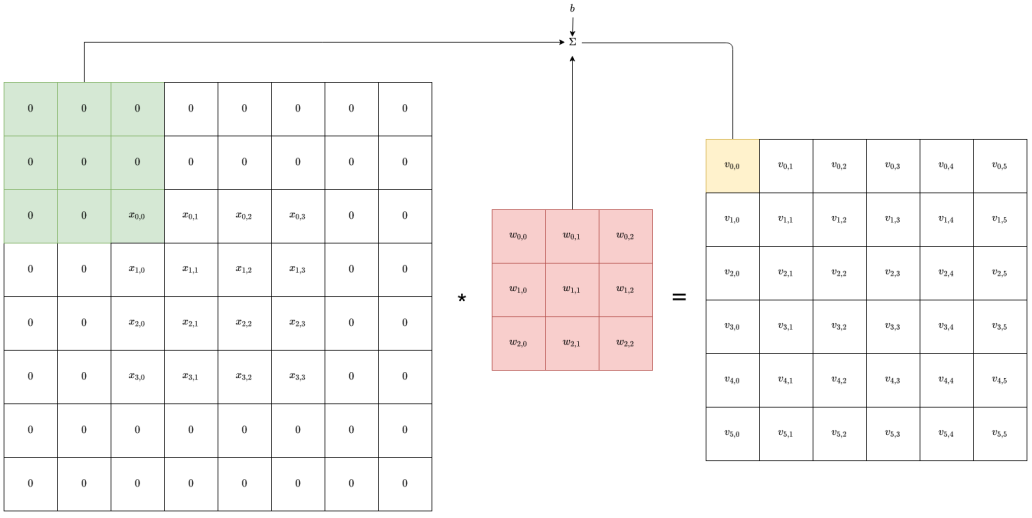


Figure 7.3: A simple convolution example using *full* convolution. The output size has increased by two rows and columns

The examples presented in this section have been in 2D, however most deep learning works with 3D convolutions. However the process of convolu-

tion is effectively the same, just applying a cube kernel instead of a square kernel.

7.3 Pooling

The pooling operation summarises a rectangular neighbour hood into a single value. This value depends on which pooling operation one chooses to use. The pooling operation is generally used after the activation function is performed on the layers potential. This is done mostly for computational gain and not because of some theoretical advantage, any operation on sparse matrices is generally a good idea.

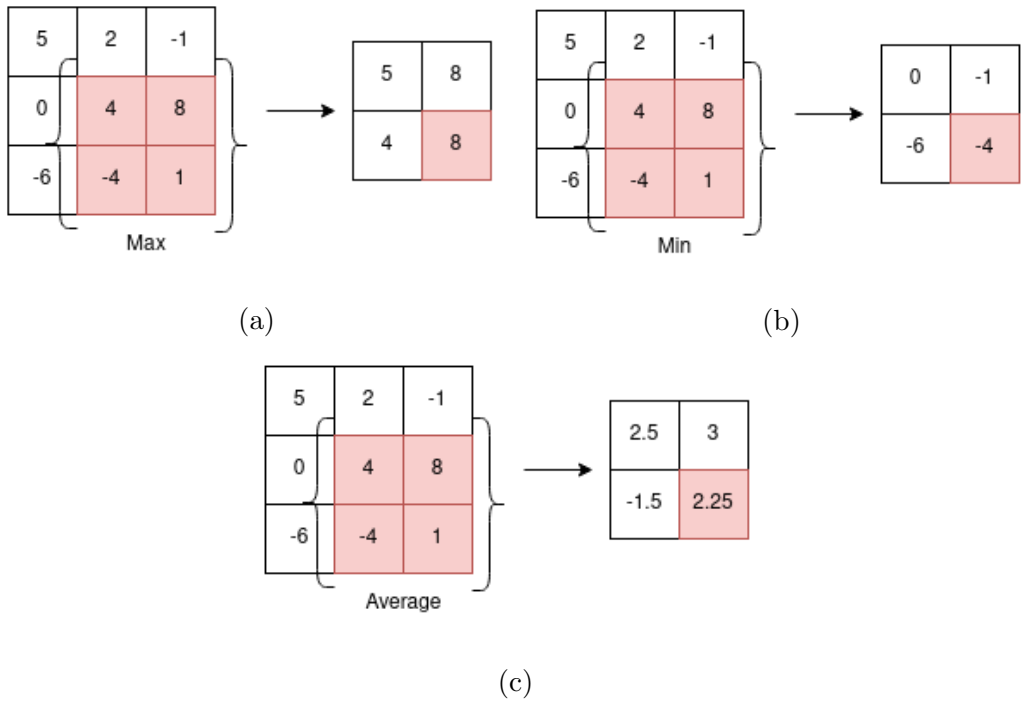


Figure 7.4: An example of some of the 2×2 window pooling methods we have today (a) is an example of maxpooling, (b) is an example of minpooling, and (c) is an example of average pooling

Pooling makes a model approximately invariant to small translation in the input [27]. Translation invariance means that the model is not affected

by small shifts, or displacements in the input image. The output is approximately the same for equal images with small translations.

Pooling also works as a "summariser". By summarising a neighbourhood of values into a single value, the effective perceptive region of a kernel increases. A kernel in the early layers might only be able to detect small features which fits into a 3×3 region. This region is effectively 24×24 after just three standard pooling operations, allowing for detection of much more global features.

Pooling referrs to a multitude of function which abstracts a set of values into a single value. Some different pooling methods are seen in Figure 7.4. The most used pooling function however is the maxpool. It is standard implimentation in several large networks such as VGG [76], ResNet [22][1], Inception V3 [82], and more. The reason for using maxpool is due to its potential of picking up on high frequency features and suppress unimportant features such as constant value regions.

7.4 A basic convolution layer

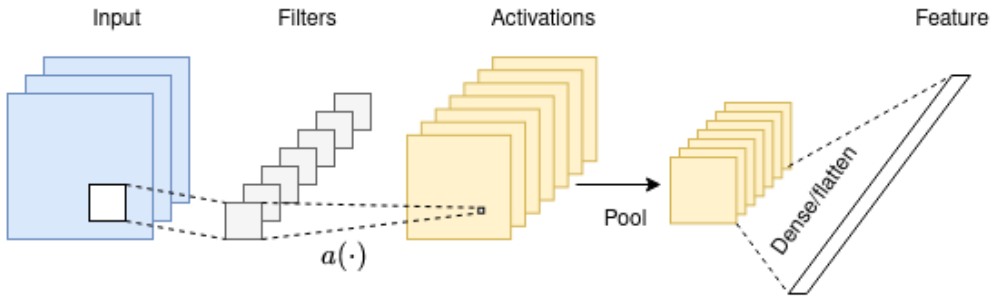


Figure 7.5: Caption

A CNN network learns multiple kernels per layer. Each kernel contains multiple weights which all are learnable. Each kernel is applied to the input via the convolution operation which outputs 2D matrix of potentials. Each potential is subjected to an activation function, such as ReLU, which was discussed in Section 6.1. When a activation matrix is calculated for each

[1]When downsampling

kernel, a pooling operation is applied to the activations. This downscales the data, which is then outputted from the layer, and sent into the next one.

Figure 7.5 shows a slightly different ending however. A CNN summarises the important features of the input, and condenses it into just a few values, spread across multiple channels. Simply flattening these matrices into a vector is a good approximation for a feature, which is what Figure 7.5 shows.

7.5 Calculating the gradient in a CNN

Just how doing a forward pass for a CNN resembles a forward pass in a MLP network, so does the gradient calculation resemble each other. Consider a simple 2D convolution operation as shown in Figure 7.1. Ignoring activations and pooling for the moment, and just considering the example. The weight gradient is calculated as

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}} = \sum_{a,b,i,j} \frac{\partial \mathcal{L}}{\partial v_{a,b}} \frac{\partial v_{a,b}}{\partial w_{i,j}} \tag{7.3}$$

Where the sum over (a, b, i, j) is really a double sum over all valid values, but written as it is for simplicity sake. The valid values are $(a, b) \in [0, 1]$, $(i, j) \in [0, 2]$ for this specific example. If we now insert for the sum $v_{a,b}$ represents in the second factor, an interesting fact appears

$$\frac{\partial y_{a,b}}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \sum_{i,j,m,n} w_{i,j} x_{m,n} = x_{m,n} \tag{7.4}$$

where again the sum is for all valid values, and for specific example we would have $(i, j) = (m, n)$, but (m, n) is used to make the derivation more general. We insert this back into eq. 7.3 and get

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}} = \sum_{a,b,i,j} \frac{\partial \mathcal{L}}{\partial v_{a,b}} \frac{\partial v_{a,b}}{\partial w_{i,j}} = \sum_{a,b,m,n} \frac{\partial \mathcal{L}}{\partial v_{a,b}} x_{m,n} \tag{7.5}$$

It should now be possible to see that calculating the gradient over a convolution operation can itself be expressed as a convolution.

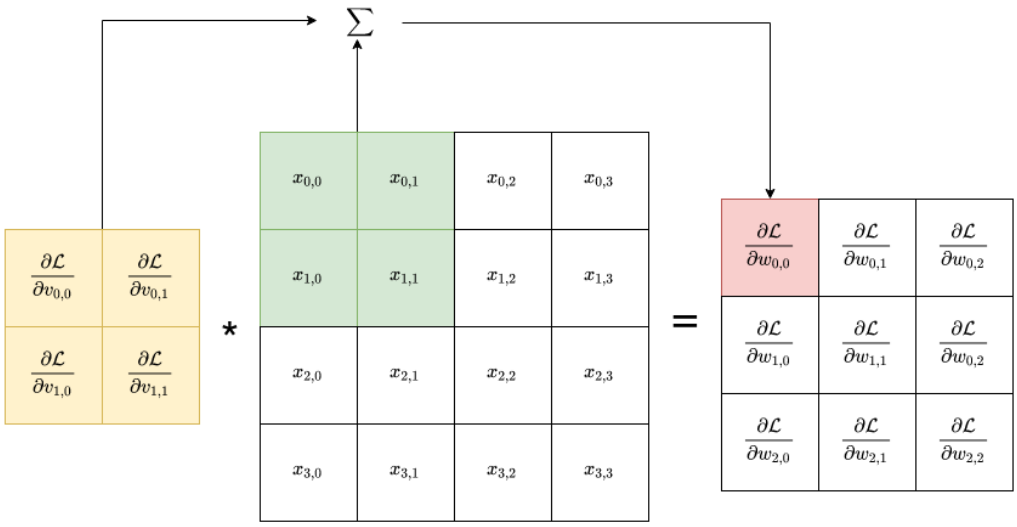


Figure 7.6: A visual guide to calculating the weight gradient for Figure 7.1

We can also express eq. 7.5 over three dimensions as

$$\frac{\partial \mathcal{L}}{\partial w_{i,j,k}} = \sum_{(a,b),(i,j,k)} \frac{\partial \mathcal{L}}{\partial v_{a,b}} \frac{\partial v_{a,b}}{\partial w_{i,j,k}} = \sum_{(a,b),(m,n,o)} \frac{\partial \mathcal{L}}{\partial v_{a,b}} x_{m,n,o} \tag{7.6}$$

$v_{a,b}$ does not add another dimension as this formula only considers a single kernel at the time. The sums again is over all valid values for the operation.

7.5.1 Gradient over the pooling operation

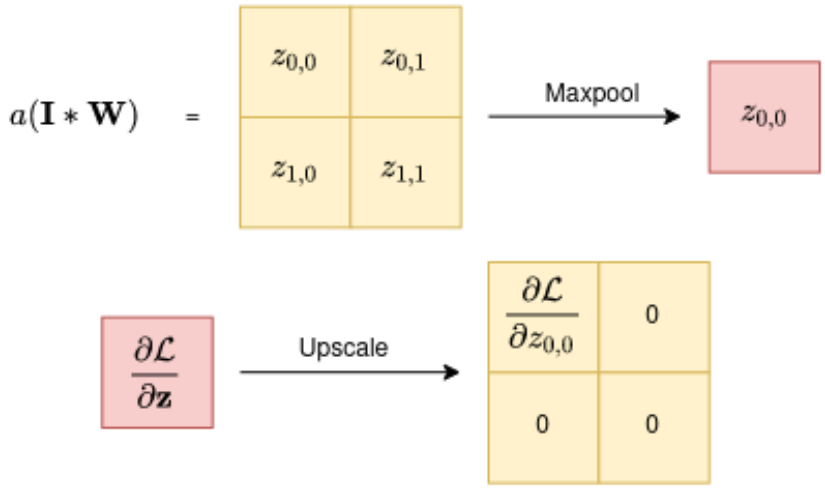


Figure 7.7: An implied forward pass through pooling, and the gradient being propagated

This section will explain how the gradient is propagated over the pooling operation. Consider a maxpool operation case as shown in Figure 7.7. The forward pass outputs four activations, where only one ($z_{0,0}$) is allowed to pass onwards after the pooling operation. All other activations might as well be zeroed out, and do not contribute to the final prediction. So when the loss is propagated backwards, only the max activation aids in the update of previous weights, biases, and further gradient calculation.

7.6 ResNet

This section sets out to explain the residual network architecture, or more widely known as ResNet. This is the network used in this thesis. ResNet was introduced to combat a degradation problem in deep networks where accuracy would get saturated and and degrade rapidly [22].

At the time of the paper, the author had noticed a decrease in accuracy when adding layers onto a deep model. This decrease in accuracy was attributed to the "muddying" of the gradient as it propagated backwards into the network. He et.al [22] solved this problem by creating the residual block which allows for the input to accompany the the feed-forward stream through a skip-connection which bypasses the convolution blocks. This skip-connection performs an identity mapping of the input, and is shown in Figure 7.8.

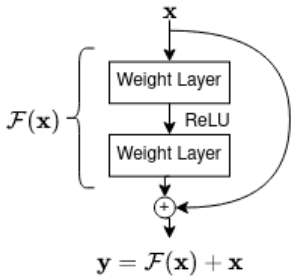


Figure 7.8: A residual block. The main component found in the ResNet architecture

The skip-connection is the key to the ResNet architecture. As they showed during their experiments with an 1202-layer network [22], there was little-to-no optimization difficulty. The skip-connection allows for this by preserving the gradient when propagating the loss back into the network. Lets consider the residual block as shown in Figure 7.8. The gradient over the block is calculated as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \tag{7.7}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial}{\partial \mathbf{x}} (\mathcal{F}(\mathbf{x}) + \mathbf{x}) \tag{7.8}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial \mathcal{F}(\mathbf{x})}{\partial \mathbf{x}} + \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \tag{7.9}$$

Carrying the gradient further back without "muddying" it, lets the earlier

layers learn lower semantic features. This would not be possible without the skip-connection as the gradient would be to abstract.

As clearly shown by Li et.al [37], the skip-connection is not too relevant for shallower networks, but as depth increases it becomes invaluable to keep the loss-landscape from turning non-convex and chaotic.

The residual block is the building block of the resnet architecture. Different resnets tend to have a number associated with them, e.g. *ResNet18*, *ResNet34*, and *ResNet101*. These numbers refer to the number of convolution layers found within the model, or *weight layers* as shown in Figure 7.8.

For the experiments in this thesis we will the ResNet network, with varying depth. Which specific network beings used will be detailed in the experiment section. It is however used as the backbone of our proposed multi-stream model.

Chapter 8

Regularization

A key challenge in deep learning is for the model to not only perform well on the training and validation data, but also perform well on new data [27]. The ability to perform well on previously unseen data is known as generalization. Generalization is said to be high when the model approximates the true underlying data distribution. Deep learning models consists of potentially millions of learnable parameters which can easily overfit to less complex data. It's therefore important to constrain the model in order to make this harder. This is called regularization, and this section will explain some common regularization techniqes.

8.1 Batch Norm

Batch normalization is not strictly a methods for regularization. It was originally introduced as a way to reduce internal covariate shift (ICS)[28], but that they found it to both enable the usage of higher learning rates and that it helped the network regularize somewhat.

Batch normalization works by transforming the output of a layer by normalizing it. Assume a CNN layer outputs some data $X \in \mathcal{R}^{N \times C \times H \times W}$. Over a minibatch $B = \{x_1, ..., x_m\}$ the batchwise mean and variance is calculated

as

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{8.1}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{8.2}$$

$$\tag{8.3}$$

over the channel (C) dimension for the entire mini-batch. The batch mean and variance is used to normalize the inputs as such

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{8.4}$$

The normalized data is finally scaled and shifted to find the output of the batchnorm layer

$$y_i = \gamma\hat{x}_i + \beta \tag{8.5}$$

The output is scaled and shifted with two parameters (γ, β) which are learnable during training. The idea behind this is to allow some leeway for the network. It should not be forced to normalize the data, but can instead learn a spectrum ranging from completely normalized output to a non-affected output. Note that if $\gamma = \sqrt{\sigma_B^2 + \epsilon}$ and $\beta = \mu_B$ the batchnorm layer becomes an identity mapping. This should in theory allow for the network to dynamically find the best form for normalization for each of the layers. This seems to be the case to an extent. Muhammad Awais et.al [5] experimented with fixed (γ, β) values, and found very little in terms of this affecting the network in any way. They showed however that these two hyperparameters tend to stay close to their initial values after completed training. 95% of values fell within the ranges ($\gamma \in [0.68, 0.93], \beta \in [-0.02, 0.45]$), and the initial value being $\gamma = 1$, $\beta = 0$.

Since the release there has been some debate on weather the original authors where right in their finding as later a paper was released by Shibani Santurkar et. al [67] which attributed the effectiveness of batch normalization to a smoothing in the optimization landscape rather than reducing ICS. This paper did not find the strong evidence for reduction of ICS as Sergey Ioffe et.al [28] found. This paper was refuted by Muhammad Awais et.al [5] which

reaffirmed the theory that batch normalization reduces ICS and refuted the hypthesis of Santurkar et. al [67]. This thesis will not try to show which side of this ongoing debate is right. The main idea which should be taken from this section is that batchnorm help models regularize, and increase performance. The exact reason for batch norm working is not exactly known as there is an ongoing debate about the effects of batch norm during training.

The two next subsections will give some more insight to the two main hypothesis to why batchnorm works.

8.1.1 The case for landscape smoothing

This position was originally proposed by Shibani Santurkar et.al [67]. They showed that adding noise from unique distributions for each batchnorm occurrence they would see a severe covariate shift. Their results showed that while covariance shift occurred, they saw an almost non-existent reduction in performance against normal batchnorm, while the noisy batchnorm model still outperformed a normal network significantly. While not finding satisfying results for the original hypothesis put forth by Sergey Ioffe et.al [28], they observed a smoothing in the loss landscape. The paper concluded that the reparameterization of the underlying optimization lead to an stabilization in said optimization.

8.1.2 The case for CSI

The original batchnorm paper by Sergey Ioffe et.al [28] defined *Internal Covariance Shift* as the change in distribution of network activation's due to a change in network parameters during training. This paper as we know showed that the network trained faster by allowing higher training rates, generalized better, and generally increases performance. This paper attributed this to the reduction in *Internal Covariance Shift* which leads to more behaved gradients. However, as the previous section stated; They tested this hypothesis and found it lacking. Their tests where put into question however by Muhammad Awais et.al [5] as they found if you push the noise into distributions with higher values of standard deviation you see not only a reduction in performance, but a crippling effect on the networks performance.

8.2 Weight decay

Machine learning algorithms tends to be quite complex. Models need to have a degree of complexity as real life data tends to be comparatively complex. Complex models are prone to overfitting however, making the model useless. A solution to this can be to make the model less complex, that is to reduce the number of parameters. This has the consequence of limiting the capabilities of our network to properly learn and represent data. This will likely make the model underfit rather than overfit. One solution to this problem is weight decay [27, 33], also known as L2-regularization to some.
Weight decay regularizes the model by favouring small weights over larger ones. The idea being that large/dominant weights should not exists, rather the model in its entirety should be used. The technique itself modifies the loss term as

$$\mathcal{L}_{tot} = \mathcal{L}_{pred} + \alpha \sum_{i=1}^{n} w_i^2 \tag{8.6}$$

Where $\mathcal{L}_{pred}$ is the loss from predictions on training data, and α is a hyperparameter that weights the weight decay term. If this is set to high we can expect to see under fitting. Optimizing an algorithm based on this loss is also known as "Ridge regression". When minimizing the loss, the added term will only be minimized when there are no dominant weights. In an ideal case, the weight would be as small as possible and all parameters would be in use.

Calling weight decay and L2-regularization is generally not a problem as the methods are equivalent. This is not the case for the Adam optimizer [49]. For SGD this is no problem, but due to a

8.2.1 L1-regularization

There is also a regularizer called L1-regularization, but also known as LASSO [27]. Similarly to the L2 version, the L1 adds a penalizing term to the loss function that will enhance features that stand out, and reduce the weights of unimportant features. The modified loss function is written as

$$\mathcal{L}_{tot} = \mathcal{L}_{pred} + \alpha \sum_{i=1}^{n} |w_i| \tag{8.7}$$

When minimizing this loss, the ideal would be for the weights to become sparse. This would minimize the sum of the absolute value of the weights. Sparsity is desired for computational gains mostly.

8.3 Dropout

There is a strategy in machine learning known as model averaging [27]. The idea is to train ensembles of models in parallel and have them all give opinion on the final prediction on some test data. This works well as one can reasonably assume different models will not make the same error when predicting on test data. So if the first model has some problems when predicting class C_i, the other models can make up for the lackluster predictability of the first model in this specific area. This strategy does not scale well however as the computational cost massively increases with each network added. Srivastava et. al [80] provided a computationally inexpensive alternative which achieves the same effect; This method is called *dropout*.

Dropout is a regularization technique that adds a random chance to drop a node in a given network. Dropping a node in practical terms means multiplying the output by zero. No output/activation from a node is equivalent to the node not being a part of the network at all. A node has a chance to be dropped each minibatch during training. But why drop nodes in a network? There are several benefits to doing so. Dropout prevents overfitting by making sure different nodes does not become dependent on the input of a few, or a singular preceding node. This allows for greater useage of the parameters in the network, and better generalization. Dropping nodes is also analogous to training multiple smaller networks, also called sub-networks. Each minibatch we sample a sub-network by dropping nodes, where the drop-probability p is user defined. Figure 8.1 shows an example of a main network, and some different subnetworks that might be sampled during training.

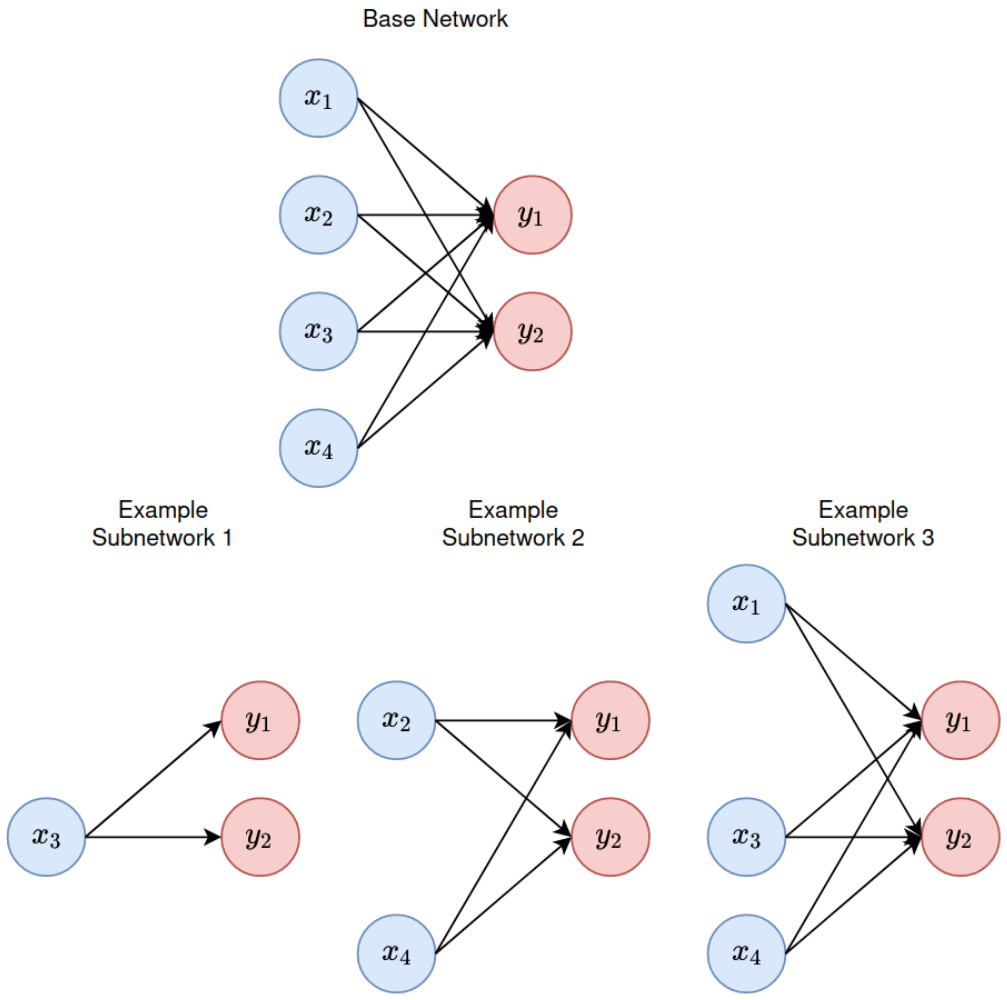


Figure 8.1: Examples of how different subnetworks looks like during training

During testing we do not perform dropout however as we want the best possible results which should in theory come from using the entire network. We instead scale the weights with the probability of retaining the node. This is done so the expected value between training and testing time stays the same.

8.4 Data augmentation

One way of improving a ML model is to introduce more data. Annotating data is expensive, both in time and cost, so there is motive to find some way around not having enough data. Data augmentation is a way of creating false data from the already existing data through augmenting certain aspects if the input data [27, 74]. It's often desirable for the model to become invariant of the transformation applied to the data (e.g. rotation, vertical/horizontal flip, etc). This also helps a network not learn undesirable or unintended features which might not immediately be obvious.

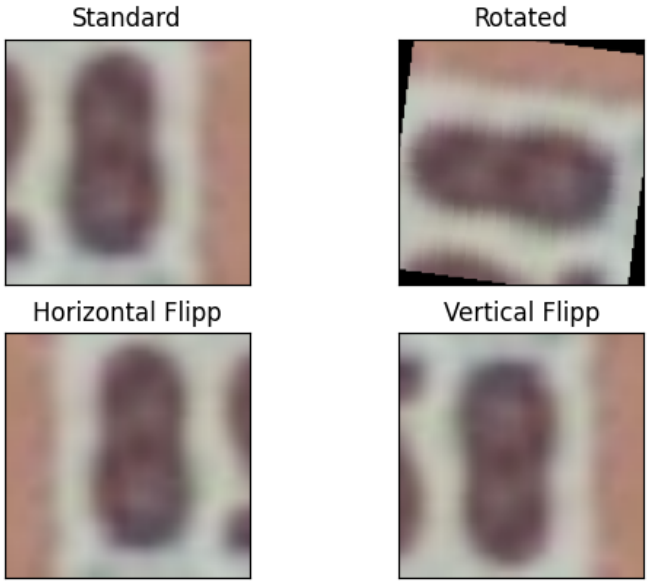


Figure 8.2: Comparison of some different data augmentation, and the standard image before augmentation

Its important to keep in mind what transformation one applies to the data however. Rotating numbers for example can lead to a 9 looking like a 6. There is also the possibility of transforming the data to a point where the model might start to learn on plain bad data.

Due to data augmentation effectively "adding" new data one already have the label for, this is technically adding prior knowledge (adding training data), thereby decreasing the models variance, which is what regularization techniques do [27].

Chapter 9

Domain Adaptation

This section will explain domain adaptation, its purpose, some DA techniques, and go indepth of the ADDA method which will be used in the experiments to solve one of the main challenges of the provided dataset.

The field of domain adaptation seeks to address the problem known as *domain shift* [92, 96, 32, 94]. Domain shift can be described as a shift in the underlying data distribution. This shift is relative to the underlying training data distribution. A shift in dataset can come from a whole range of sources. Illumination, pose, image quality, resolution and more [92, 32].

Domain adaptation is generally used as an information transfer tool. Often one will have two datasets of the same task, but some feature in the datasets differ to the point were a network trained on one dataset does not perform well on the other. This is due to domain shift. The working theory is that if one could remove, or significantly reduce the domain shift, it is possible to use the label information from one domain on the other. There is no shortage of methods which address this problem, and there is a section later (Section 9.3) which goes into some of the many methods within the field later.

DA differentiates between two domains, named the source and target domain. The source domain refers to the dataset which has the labeled information. The labeled information is then transferred onto the target domain data. The target domain data is the data which is shifted relative to the source domain. The target domain is usually not labeled, or has a few labels per class.

Much like normal end-to-end training, the DA field uses the ideas of supervised, semi-supervised, and unsupervised to describe different DA techniques. Note that these terms refers to the target domain data in this context. An unsupervised DA task has no labels for the target data, but usually have a fully labeled source domain.

9.1 Annotation

This thesis denotes a domain as $\mathcal{D}$. A domain spans some feature space $\mathcal{X}$, with a specific data distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. A domain will have an associated task $\mathcal{T}$, which consists of some feature space $\mathcal{Y}$ and a predictive function $f(\cdot)$ for the specific task.

Domain adaptation mainly differentiate between two domains. The data which is labeled and which was used to train the model in the first place is known as the *source domain*. We use the subscript s to denote that something belongs in the source domain, e.g. $\mathcal{D}_s = \{(x_s, y_s)\}$. The second domain is the *target domain*. The data which the model is intended for is found in the target domain. The data is mostly unlabeled, but some DA methods require a few labeled examples from this domain. We therefore differentiate between unlabeled target data with the subscript ut, $\mathcal{D}_{ut} = \{(x_{ut})\}$, and labeled target data with the subscript lt $\mathcal{D}_{lt} = \{(x_{lt}, y_{lt})\}$.

9.2 General DA theory

Domain adaptation (generally) makes the assumption that the primary task in the source and target domain is equal, that is $\mathcal{T}_s = \mathcal{T}_t$. The word task in this case refers to what classification a network makes. If there are two different dataset with images of numbers between $0 - 9$ where each number is considered a separate class, and two networks are made to classify each image into said classes, then those two networks are performing the same *task*.

We separate domain adaptation into what we call *Homogeneous DA* and *Heterogeneous DA* [92]. The homogeneous DA case refers to the case where the feature domains for both source and target are equal. The domain shift stems either explicitly or mostly from a change in the marginal distribution

of the source and target. The opposite case to homogeneous DA is the heterogeneous case. The feature spaces for the source and target are not equal any longer, that is $\mathcal{X}_s \neq \mathcal{X}_t$. The number of dimensions in the feature space might also vary, that is $d_{\mathcal{X}_s} \neq d_{\mathcal{X}_t}$.

9.3 Key technical approaches

The field of domain adaptation is vast. This section should be viewed as a short summary of the field, with examples of methods and work done in some of the approaches. The intention behind this being to show the vastness of the DA field without going into too much detail with methods irrelevant to this thesis.

DA can be divided into two categories. Multistep, and singlestep. When the source and target distribution does not relate much to each other there is incentive to go through a intermediate domain [92]. This is called multi-step domain adaptation, and is aimed at transfering knowledge between seemingly unrelated data. This works by employing intermediate domains. The model mapping is then transfered onto this domain, which results in greater distribution overlap with the actual target domain. This overlap is important to achieve so discrepancy-based losses can be employed. Some known methods which employees multi-step DA are the *Distant Domain Transfer Learning* (DDTL) [83], and *Deep Learning from Domain Adaptation by Interpolating between Domains* (DLID). The DDTL method uses an encoder-decoder function to add and remove instances to and from the source domain until it has reached the target domain. The DLID method creates its intermediate domain by sampling from both source and target domain, then replacing the source examples over time.

Single step DA is by far the most used and published form of DA [92, 94], and is what we will be using through this thesis. The single step DA does not rely on any intermediate information. Rather it assumes there is a discrepancy between two overlapping distributions which can be reduced. The idea behind this being that when the discrepancy is reduced, the labeled source information is directly transferable onto the target domain. No specific methods will be presented here as all methods discussed from here will be single-step DA methods.

9.3.1 Discrepancy-based DA

Discrepancy-based DA methods refer to methods which attempts to reduce some discrepancy measure [92]. The discrepancy in question is that of the difference between the source and target distributions. This can come in the form of distance between two defined points (usually mean) in the distributions, alignments, or other metrics in which difference can be measured.

Some of the more well known measures are the *Maximum Mean Discrepancy* (MME) which estimated the difference in distribution by the distance between the mean source and target feature mapping. Other discrepancy measures that are being used are the $\mathcal{H}$-divergence [6, 66, 64], Kullback-leibler divergence [64], and many more ways of measuring discrepancy between domains.

Some examples of discrepancy-based DA methods are the Minimax Entropy method, a semi-supervised clustering based DA approach which utilize prototypes with pulls features closer to itself. the CAT (Cluster Alignment with a Teacher) [11] method which uses a "teacher" to force cluster alignment in feature space; and lastly Invariant Latent Space methods such as the one presenter in [23] where the goal is not to align the feature regions, but rather find a latent space where the goal is to minimize the notion of domain variance all together.

9.3.2 Adversarial-based DA

Adversarial-based DA methods refers to a group of methods which fundamentally builds on the GAN method [92]. By having a network attempt to discriminate between features from separate domains, and having the target feature extractor trying to trick this discriminator network it forces the target feature extractor to align its features with the source domain to achieve its given goal.

The adversarial-based approach takes its core idea from the following min-max criterion [92, 94]

$$\min_G \max_D V(G, D) = \mathbb{E}_{\mathbf{x}\sim\mathbf{p}_{data}(\mathbf{x})}[log(D(x))] + \mathbb{E}_{\mathbf{z}\sim\mathbf{p}_z(z)}[log(1 - D(G(z)))] \tag{9.1}$$

where D is the domain discriminator, and G is the generator or the target feature extractor.

A well known method is the Adversarial Discriminative Domain Adaptation (ADDA) method [87], which we'll go into much more detail in the following section; The Domain Adversarial Training of Nerual Network (DANN) [19] which utilises a single network for both source and target data. This network is trained supervised on labeled source data, as well as being trained adversarial on domain labels to force domain alignment in the feature space. The Conditional Adversarial Domain Adaptation (CDAN) [48] method conditions the features on the classifier output in order to perform DA, and bound the target risk to the source risk. This method can add entropy to make the CDAN+E method which simulates a semi-supervised learning scheme in its assumed unsupervised setting. Lastly the Graph Convolutional Adversarial Netowrk (GCAN) [50] method. The GCAN employs a graph neural network architecture which utilises a Data Structure Analyzer (DSA) to generate structure scores. These scores, as well as the features, are then combined to create densely connected instance graphs. This is also combined with a class, and domain criterion to create a solid DA method.

9.3.3 Reconstruction-based DA

Reconstruction-based DA methods refer to methods which utilises data reconstruction in order to improve the DA performance [92]. Such methods tend to ensure that semantic features are kept, while manipulating the input in order to adapt it onto the source domain.

One of the most well known reconstruction-based DA methods is the Cycle-Consistent Adversarial Domain Adaptation (CyCADA) [24]. CyCADA utilises autoencoders which reconstruct target images in the source domain, and vice versa. By enforcing that the reconstructed image has the same feature representation, and that going from target to source to target again does not change anything in the image, CyCADA can ensure a doamin-invariant feature representation is achieved by the model among other methods CyCADA utilizes. Another reconstruction-based DA approach is the Fourier Domain Adaptation for Semantic Segmentation (FDA) [99]. FDA utilises the fourier transform, and replaces the low frequencies of the labeled source images with the low frequencies of the target images. The idea being that training on this transformed data should yield a network which can confi-

dently classify target images without ever introducing a true target image. Switching out the low frequency components keeps the larger semantic features of the image, but "stylizes" the source image in the target image style.

9.4 ADDA

The adversarial disciminative domain adaptation [87] (or ADDA for short) is one of the most well known adversarial approaches to DA. It can be said to be one of the most "pure" implementations of the approach, as it follows the GAN [20] model quite closely. This is an unsupervised DA model, so we have no labels for the target domain. The model takes advantage of the fact that the source and target feature spaces are shared, and their mapping is just shifted relative to each other.

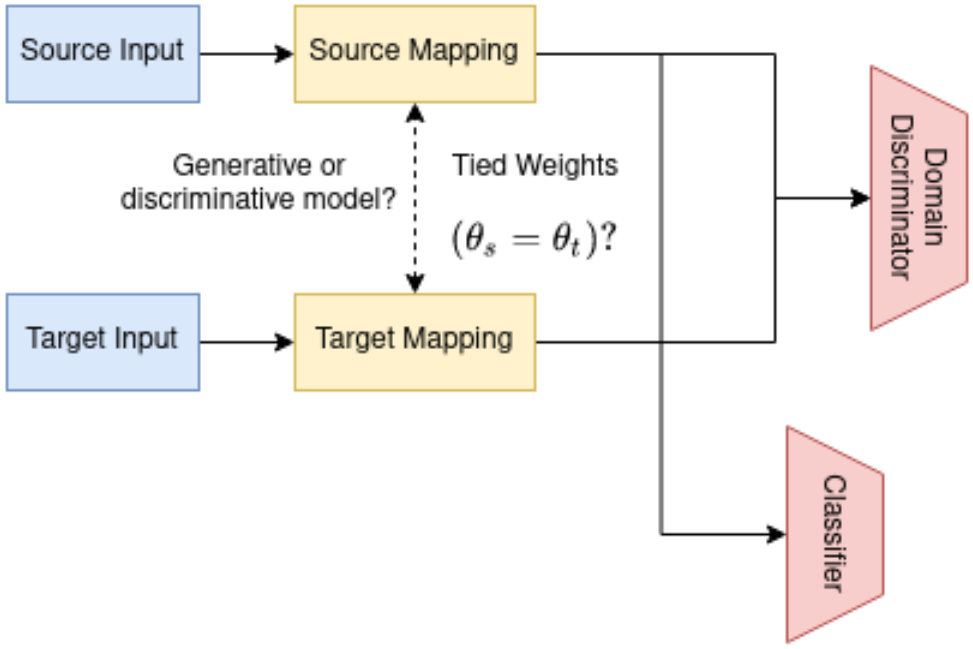


Figure 9.1: The basic idea of the ADDA method.

ADDA is an unsupervised DA technique. This means the target dataset does not have labels available for training. Classification of this data is

therefore impossible without the introduction of some outside information. The source dataset is fully labeled, is used for the same task which is digit classification, but is different in how the data is collected. The source dataset consists of street number digits, while the target dataset consists of greyscale handwritten digits. This difference in data leads to a domain shift.

With the source data fully labeled, our task becomes to somehow transfer the trainable knowledge of the source labels onto the target dataset. The ADDA method starts with training a network on the source data. This network learns a mapping of the data onto a feature space, denoted as $M_s : (\mathbf{X}_s, \boldsymbol{\theta}_s) \Rightarrow \mathbf{f}_s$; where M denotes the source mapping function, $\mathbf{X}_s$ denotes source data, $\boldsymbol{\theta}_s$ denotes the source networks parameters, and $\mathbf{f}_s$ denotes the source features outputted by the function M_s.

After finishing training on the source data, the next step is to transfer the source parameters onto an target network of equal architecture. That is $\boldsymbol{\theta}_s = \boldsymbol{\theta}_t$. Due to the difference in data, it is a fair assumption that the mapping onto the feature space will not be the same. One can of course check this after training, but it's well documented in the DA literature that domain shift will cause a drop in performance. We show this multiple times in the result section (Section 13).

The goal is to make the target network map its features onto the source mapping region, and align with the source features. This will allow the source classifier to work on the target data without ever introducing some target labels.

The method of achieving this goal is inspired by the GAN method [20] of adversarial training. This method can be described by a "cat and mouse" allegory due to the adversarial nature of the method. We introduce a discriminator network. The discriminator is given a single job, which is to discriminate between the domains. Ideally it should be able to tell a feature from the source data from a feature from the target data. This objective is mathematically described by the loss

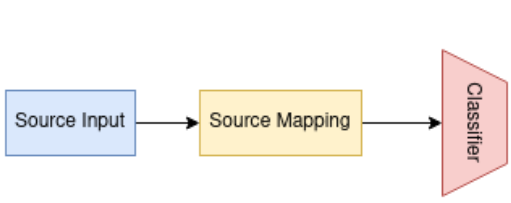
$$\begin{aligned} \min_D \mathcal{L}_{adv\ D}(\mathbf{X}_s, \mathbf{X}_t, M_s, M_t) = &-\mathbb{E}_{\mathbf{x}_s \sim \mathbf{X}_s}[log(D(M_s(\mathbf{x}_s)))] \\ &-\mathbb{E}_{\mathbf{x}_t \sim \mathbf{X}_t}[log(1 - D(M_t(\mathbf{x}_t)))] \end{aligned} \tag{9.2}$$

Next we want to train the target mapper to "trick" the discriminator. Ideally it wants to map features onto the source mapping region, which makes

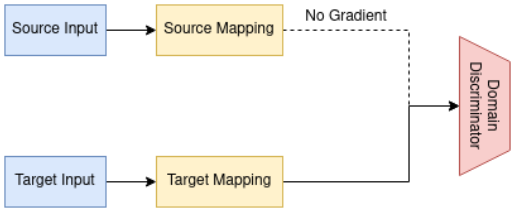
it impossible for the discriminator to

$$\min_{M_s, M_t} \mathcal{L}_{adv\ M}(\mathbf{X}_s, \mathbf{X}_t, D) = \mathbb{E}_{\mathbf{x}_t \sim \mathbf{X}_t}[log(D(M_t(\mathbf{x}_t))] \tag{9.3}$$

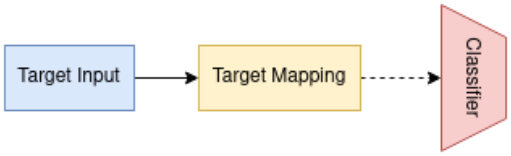
When the mapping of target features completely overlap the source mapping region, the discriminator will have a minimum of 0.5 which is equal to simply wild guessing. The domain adaptation process is now complete, and the source classifier can be used successfully on the target data.



(a) Training of the model in a supervised manner on the source data.



(b) Domain adaptation step. The source model and data is used, but the source model is not updated in anny way



(c) Final step. The target model should now be usable with the source classifier

Figure 9.2: Figure showing the three main steps of the ADDA method as proposed by Tzeng et.al [87]

Adda is part of a generalized DA framework proposed by Tzeng et.al [87]. Figure 9.1 illustrates this framework where models can be designed on a few main choices. Weather to use tied/untied weights, and generative/discriminative models. For the generative or discriminative model question; Up until now only the discriminative network has been explained. It should however, according to Tzeng et.al[87] be possible to use a generative model as to generate samples from the source mapping. This would in many ways resemble the GAN[20] very closely, and it's therefore not a topic this thesis will explore further. The second question posed is weather to use tied or untied weights. Up until now it has been assumed that untied weights have been used due

to it being simpler to grasp the idea behind the ADDA method in this way. Using tied weights simply means that the source model must also be trained further. Now the goal does not become to map target onto the source mapping region, but to rather find some shared mapping $M = M_t = M_s$. This would be something akin to the DANN method [19], which was mentioned in the last section.

9.5 DA in the medical field

The medical field is a prime scene to apply DA to. Domain shift can occur between different equipment, different settings, and just how the data is collected will lead to some bias which shifts the data. It also reduces the labeled training data needed as models can be adapted to local data. The deep learning community has recently taken an interest into DR classification. It would seem that most of the work until now has been on supervised classification, model explainability, and data collection. As this problem gains traction the domain shift problem between instruments have become more apparent [98]. While not a whole lot of work has been done yet on DR domain adaptation, there are a few published studies which shows promise in the application of DA [98, 77, 91, 72]

To the authors knowledge there are two ways of approaching general DR classification. Using optical coherence tomography (OCT) images, and retina images. DA has shown to work on OCT [91]; however this is not relevant for the thesis' experiments. This is due to DR screening images being images of the retina, and OCT being a form of tissue analysis which obtain sub-surface images inside the eye. We are interested in the former. Retina image classification has shown potential for real life applications [90, 21, 68]. As mentioned however there is equipment differences which Yang et.al [98] approached by taking a reconstruction-based approach to the problem, and showed that performance can be improved by adapting the models to the equipment it would have been used on. Shen et.al [72], and Song et.al took a different approach by analysing both retina image, and separate regions of the image. We cannot use these methods for our dataset however, as these methods are all single stream, and cannot model dependency between multiple retina images.

All these methods uses images where the entire retina is fit onto a single image. This is how the large public datasets of diabetic retinopathy images presents their data, but it's not quite how the data is collected for our dataset. A practitioner will take multiple images through the pupil at different angles, and these images will be "stitched" together which will be presented later in the experiment section (See Figure 10.3). Due to having multiple images for a single eye, it's important to adapt a model which can accommodate for this. The model developed for this thesis is a multi-stream architecture that supports domain adaptation while at the same time addresses the challenges in the intro.

Part III

Materials and Methods

Chapter 10

Diabetic retinopathy detection

This section aims to describe the material this thesis works with. Beginning with an in-depth dive into what diabetic retinopathy is, how it affects the eye, and a presentation of the dataset.

10.1 The Eye

To understand diabetic retinopathy it's important to have an understanding of the eye, and how diabetic retinopathy affects the eye.

The iris/pupil and lens area of the eye is of no concern as DR does not affect these areas directly. Inside the eye we have this clear gel-like substance called "vitreous gel". This substance fills the entire inside of the eye and keeps the eyeball in a spherical shape. The wall of the eye is called the retina. This is where the photo receptor cells are located. These cells absorb the light coming into the eye, and reform it into an electrical signal which is sent through the optical nerve into the brain. In the back of the retina there are retinal blood vessels which supply the retina with oxygen rich blood. An illustration of the eye with the elements described here is seen in Figure 10.1.

10.1.1 Diabetic Retinopathy

Diabetic retinopathy is a diabetic complication which affects the eye. It affects roughly one-third of the diabetic population (34.6%) [93], which comes out to about 146 million cases world wide. Vision loss from DR is absolutely preventable. Early detection is however important for practitioners to guide

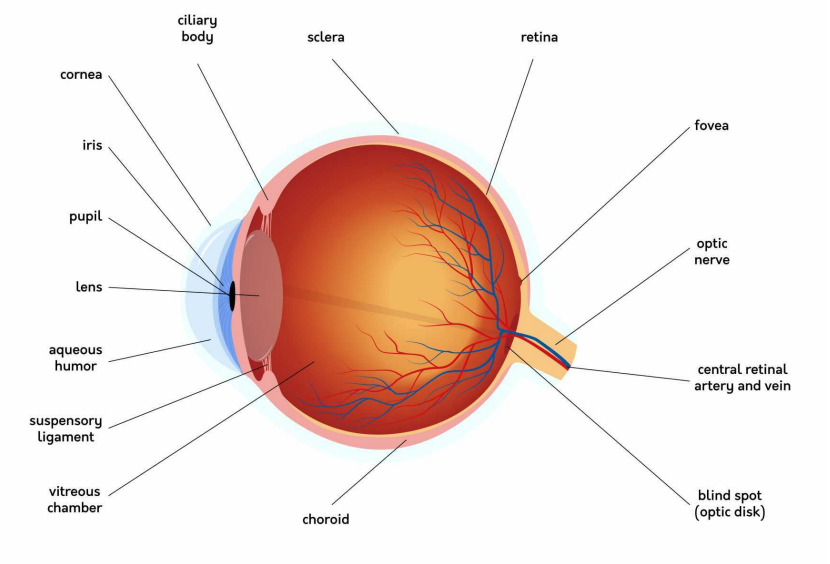


Figure 10.1: Artistic rendition of the human eye.

the patient in the right direction and prevent further degradation. As obesity rises, so do diabetes [93], and we expect there will be more frequent cases of DR appearing. Additionally the world is predicted to lack a vast quantity of health workers in the coming years [46, 44]. This is a huge motivation for creating algorithms to aid in detection of DR, as this will increase diagnostical accuracy, and reduce workload of practitioners [68, 21]. Due to the process of diagnosing DR being mainly though image interpretation, applying deep learning techniques is perfect for this task.

DR is categorised into five different stages. The stages vary from no DR present in the eye, to proliferate DR which is the worst stage of the condition. Each stage has symptoms of varying degrees of severity, but it has been shown that deep learning models can pick up on these symptoms [68]. The five stages, and relevant symptoms are as follows

- **No DR**: This category shown no signs, or the utmost early signs of DR.
- **Mild non-proliferate DR**: The mild non-proliferate DR stage (or Mild DR for short) is the first stage of an eye developing proliferate DR. Non-proliferate simply means there are no new rapidly growing

blood vessles. In this stage we can see the development of microaneurysms. Microaneurysms are small bulges and abnormalities in the already existing blood vessels and can be seen on retinal images as small red dots. During this time, the symptoms tend to cluster and not spread around the eye too much. The amount of microaneurysms can vary from 3 to 30 before the condition worsens.

- **Moderate non-proliferate DR**: During the moderate DR stage parts of the blood vessels can get blocked, and leaks / hemorrhages can occur in the eye. Due to this, some parts of the retina might experience a lack of fresh blood and oxygen. When parts of the retina is starved for oxygen, we say the tissue is ischemic. This ischemic tissue appears white, and due to it looking like balls of cotton we tend to call this area "Cotton wool spots".

- **Severe non-proliferative DR**: In the severe DR stage we might see small irregular blood vessels appearing. This signals the beginning of the proliferate DR stage.

- **Proliferate DR**: The proliferate DR stage is also known as the neovascular DR stage; Neo-vascular meaning that new blood vessels are forming. This is considered the end stage of DR. During this stage we'll see an increase in newly formed blood vessels (therefore proliferate). The problem with new blood vessels forming is the fact that they are extremely weak, and will rupture. This causes fluid and blood to spew out into the eye. This might cause scaring, which when dries up can pull on the retina to the point where we experience what is known as retinal detachment. The newly formed blood vessels can also spew fluid into the vitreous, causing what's known as vitreous hemorrhage. This can at worst hinder light from hitting the retina, impairing the vision of the patient.

Visual examples of these stages is given in Figure 10.4

10.2 Diabetic Retinopathy dataset

The diabetic retinopathy dataset has been provided by Universitetssykehuset Nord-Norge HF (UNN). The two datasets provided where collected in relation

to the sixth and seventh *Tromsø Study* [17]. The sixth study collected the data used for the first *Tromsø Eye Study* [7], and the seventh study collected the data for a new eye study which had not been published as of this thesis. The two datasets are differentiated by T6 (Data from the sixth study), and T7 (Data from the seventh study). The T6 dataset is collected from a random, but representive, population within the municipality of Tromsø with the age group $38-87$. The information of collection for the T7 dataset is not released yet as of this thesis, but it's reasonable to assume the data has been collected in a likewise fashion.

The distribution of the datasets are as follows

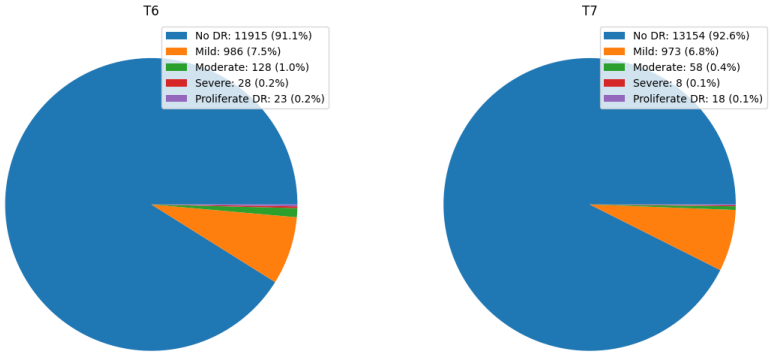


Figure 10.2: A piechart figure showing the class imbalance in the datasets T6 and T7

Dataset	*No DR*	*Mild*	*Moderate*	*Severe*	*Proliferate*
T6	11915 (91.1%)	986 (7.5%)	128 (1.0%)	28 (0.2%)	23 (0.2%)
T7	13154 (92.6%)	973 (6.8%)	58 (0.4%)	8 (0.1%)	18 (0.1%)

Table 10.1: A table showing the number of class-wise examples and the percentage of the dataset they represents.

The datasets with information such as if the eye is the patients left or right eye. Each eye consists of several images. The amount of images vary, but should technically be six images. Some data points are immediately cut

from the used dataset on the basis that they have five or fewer examples. Figure 10.3 shows how images are taken in the retina. Each circle is put into its own image.

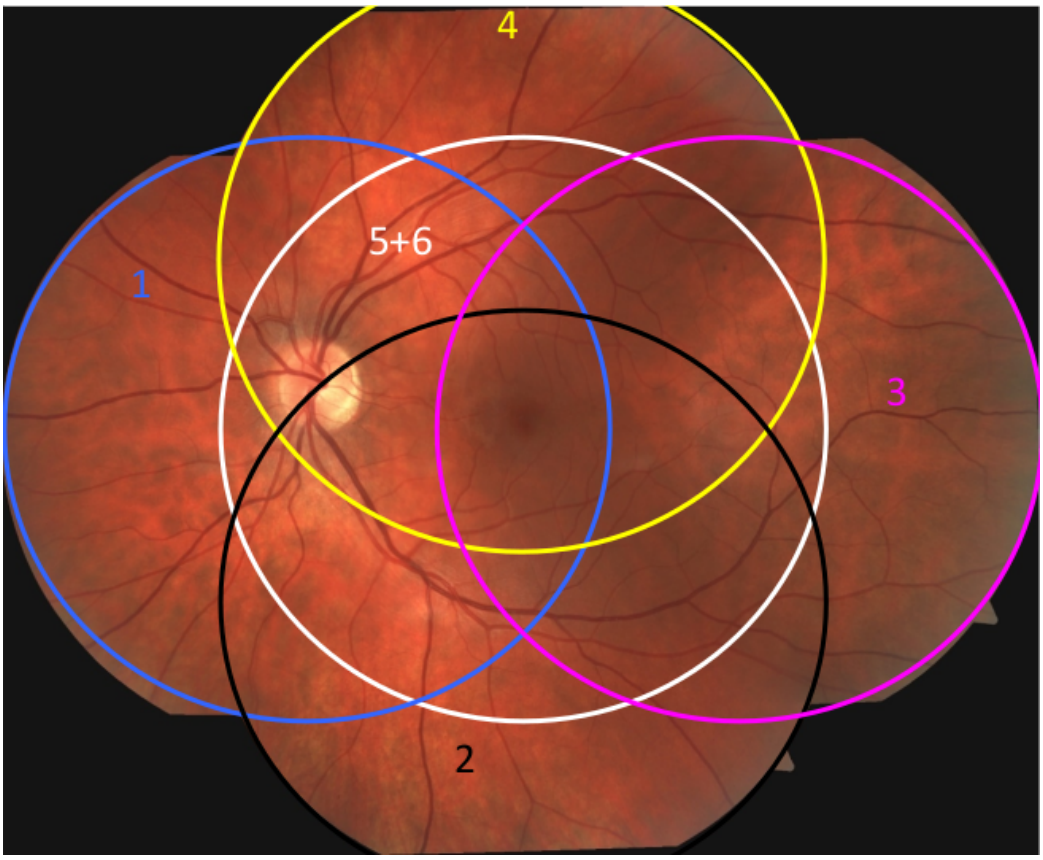


Figure 10.3: How multiple images combined constitutes the entire backside of the eye. Image is used with permission from Geir Bertelsen and is sourced from *Prosedyrebok Øyestasjon Tromsø 6*, an instruction manual for the usage of the *Fundusfoto Visucam 500* machine.

Images 1-5 are taken at a 45° through the pupil, while image 6 is taken at a 30° through the pupil in the same region as image 5.

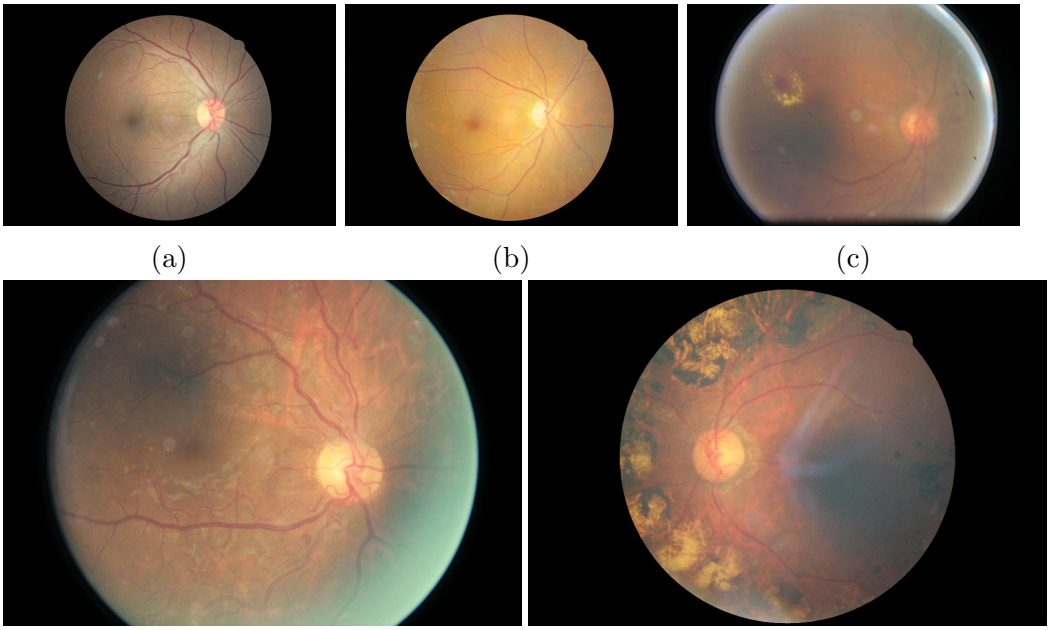


Figure 10.4: One example from each class. Images taken from the public APTOS 2019 dataset. (a) No DR class example. No visible lesions present. (b) Mild DR class example. Has what seems like a lesion down left. (c) Moderate DR class example. Has a small ischemic spot, and what seems like some micro aneurysms. (d) Severe DR class example. Has some cotton wools spots going on, and a small hemorrhage. (e) Proliferate DR class example. Has a massive case of cotton wool spot, and potentially some retinal detachment

Chapter 11

Methods

This section describes the proposed model that has been designed as part of this thesis to address the challenges presented in the "Dataset challenges" section (Section 1.2.1). To summarise the challenges: The datasets provided contains retina images. Each retina consists of multiple images as shown in the previous section. Standard CNNs cannot model dependency between the same-retina images, so we must construct a model capable of this. Each retina should consist of six images, but due to subjects breaking off the DR screening early, or the practitioners taking multiple images due to blurry/bad images, there are some retinas with a lack or excess of images. The images from differing regions of the eye are not structured in any way either, so out model must accommodate for the fact it can have any region as an input. Finally, the datasets is expected to have experienced a domain shift, moving from T6 to T7, and as such we wish to make sure our model is compatible with domain adaptation techniques. To be more precise, the ADDA method has been chosen for this thesis.

To deal with the challenges listed, we've constructed a multi-stream network for DR detection. This network is capable to take as an input a variable number of images for each eye. The inputs are processed individually, and then their features are fused to create a feature vector representing the eye as a whole. This network is constructed in such a way that it should be compatible with most domain adaptation methods.

The coming sections will go into details on the network, how its components address the different challenges, and how ADDA is applied with this

model in mind.

11.1 The multi-stream architecture

The theory presented in this thesis has up until now really only considered single-stream architectures. Single-stream architectures refer to a set of methods which takes in a single type of data (e.g. images, text, audio, etc), and output some result for that specific input, and are by far the most commonly type of architecture. To this end, there is no real dependency between images. One image cannot affect the results of another during prediction. This works for a lot of tasks such as DR detection on single retina images as discussed in section 9.5. This thesis, however, considers multiple images for a single eye, where cross-image dependency is important for a proper classification of the entire eye. For this problem we've chosen to construct a multi-stream model which has the desired properties to classify multiple images, and model dependency across images.

The main body of theory surrounding multi-stream networks is generally done on action recognition tasks [101, 95, 86, 75]. Action recognition is a video detection task where the objective is to figure out which action is taken from a video. In 2014 Simonyan et.al [75] found that paying attention to the temporal component lead to their network learning important context clues, compared to still images. An example is an image of someone seemingly sitting down can also seem like someone getting up from the chair. Adding the temporal component will make weather someone sitting down or getting up much more trivial than when analyzing a still image.

Multi-stream models are models where data is processed in parallel through multiple streams. The information extracted from each stream is often fused together to model dependence between the separate data. This fusion of features creates a final representation which represents the data as a whole. In practice there are few multi-stream models which are alike. They tend to be built-for-purpose. Either made for a specific set of data, a specific task, etc. where such a model is useful.

In order to address **challenge** #1, a multi-stream architecture is a natural solution for the UNN datasets. There are multiple images representing the

same eye. Each image might hold important semantic information which can aid in the construction of a final feature vector representing the entire eye. In the next section we outline the proposed multi-stream model to deal with the UNN datasets.

11.2 The Model

This section sets out to explain the constructed multi-stream model. The model constructed for the task of DR classification is a multi-stream CNN model. It was made with the presented challenges in mind, and is capable of solving said challenges. The flow of the model is shown in the figure below, Figure 11.1.

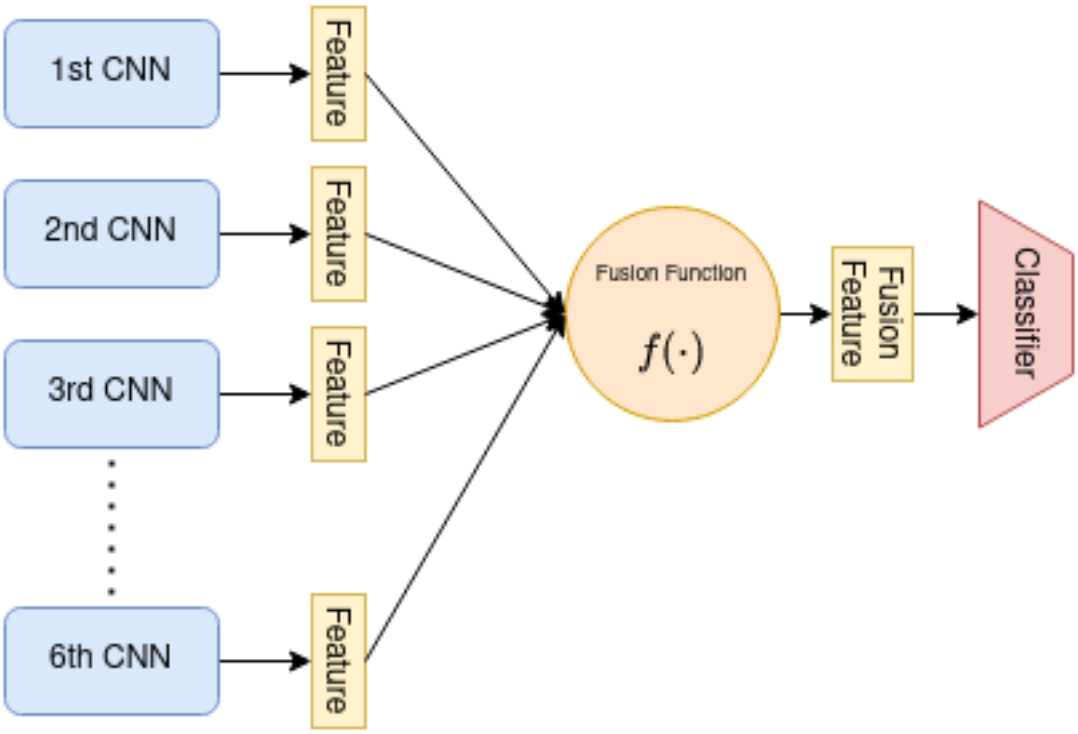


Figure 11.1: Illustration of the network architecture

The figure shown is rather general. The CNN structure, the fusion function, and the classifier are all important components, but can be changed to fit new datasets, needs, etc. In our case we've chosen ResNet, the model discussed in Section 7.6, as the CNN structure.

The model structure reflects the DR dataset. It is build for purpose unlike most off-the-shelf networks which are more generalized in the regard

that they are applicable over multiple tasks.

The network is able to take in multiple separate images, and extract important semantic features from each image. Each stream is fused together through a fusion function, which is classified through a MLP network.

11.2.1 Model intuition

The network adapted for this task is not hard to understand, and the intuition should come easy. This section will also explain the network in terms of the DR dataset as this is the purpose for it being constructed.

The first part of the network sends the data through a CNN in order to obtain image-level features that summarizes the information in one particular image. Assuming the network is trained, the output will be a feature vector which represents the important features in the image. This is then done for each image for a single eye. Each of the individual features vectors contains information that we desire to summarise into a single vector. To obtain our eye-level features that can be used to classify the eye according to our provided label. The independently extracted information is summarised through a fusion function, which we will explain in detail in a later section. This feature vector is classified through a MLP network, and the classification is compared to the eyes ground-truth using the CE[1] classification loss described in Section 4.1.1.

11.2.2 How the model works

This section is meant to give the reader a more practical understanding of how the presented network works. The motivation behind giving this understanding is to strengthen the argument that this network is theoretically compatible with most, if not all domain adaptation techniques. The model presents itself with different streams. However as those who are familiar with training ensembles of networks will know, this is a memory heavy task, due to the fact that we have variable number of images, potentially taken in differing order, and no information of which of the six locations in the eye (see Figure 10.3) corresponds to, each CNN must be able to detect symptoms equally. This in turn effectively makes all the CNN networks share the same

[1]Cross entropy

weights, and the multi-stream network can be achieved with some dimension manipulation (**Challenge** #3)!

Consider a datapoint $\mathbf{X} \in \mathcal{R}^{N \times M \times C \times H \times W}$. The dimensions here can be interpreted as N unique eyes with M corresponding images, where each image has C channels, H rows, and W columns.

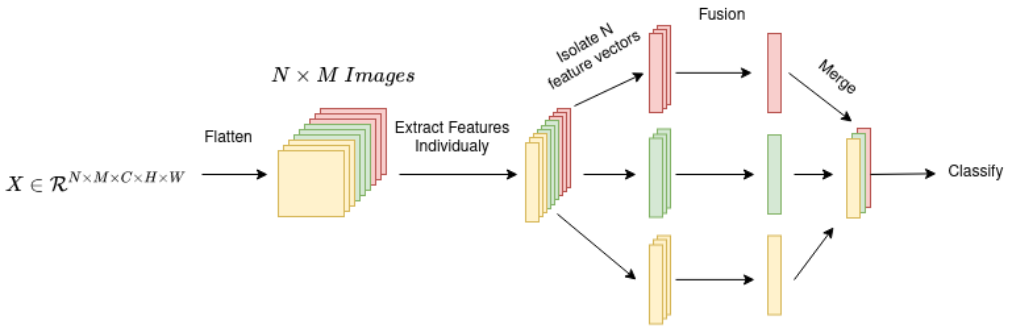


Figure 11.2: Visualisation of how the network process dependent images

In the figure above it's assumed that $N = M = 3$. The spatial dimensions are of no concern to understand the model. The separate eyes are color coded as to make the visualisation easier to follow. The input is flattened across the $M \times N$ dimensions, where images from the same eye are kept next to each other. The data now has the dimensions $\mathbf{X} \in \mathcal{R}^{(N*M) \times C \times H \times W}$, which a single CNN network is more than capable of extracting features from. Each image will have their feature extracted separately by the CNN model. Assuming the feature vectors have l features, the output features will have dimensions $\mathbf{X_f} \in \mathcal{R}^{(N*M) \times l}$. The features will have the order in-which the images where inputted. Knowing this, it is possible to expand the feature vectors into three dimensions. The dimensions will have the dimensions $\mathbf{X_f} \in \mathcal{R}^{N \times M \times l}$. In practice it is possible to apply a fusion over the M dimension, and collapse it. In Figure 11.2 this is shown as isolating the eye-wise feature vectors, but this is just an illustration, and in practice most fusion functions will leave the M dimension being $M = 1$, which is easily collapsible. This will give us one feature vector per eye (N features), which can then be classified and the results can be compared against the eye-wise labels from the dataset. To summarise, let $S = C \times H \times W$, then the change in the data dimensions over the model should be as follows

$$\mathcal{R}^{N\times M\times S} \xrightarrow{Flatten} \mathcal{R}^{(N*M)\times S} \xrightarrow{CNN} \mathcal{R}^{(N*M)\times l} \xrightarrow{Expand} \mathcal{R}^{N\times M\times l} \xrightarrow{Fusion} \mathcal{R}^{N\times l} \xrightarrow{Classify}$$

What has been shown here is that the multi-stream model is in fact a normal CNN structure with a flattening process before inputting the data, and a fusion step between the output features, and classifying the features. Additionally we have shown the network is capable of taking in any number of images per eye, as long as the amount of images (M) is equal[2] **(Challenge** #2**)**! This is turn can allow for the network to be applied to a multitude of different DA methods, as well as other beneficial deep learning techniques and methods, addressing **Challenge** #4. This will be further detailed in Section 11.3. Applicable methods which is of particular interest to the medical imaging field is that of explainability and interpretability methods. This is a topic which will be discussed further in the discussion section (Section V).

11.2.3 Fusion functions

A core part of the proposed model is the choice of fusion function. Any method of combining multiple vectors into a single vector is applicable here, and the choice of fusion can lead to some interesting behaviours in the network. This section will present a few fusion functions, their expected behaviour and pros / cons. The choice of fusion function is directly related to **challenge** #2. Recall that **challenge** #2 requires the development of a model that can handle a variable number of input images. As both the feature extractor (CNN) and the classifier are considering the individual image level features and the eye level features, respectively, the fusion function presents the bottleneck with regards to this challenge.

The concatenate fusion

The concatenate fusion function is one of the most non-compromising fusion functions. While other fusion functions will sacrifice some information in order to create a more total view of the complete data, the concatenate

[2]Note, this is just an implementation choice for ease of implementation, the model generalized to an arbitrary number of images.

simply stacks each feature vector on top off one another. This seems reasonable initially. No information is lost, and all features are considered on equal footing. However there is a scaleability problem. As the number of inputs increases, so must the first classification layer. This directly impacts **Challenge** #2, as a change in the classification layer will mean that the model will not scale to an arbitrary number of images. Dimension-wise, this fusion can be described as

$$\mathcal{R}^{N\times M\times l} \xrightarrow{concatenate} \mathcal{R}^{N\times (M*l)}$$

Another important moment for the concatenate fusion is how susceptible it is to noisy data. Unlike the fusions we will present next, the append fusion has no way of dealing with noisy data being propagated forward with the healthy feature vectors. We can imagine that in some cases this will lead to low confidence in predictions, if not wrong predictions all together.

The mean fusion

The mean function is another intuitive fusion function. This fusion regard each individual feature as equally important, allowing all of them an equal say in the construction of the final feature vector. Depending on the number of streams, this fusion function allows for the creating of a fusion function which is more robust against noise. It is also able to scale to an arbitrary number of images, thus addressing **Challenge** #2. This fusion function however relies on a majority clean images to be effective, as we provide evidence for in the results section. If this is used with multiple feature vectors from noisy data [3], the noise can, and most likely will drown out the features of the healthy data. In a DR setting, this means that patients where one image is mild DR and the remaining No DR, the model might lean towards the majority class position, No DR in this case.

Max fusion

The max fusion is a way to combine only the most prominent features into a representative feature for the combined feature vector. The idea behind this is somewhat more complex than the previous two, but still simple. In

[3]Noisy in this case means data which either is, or appears to be from a different class. This can be a healthy eye image which is part of a diseased eye, an explicit diseased eye inputted as a healthy eye.

computer vision, the extracted features will have high activation's for areas where important semantic features have been detects, such as edges, lesions, and such. Unlike the mean fusion which might muddy out these detected features, the max fusion wishes to extract only the highest feature for a given position over all the feature vectors. Take the following toy example with four individual feature vectors. The final output feature is found by taking the max value over the rows

$$Max(\mathbf{F}) = max\left(\left[\begin{bmatrix}0\\ \mathbf{8}\\ 1\\ \mathbf{8}\end{bmatrix}\begin{bmatrix}4\\ 2\\ \mathbf{8}\\ 4\end{bmatrix}\begin{bmatrix}1\\ 3\\ 7\\ 3\end{bmatrix}\begin{bmatrix}\mathbf{9}\\ 3\\ 6\\ 3\end{bmatrix}\right]\right) = \begin{bmatrix}9\\ 8\\ 8\\ 8\end{bmatrix}$$

In some regards, this fusion function acts much in the same way as a maxpool operation. One of the downsides to this is the potential lack of gradients being propagated backwards to a non-max position in the feature vector, potentially leading to less effective training, due to most features effectively being zeroed out.
The max fusion scales to an arbitrary number of images, and thus addresses **Challenge** #2.

11.3 ADDA

This section explains how the ADDA [87] algorithm is merged with the proposed multi-stream network. Due much in part to the arguments put forth in section 11.2.2, this model should be compatible with domain adaptation methods, and most importantly for us ADDA.

To be more specific, the CNN, which is the feature extractor part of the proposed network can be viewed from both a multi-stream and a single-stream perspective. As many DA approaches, among others the ones discussed in Section 9.3, rely on adapting the feature representation from the target domain to the source domain. We can perform domain adaptation from the single-stream perspective. For ADDA in particular, given the two multi-stream architectures, one for the source, and one for the target domain, we can train the source model in a supervised manner.

We can then take one of the CNN streams (Remember we use shared weights in the streams, making the individual streams identical) from the

target network and align its distribution with the source feature distribution using a discriminator as described in in Section 9.4. Again, the source features can be obtained from a single CNN of the multi-stream source CNN due to the shared weights. Once the target feature extractor has been updated to provide aligned distribution, we can replicate it in our multi-stream architecture and classify target eyes (Consisting of an arbitrary number of images) by feeding the features to the source multi-stream architecture. This all leads into solving **Challenge** #4.

Part IV

Experiments and Results

Chapter 12

Experiments

This section aims to justify and explain why and how the experiments where done. The results of the experiments will be presented in the *Results* section.

12.1 Proof of concept

The DR data was not collected with machine learning, in mind, and due to noisy[1] labels it is difficult to analyze the various properties of the model efficiently. We, therefore, wish to test the model independently of the UNN datasets. We have therefore employed the well known MNIST and SVHN datasets to show how the network can perform in a simple setting. This experiment also sets out to show that the model is compatible with the ADDA method **(Challenge** #4**)**.

There are two main experiments that we do with the SVHN and MNIST datasets. First we show that a multi-stream can domain adapt onto another multi-stream network, where we use SVHN as the source data, and MNIST as the target data. The main goal of this experiment is to show a general multi-stream model can be domain adapted onto another multi-stream model. This experiment also incorporates a model test of robustness against noise. This is done by artificially adding a wrong image to the input stream. We utilise the same implementation as we will use for the DR dataset, and experiment with how the performance degrades going from no noisy inputs,

[1]Noisy in this case means data which either is, or appears to be from a different class. This can be a healthy eye image which is part of a diseased eye, an explicit diseased eye inputted as a healthy eye.

to five noisy inputs. This is done both with noise from the same, but wrong class, and noise from a random wrong class.

The second experiment using SVHN and MNIST is adapting the multi-stream model onto a single-stream network. While the method section explained why this is theoretically possible, we desire to show it empirically too. This is also done in order to further cement that this model has met the criteria to solve **Challenge** #4. The hypothesis being if we can adapt a multi-stream model onto a multi-stream model, and a multi-stream model onto a single-stream model, then the proposed multi-stream model can be adapted onto a network of arbitrary inputsize, which further shows the model can deal with **Challenge** #2.

These experiments uses the same multi-stream architecture as we will use for the DR classification. This is the multi-stream network shown in the methods section. The only difference is we use a simpler ResNet18 architecture as the CNN network when dealing with the SVHN and MNIST datasets. This is mostly due to these datasets being much less complex than the DR datasets. Furthermore the only data augmentation which was performed on the SVHN and MNIST datasets where normalization. MNIST normalized with the mean and standard deviation ($\mu = 0.5, \sigma = 0.5$), while SVHN used ImageNets means and standard deviation ($\mu = [0.485, 0.456, 0.406], \sigma = [0.229, 0.224, 0.225]$).

12.2 MNIST and SVHN datasets

MNIST and SVHN are two well known datasets containing number between zero and nine. The MNIST dataset is composed of 60000 training images of handwritten digits. The SVHN dataset consists of 73257 images of real-life house digits. Examples from each datasets can be seen in Figure 12.1.

MNIST and SVHN are two well known datasets containing numbers between zero and nine. The MNIST dataset is composed of 60000 training images, and 10000 test images. The training images where split into a training set of 50000 images, and a validation set of 10000 images. The validation set is used for tuning the hyperparameters. The SVHN dataset consists of 73257 training images, and 26032 test images. The training set is split into

50000 training images, and 23257 validation images. These dataset splits can be seen in Table 12.1. For all experiments with MNIST and SVHN, we use SVHN as source data, and MNIST as target data.

(a) Ten MNIST examples from each class.

(b) Ten MNIST examples from each class.

Figure 12.1: Examples from all classes for both MNIST (Left) and SVHN (RIght)

Split/Dataset	**MNIST**	**SVHN**
Training	50000	50000
Validation	10000	23257
Test	10000	26032

Table 12.1: Dataset splits

12.3 Diabetic Retinopathy classification

The main experiment for this thesis is to classify DR images, and perform domain adaptation on the two datasets provided by UNN. There is an expected domain shift between the datasets due to different equipment used. The before and after domain adaptation performance will be shown in the result section, denotet as "Pre DA" and "Post DA" respectively.

The DA method chosen for the experiment is the ADDA method which was explained in detail in section 9.4. This method was chosen as it is fairly

common, robust, and well understood.

Due to the size of the dataset, and limitations on storage capacity the dataset has been pruned. Some datapoints do not have six images attached to them, and some have more. To create uniformity in the dataset, the eyes with less or equal to five images associated with it where not used for training or validation. For images consisting of more than six images, a random selection of six images where chosen to represent the eye. Due to the authors lack of medical expertise this was chosen as a simpler, quicker, and probably equal solution to manually attempt to choose the images. However, note that this is solely an implementation inspired choice, as the model generalizes to a variable number of images.

The main experiment for this thesis is DR image classification, and perform domain adaptation between the two datasets. For this task the T6 has been chosen as the source domain, and the T7 has been chosen as the target domain. This thesis mainly concern itself with an balanced and binary DR classification task to reduce the imbalance, decribed in Section 1.2.1, which is considered out of the scope of this thesis. The binary classification setting combines all classes from mild to proliferate into a class we will call DR. The second class is the No DR category.

Dataset / Class	No DR examples	DR examples
T6	1165	1165
T7	1057	1057

Table 12.2: Number of examples per class. The DR class is a combination of the *mild*, *moderate*, *severe*, and *proliferate* class.

Completing this final experiment successfully will be our final proof of the models capability to solve **challenge** #1 - **challenge** #4

Chapter 13

Results

13.1 Effects of noisy data on Multistream model

The following section presents the finding for the first MNIST and SVHN experiment. This experiments adapts a multi-stream model onto another multi-stream model with the ADDA method, where we provide evidence supporting our models capabilities of addressing **Challenge** #4. To even perform ADDA, we have to train the multi-stream model in a supervised setting which addresses **Challenge** #1. We also show the models ability to learn under artificially inserted noisy[1] data.

Dataset / Nr noisy images	0	1	2	3	4	5
SVHN	0.9998	0.9994	0.9899	0.5117	0.7885	0.1959
MNIST (Pre DA)	0.3263	0.3243	0.1284	0.1031	0.1235	0.1135
MNIST (Post DA)	0.9970	0.9814	0.9188	0.4964	0.5777	0.1134

Table 13.1: Accuracy on MNIST and SVHN data when introduced to same-class noisy data.

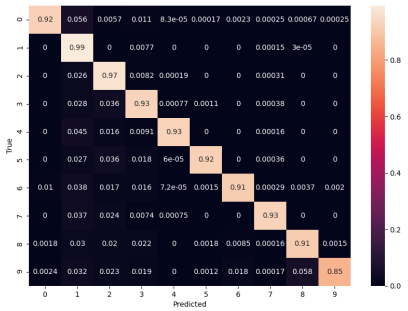
[1]Noisy in this case means data which either is, or appears to be from a different class. This can be a "1" written so badly it looks like a "7", or deliberately inputting wrong data, e.g labeling a "0" an "8"

Dataset / Nr noisy images	0	1	2	3	4	5
SVHN	0.9998	0.9838	0.9762	0.9134	0.4113	0.1821
MNIST (Pre DA)	0.3544	0.1356	0.1284	0.1356	0.1119	0.0911
MNIST (Post DA)	0.8089	0.7817	0.7689	0.7341	0.3071	0.1241

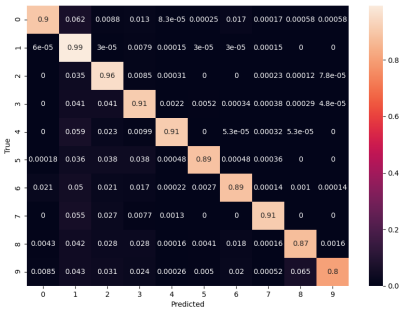
Table 13.2: Accuracy on MNIST and SVHN data when introduced to random-class noisy data.

Table 13.1 shows results for structured noisy input. With structured noise we mean noise which is sampled from the same wrong class. The accuracy scores are calculated on the MNIST and SVHN test set after training completed and the hyper parameters had been tuned by observing the validation set. In Figure 13.1a to 13.1f, we also provide confusion matrices for all permutations of the input.

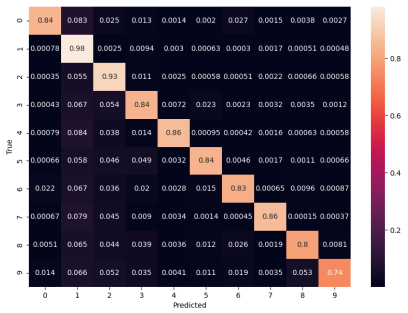
The fusion function used for this experiment is a simple mean of all the individual features. The more detailed results will be presented below. The results in Table 13.1 are the final test data results, while the confusion matrices and tSNE plots below shows data from the validation set, as the test set performed significantly better than the validation set. This trend was seen across the board. The test set performed better than the validation set overall.



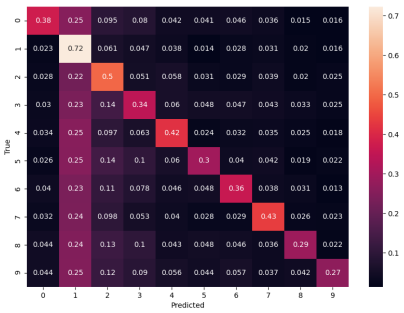
(a) Source confusion matrix for 0 noisy images



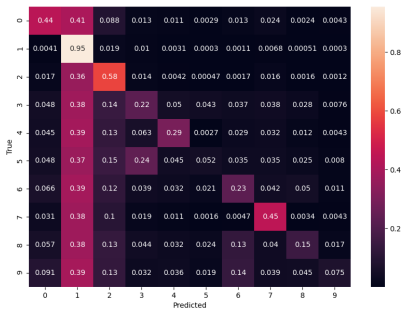
(b) Source confusion matrix for 1 noisy images



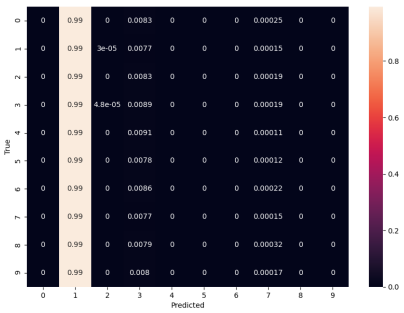
(c) Source confusion matrix for 2 noisy images



(d) Source confusion matrix for 3 noisy images



(e) Source confusion matrix for 4 noisy images



(f) Source confusion matrix for 5 noisy images

Figure 13.1: Confusion matrices for 0-5 noisy SVHN images (source data)

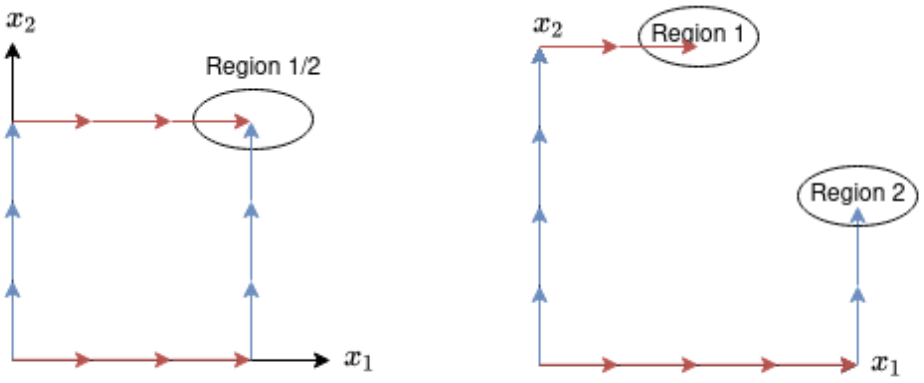
Lets consider the results of the source data to begin with.

No noise: The zero noise experiment showed excellent results. Both zero noise experiments, shown in Table 13.1 and 13.2, performs almost perfectly. There is a difference in the post DA MNIST accuracies however, but this is most likely due to a bad initialization. We can reason this as there is no difference in the experiments at this time, no noisy data. Just this experiment gives us empirical evidence that the model is capable of dealing with **Challenge** #4. The following paragraphs will analyse the models noise robustness.

One and two noisy images: In the one and two noise experiments we observe a small degradation. The input noise structure differs between the two experiments. Whereas Table 13.1 shows results where all noisy input images are from the same class, Table 13.2 shows results where the input noise is randomly sampled from wrong classes. The solid performance in light of input noise can be attributed to the fusion function for the random noise at least. However, we still expect the problem to get more difficult with the addition of more noise.

Three to five noisy images: The results for the three, four, and five noisy images are rather interesting, and somewhat revealing to how one should interpret the mean fusion function. Observe in Table 13.1 how the performance falls significantly for three, and five noisy images. Meanwhile the performance for four noisy images are reasonably good.

This good performance can be attributed to the noisy data always being from the same class. As a simplified illustrative example, consider a two-dimensional feature space where the extracted features from each class is always a unit vector, $\mathbf{f}_1 = [1, 0]$, $\mathbf{f}_2 = [0, 1]$. For a case of three clean, and three noisy input images the feature vectors will always map onto the same feature region. For a case where the number of clean and noisy images are not equal however, the combination of feature vectors will map onto different regions, assuming no feature vector is equal, which can then be separated by the classifier. The two cases discussed here is visualised in Figure 13.2

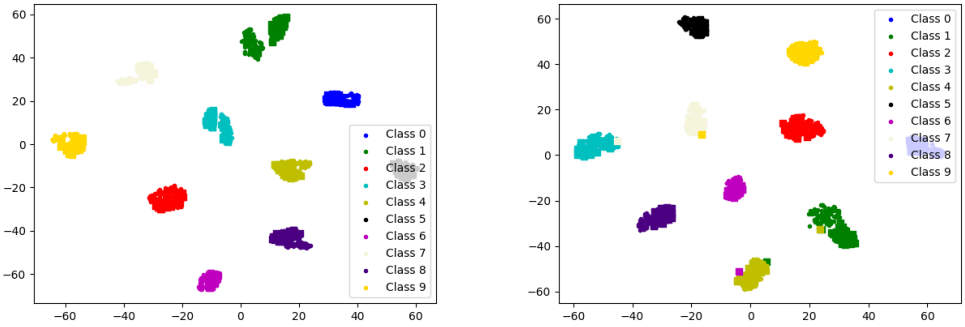


(a) How three clean and three noisy images might map onto the same region

(b) How two clean and four noisy images can map onto different regions

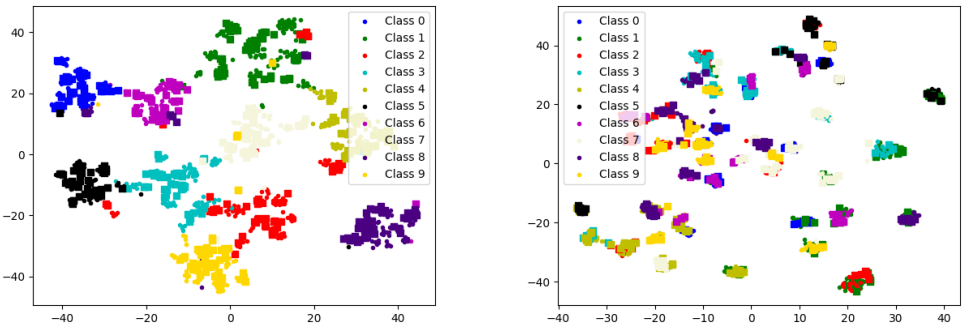
Figure 13.2

This hypothesis is strengthen when considering Table 13.2. When random class noise is introduced there is a steady decline in performance which is more in line with our intuition of how one expects the noise to affect the model. One can also observe the performance of three noisy / three clean images has improved for the random noise experiment, as the features are not mapped onto the same region. At least not frequently enough for the classifier to pick up on.



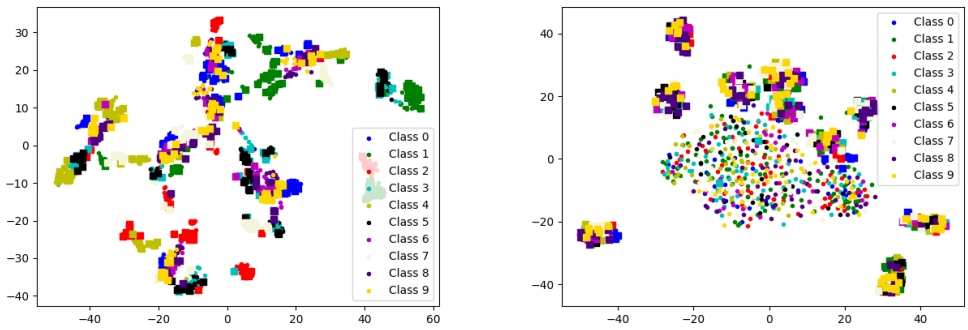
(a) No noisy image (b) One noisy image

Figure 13.4: tSNE plots of SVHN (Source), and MNIST (Target) together. Source is shown as a circle, and target as a square. These images are from the structured noise experiments



(a) Two noisy images (b) Three noisy images

Figure 13.5: tSNE plots of SVHN (Source), and MNIST (Target) together. Source is shown as a circle, and target as a square. These images are from the structured noise experiments



(a) Four noisy images

(b) Five noisy images

Figure 13.6: tSNE plots of SVHN (Source), and MNIST (Target) together. Source is shown as a circle, and target as a square. These images are from the structured noise experiments

The main result to take away from this experiment is that the multi-stream structure presented in section 11.1 is compatible with DA methods, specifically the ADDA method for this experiment. This provides further proof of the models ability to deal with **challenge** #4. With the exception of five noisy input images, Table 13.1 and 13.2 shows an improvement of performance after domain adaptation has occurred.

The difference in post DA MNIST for the structured noise, Table 13.1, and the random noise, Table 13.2, is most likely due to a bad initialization for the model which dealt with the random noise insertion. There is also the possibility off a slight miss-alignment during the DA process. If we go by the first hypothesis however, it stands to reason that if one network has had a bad initialization, then all the other must also have had it as the networks where trained in sequence on the same random seed.

Finally for this experiment, let us discuss Figure 13.6b. Here we see a clear case of the domains not aligning. However as we can clearly see in Table 13.1, the four noisy image case could achieve alignment. It is most likely the case that the five noisy image case could also achieve alignment, and a good accuracy. However if we take a look at the average losses over training in Figure 13.7

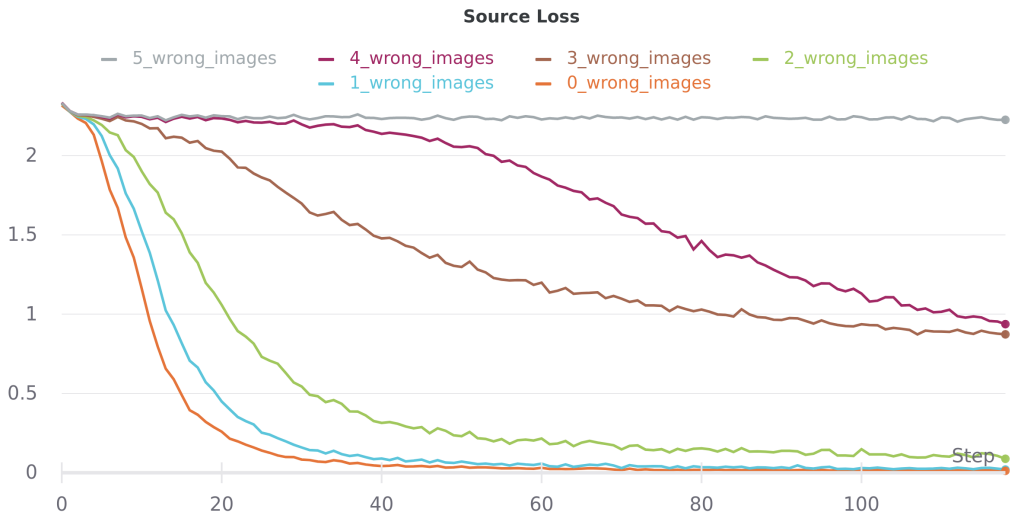


Figure 13.7: Loss for structural noisy image

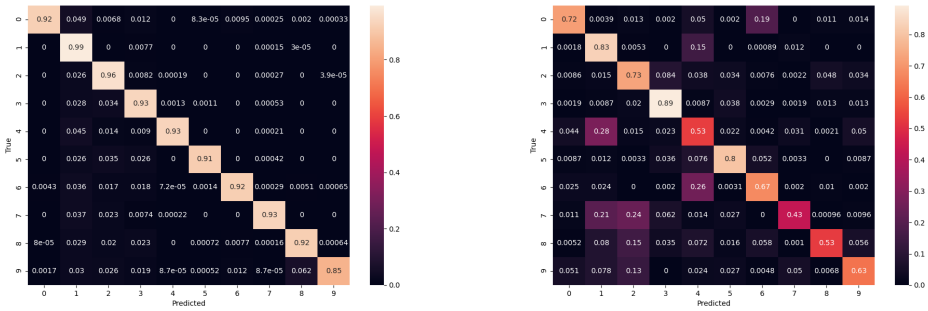
If we look closely, we can see a slight downwards trend for the five noisy image loss (the grey line). We can imagine that if trained for long enough, the network could learn all the feature regions for all the different clean plus noisy label combinations. It is off no interest to actually train a network which takes this ling to barely go down in loss value. But this is all to say that the five noise experiment failed to align not because of the DA method, but because the network had not yet finished training.

13.2 Multistream DA onto single network

This section presents the results for the second MNIST and SVHN experiment. Here we perform domain adaptation with a source multi-stream model, and a target single-stream model. The DA method used here is also ADDA. This section provides further support for our models capabilities of addressing **Challenge** #4. The results from this experiment in combination with the previous experiment provides evidence for our models capability of addressing **Challenge** #2.

Dataset	Accuracy
SVHN	1.000
MNIST (Pre DA)	0.3485
MNIST (Post DA)	0.7116

Table 13.3: Accuracy from multi-stream SVHN model and single-stream MNIST.



(a) Source confusion matrix

(b) Target confusion matrix

The fusion function used in these experiments is also a mean of all output feature vectors from the CNN networks.

Table 13.3 shows the main result of this section which is that a multi-stream model can be adapted onto a single-stream model. The increase in performance shows that a significant alignment between source and target has occurred. The adaptation is not perfect, and comparing to multi-stream to multi-stream DA showed in Table 13.1 there is a significant discrepancy of 0.2854. One explanation for this discrepancy might be due to the multi-stream network being more used to "solid" features. As the multi-stream takes the mean of image-level features, it is reasonable to assume the feature vector outputted from the fusion function will often be close to the mean of their class distribution, especially for these types of experiments. If the discriminator learns on these stronger features it might start to learn a more narrow class distribution which makes the classifier more volatile when it comes to the hard examples.

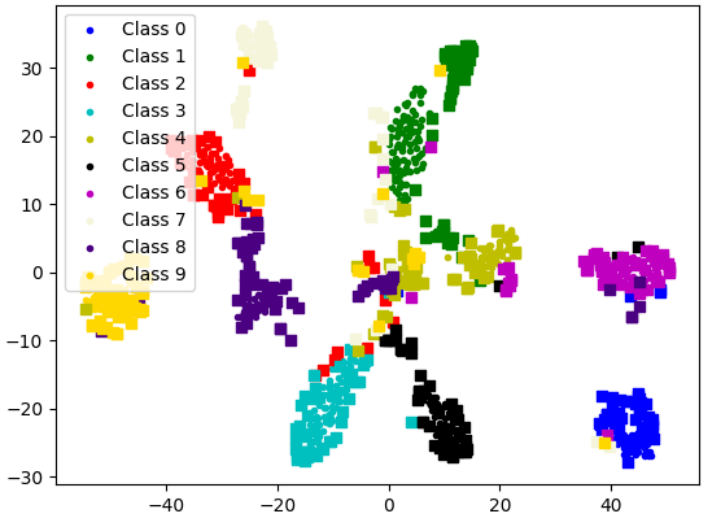
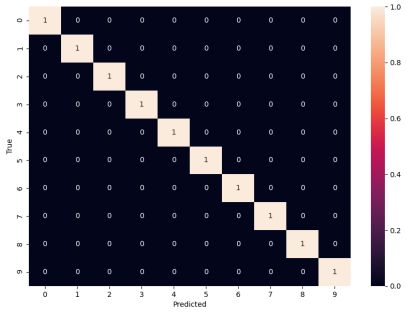
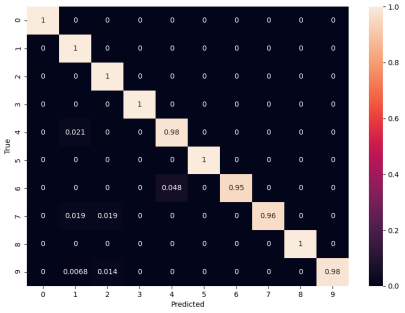


Figure 13.9: tSNE plot showing the alignment between a multi-stream source model, and a single-stream target model. Source (circles), target (squares)

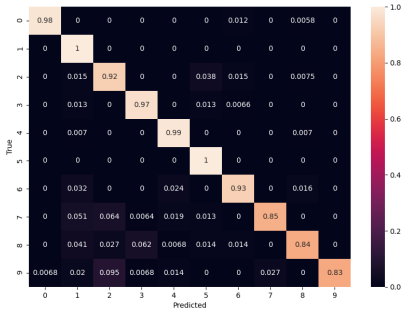
A high accuracy is not what is important here however. The important take away from these results are that if it's possible to domain adapt onto a single network, then it is possible to perform domain adaptation onto a multi-stream model of any size; from single-stream to n-sized multi-stream models. These results show it should be possible to train a model on the labeled knowledge of the Tromsø eye study data which can be transfered onto data collected from any clinic.



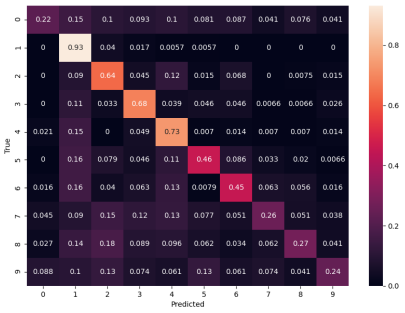
(a) Target confusion matrix for 0 noisy images



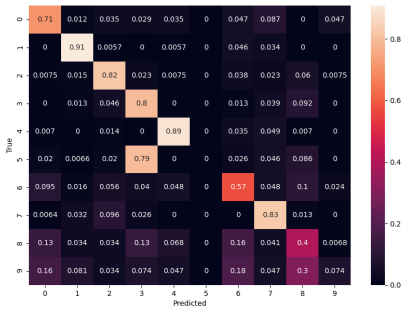
(b) Target confusion matrix for 1 noisy images



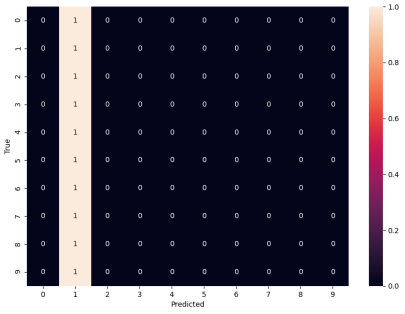
(c) Target confusion matrix for 2 noisy images



(d) Target confusion matrix for 3 noisy images



(e) Target confusion matrix for 4 noisy images



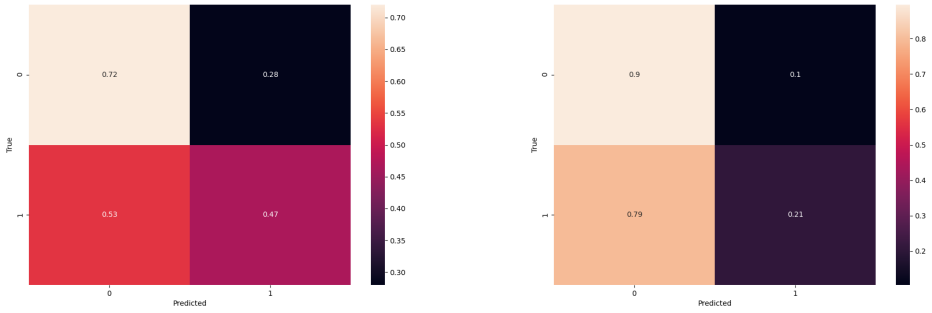
(f) Target confusion matrix for 5 noisy images

13.3 DR data

This section provides the results achieved when training, and testing on the UNN DR datasets. The parameters, and other settings are detailed in the experiment section. The best results from the DA experiments are listed below in Table 13.4. These results shows us our model addressing all challenges mentioned in Section 1.2.1

Dataset / Accuracy	Binary Class
T6	0.5949
T7 (Pre DA)	0.4786
T7 (Post DA)	0.5357
Discriminator	0.4964

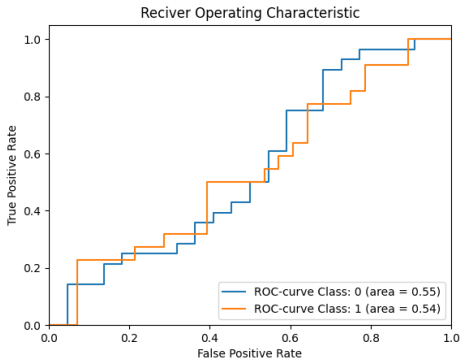
Table 13.4: Accuracies achieved on binary DR dataset



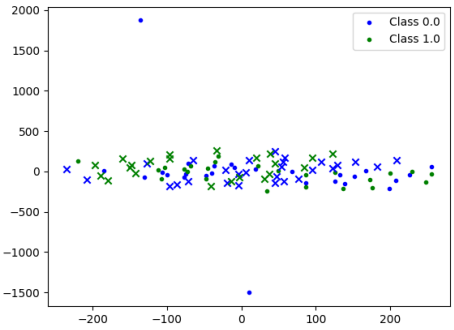
(a) Source domain confusion matrix (b) Target domain condusion matrix

Figure 13.10

While these results are not as well as hoped, we see a definitive improvement between the pre and post DA step. The supervised trained model has also learned something which is obvious from its +9.5% above random guessing accuracy. As stated earlier the considered dataset comes with severe challenges and these results need to be considered as a first step towards designing DR detection models.



(a) ROC-curve and AUC



(b) tSNE plot for DR. Source (Circles), and Target (Crosses) are displayed class-wise. Class 0 being "No DR", and class 1 being "DR"

Figure 13.11: Some additional figures for the DR classification experiment

From the tSNE plot shown in Figure 13.11b shows us what seems to be a a domain alignment. This makes sense, as the discriminators accuracy, seen in Table 13.4, is close to random guessing. It would seem that if the supervised setting had performed better, a proper alignment could have occured.

These results might not be as well as we hoped for, but it could have been expected. Back in the DR method section there was a figure showing the class imbalance, Figure 10.2. Note that the two prominent classes are the **No DR** class, and the **Mild** class. The difference between these classes is small spots barely the size of a few pixels. These classes are so similar it doesn't come as a surprise that they overlap as much. If this doesn't make sense, have a look at Figure 10.4a, and 10.4b and try to spot the difference.

The model has shown that it can deal with the challenges presented in the introduction when applied to the benchmark datasets. We also see that these challenges have been addressed in these results, yet on a small scale. With these results, in combination with the previous experiments and everything outlined in the methods Section 11, we can confidently state out model has addressed all challenges stated in Section 1.2.1.

The first challenge, being to make a model which can model dependence between images is the challenge with the most proof to back it up at this time. We have shown this addressed in all three experiments; Twice in the first SVHN and MNIST experiment, one in the second SVHN and MNIST experiment, and once in the final DR classification experiment. This challenge even stands out the most in the final DR classification results seen in Table 13.4. The solution to this challenge is the fusion function, which takes image-level features, and combines them into a feature which addresses the overarching structure.

The second challenge, being to make a model which can take in an arbitrary number of input images, has been addressed in all experiments too. It saw the most empirically support for it being addressed in the second SVHN and MNIST experiment. There we showed the possibility of varying input sizes. This is also the challenge with the most mathematical backing as shown in Section 11.2.2. We have given solid arguments for the model being general for any inputsize, only limited by memory in the practical setting.

The third challenge, being that we cannot expect data to be inputted in a specific order, has been addressed both theoretically in Section 11.2.2, and shown to work empirically in this experiment. This was addressed by manipulating dimensions in such a way our model simulates a multi-stream model containing multiple CNNs with shared weights, but just containing a single CNN in the practical setting. This forces the CNN to learn to detect features on a general basis, making input order irrelevant.

The forth and final challenge was more practically motivated. The constructed model should have the capabilities of performing domain adaptation. Due to the single shared weight CNN, we have argued the capabilities of applying ADDA in Section 11.3. But the same argument can be made for many other domain adaptation structures which relies on a single, or multiple CNNs. This network should be capable of performing most, if not all DA methods. The challenge has also been addressed in all experiments, with clear results of domain alignment achieved totally, or partially.

Part V

Discussion and future work

Chapter 14

Discussion and Future directions

14.1 Dealing with imbalance in DR datasets

This thesis has experimented with binary DR classification. Some side experimentation was done on unbalanced five class DR, but to no satisfactory results. This experimentation was mainly done with weighted cross entropy loss. Even this makes training on the unbalanced DR data hard, as some classes are just too rare to properly learn from. The author was however made aware of a loss known as focal loss [39], which might do the job in helping the model learn from the few severe or proliferate examples there are. The focal loss was discussed in the theory section (Section 4.1.2) and provided a good alternative for to the cross entropy loss when faced with imbalanced data.

14.2 Attention as a fusion function

The fusion functions that have been described in the method section have mainly been static fusion functions. This however can be detrimental, and lead to unfortunate or unforeseen network behaviours which was displayed in the structured noisy data results. Initially it was not expected for the network to be able to learn all permutations of the clean data and the noisy data, but it was an interesting find and gave insight to the fusion function.

The fusions presented in this thesis are simple, and intuitive in nature, but some advanced fusion function could improve performance. A dynamic fusion function which could change depending on the task would be a good fit for the model. For future work we recommend looking into attention networks which can learn weights to create a weighted sum fusion of the feature vectors instead [89]. This allows for the focus on important features, instead of the more static versions.

14.3 Explainability and Interpretability

In a practical setting any medical imaging system should be compatible with deep learning explainability and interpretability methods. While it was not part of this thesis, we argued in the method section for our multi-stream model effectively having a single view interpretation To this end, we see no reason for the network to not be compatible with methods such as GRADcam [69], guided backpropagation [79], etc, and encourage further investigation into this direction.

Chapter 15

Conclusion

This thesis set out to classify never before seen diabetic retinopathy data provided by UNN. This data was not collected with machine learning in mind, and to this end we constructed a multi-stream model for DR detection that addresses the challenges inherent to the data. In particular the multi-stream architecture allows the processing of multiple dependent input images. Our carefully designed fusion mechanism further allows us to fuse multiple feature vectors extracted from different parts of the retina, into a feature vector which represents the eye as a whole. Due to differing number of inputsizes, our model was designed to take in an arbitrary number of dependent images as input, only limited by memory. The datasets we worked with was chaotic, and the retina region of each eye was not provided. The model therefore had to be able to detect DR lesions in general, as any retina images from any part could enter any of the models streams. Finally, for practical reasons when it comes to real life deployment, it's important that out model can deal with the changing data distributions. To this end we made our model compatible with the ADDA domain adaptation method, and argued for its applicability also to other DA methods. Thereby addressing our main challenges.

Experimental evaluation was performed on benchmark datasets to explore the properties of the model as a whole, and its advantages. These evaluations highlight the models potential to address the presented challenges. Finally we experimented on two realistic datasets which where provided by UNN. The datasets were not collected for the purpose of machine learning, and therefore provided a unique opportunity to address real life problems. Our proposed solution lets us take a first step towards automatic DR detection based on retina images from the population of Tromsø.

Bibliography

[1] Diabetic eye screening: guidance on camera approval, Jan 2014.

[2] Deep image mining for diabetic retinopathy screening. *Medical Image Analysis*, 39:178–193, 2017.

[3] Ryo Asaoka, Hiroshi Murata, Kazunori Hirasawa, Yuri Fujino, Masato Matsuura, Atsuya Miki, Takashi Kanamoto, Yoko Ikeda, Kazuhiko Mori, Aiko Iwase, et al. Using deep learning and transfer learning to accurately diagnose early-onset glaucoma from macular optical coherence tomography images. *American journal of ophthalmology*, 198:136–145, 2019.

[4] Association of American Medical Colleges. The Complexities of Physician Supply and Demand: Projections From 2018 to 2033. Technical report, Washington, D.C., June 2020.

[5] Muhammad Awais, Md. Tauhid Bin Iqbal, and Sung-Ho Bae. Revisiting internal covariate shift for batch normalization. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11, 2020.

[6] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, 2009.

[7] Geir Bertelsen, Maja Erke, Therese Hanno, Ellisiv Mathiesen, Tünde Petö, Anne Sjølie, and Inger Njølstad. The tromsø eye study: Study design, methodology and results on visual acuity and refractive errors. *Acta ophthalmologica*, 91, 09 2012.

[8] Aleksandar Botev, Guy Lever, and David Barber. Nesterov's accelerated gradient and momentum as approximations to regularised update descent, 2016.

[9] Gabriela Csurka, Fabien Baradel, Boris Chidlovskii, and Stephane Clinchant. Discrepancy-based networks for unsupervised domain adaptation: A comparative study. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[10] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[11] Zhijie Deng, Yucen Luo, and Jun Zhu. Cluster alignment with a teacher for unsupervised domain adaptation. *CoRR*, abs/1903.09980, 2019.

[12] Mariachiara Di Cesare, James Bentham, Gretchen Stevens, Bin Zhou, Goodarz Danaei, Yuan Lu, Honor Bixby, Melanie Cowan, Leanne Riley, Kaveh Hajifathalian, Lea Fortunato, Cristina Taddei, James Bennett, Nayu Ikeda, Young-Ho Khang, Catherine Kyobutungi, Avula Laxmaiah, Yanping Li, Hsien-Ho Lin, and Julio Cisneros. Trends in adult body-mass index in 200 countries from 1975 to 2014: A pooled analysis of 1698 population-based measurement studies with 19·2 million participants. *The Lancet*, 387:1377–1396, 04 2016.

[13] Bin Ding, Huimin Qian, and Jun Zhou. Activation functions and their characteristics in deep neural networks. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 1836–1841, 2018.

[14] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[15] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

[16] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

[17] Anne Elise Eggen, Ellisiv B. Mathiesen, Tom Wilsgaard, Bjarne K. Jacobsen, and Inger Njølstad. The sixth survey of the tromsø study (tromsø 6) in 2007–08: Collaborative research in the interface between

clinical medicine and epidemiology: Study objectives, design, data collection procedures, and attendance in a multipurpose population-based health survey. *Scandinavian Journal of Public Health*, 41(1):65–80, 2013. PMID: 23341355.

[18] Alpaydın. Ethem. *Introduction to machine learning*. MIT Press, 3 edition, 2014.

[19] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks, 2016.

[20] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[21] Varun Gulshan, Lily Peng, Marc Coram, Martin C. Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, Ramasamy Kim, Rajiv Raman, Philip C. Nelson, Jessica L. Mega, and Dale R. Webster. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA*, 316(22):2402–2410, 12 2016.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[23] Samitha Herath, Mehrtash Harandi, and Fatih Porikli. Learning an invariant hilbert space for domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[24] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A. Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. *CoRR*, abs/1711.03213, 2017.

[25] Sara Hooker. The hardware lottery, 2020.

[26] Zilong Hu, Jinshan Tang, Ziming Wang, Kai Zhang, Ling Zhang, and Qingling Sun. Deep learning for image-based cancer detection and diagnosisa survey. *Pattern Recognition*, 83:134–149, 2018.

[27] Goodfellow. Ian, Bengibl Yoshua, and Courville Aaron. *Deep learning*. MIT Press, 2016.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[29] Ashish K. Jha, Catherine M. DesRoches, Peter D. Kralovec, and Maulik S. Joshi. A progress report on electronic health records in u.s. hospitals. *Health Affairs*, 29(10):1951–1957, 2010. PMID: 20798168.

[30] Nour Eldeen M Khalifa, Mohamed Loey, Mohamed Hamed N Taha, and Hamed Nasr Eldin T Mohamed. Deep transfer learning models for medical diabetic retinopathy detection. *Acta Informatica Medica*, 27(5):327, 2019.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[32] Wouter M. Kouw. An introduction to domain adaptation and transfer learning. *CoRR*, abs/1812.11806, 2018.

[33] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[34] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: A large video database for human motion recognition. In *2011 International Conference on Computer Vision*, pages 2556–2563, 2011.

[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[36] Cindy S. Lee, Paul G. Nagy, Sallie J. Weaver, and David E. Newman-Toker. Cognitive and system factors contributing to diagnostic errors in radiology. *American Journal of Roentgenology*, 201(3):611–617, Sep 2013.

[37] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *CoRR*, abs/1712.09913, 2017.

[38] Xiaogang Li, Tiantian Pang, Biao Xiong, Weixiang Liu, Ping Liang, and Tianfu Wang. Convolutional neural networks based transfer learning for diabetic retinopathy fundus image classification. In *2017 10th international congress on image and signal processing, biomedical engineering and informatics (CISP-BMEI)*, pages 1–11. IEEE, 2017.

[39] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[40] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A. W. M. van der Laak, Bram van Ginneken, and Clara I. Sanchez. A survey on deep learning in medical image analysis. *CoRR*, abs/1702.05747, 2017.

[41] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, and Clara I. Sanchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60 – 88, 2017.

[42] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88, 2017.

[43] Fang Liu, Zhaoye Zhou, Alexey Samsonov, Donna Blankenbaker, Will Larison, Andrew Kanarek, Kevin Lian, Shivkumar Kambhampati, and Richard Kijowski. Deep learning approach for evaluating knee mr images: Achieving high diagnostic performance for cartilage lesion detection. *Radiology*, 289(1):160–169, 2018. PMID: 30063195.

[44] Jenny X Liu, Yevgeniy Goryakin, Akiko Maeda, Tim Bruckner, and Richard Scheffler. Global health workforce labor market projections for 2030. *Policy Reaserch Working Paper*, 7790, 2016.

[45] Xiaoxuan Liu, Livia Faes, Aditya Kale, Siegfried Wagner, Dun Fu, Alice Bruynseels, Thushika Mahendiran, Gabriella Moraes, Mohith Shamdas, Christoph Kern, Joseph Ledsam, MD Schmid, Konstantinos Balaskas, Eric Topol, Lucas Bachmann, Pearse Keane, and Alastair Denniston. A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis. *The Lancet Digital Health*, 1, 09 2019.

[46] Jenny X Liua, Yevgeniy Goryakin, Akiko Maeda, Tim Allen Bruckner, and Richard M. Scheffler. Global health workforce labor market projections for 2030. Policy Research Working Paper Series 7790, The World Bank, August 2016.

[47] Shih-Chung Lo, S.-L.A. Lou, Jyh-Shyan Lin, Matthew Freedman, Minze Chien, and Seong Mun. Artificial convolution neural network techniques and applications for lung nodule detection. *Medical Imaging, IEEE Transactions on*, 14:711 – 718, 12 1995.

[48] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I. Jordan. Domain adaptation with randomized multilinear adversarial networks. *CoRR*, abs/1705.10667, 2017.

[49] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[50] Xinhong Ma, Tianzhu Zhang, and Changsheng Xu. Gcan: Graph convolutional adversarial network for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[51] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[52] Seymour A. Papert Marvin Minsky. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, Cambridge, expanded edition, 1987.

[53] Sarfaraz Masood, Tarun Luthra, Himanshu Sundriyal, and Mumtaz Ahmed. Identification of diabetic retinopathy in eye images using

transfer learning. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 1183–1187, 2017.

[54] Sarfaraz Masood, Tarun Luthra, Himanshu Sundriyal, and Mumtaz Ahmed. Identification of diabetic retinopathy in eye images using transfer learning. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 1183–1187. IEEE, 2017.

[55] Tanya Nair, Doina Precup, Douglas L. Arnold, and Tal Arbel. Exploring uncertainty measures in deep networks for multiple sclerosis lesion detection and segmentation. *Medical Image Analysis*, 59:101557, 2020.

[56] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. volume 27, pages 807–814, 06 2010.

[57] Stanislav Nikolov, Sam Blackwell, Ruheena Mendes, Jeffrey De Fauw, Clemens Meyer, Cían Hughes, Harry Askham, Bernardino Romera-Paredes, Alan Karthikesalingam, Carlton Chu, Dawn Carnell, Cheng Boon, Derek D'Souza, Syed Ali Moinuddin, Kevin Sullivan, DeepMind Radiographer Consortium, Hugh Montgomery, Geraint Rees, Ricky Sharma, Mustafa Suleyman, Trevor Back, Joseph R. Ledsam, and Olaf Ronneberger. Deep learning to achieve clinically applicable segmentation of head and neck anatomy for radiotherapy. *CoRR*, abs/1809.04430, 2018.

[58] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018.

[59] World Health Organization. *World report on vision*. World Health Organization, 2019.

[60] Craig A. Pedersen, Philip J. Schneider, and Douglas J. Scheckelhoff. ASHP national survey of pharmacy practice in hospital settings: Prescribing and transcribing—2016. *American Journal of Health-System Pharmacy*, 74(17):1336–1352, 09 2017.

[61] Francesco Piccialli, Vittorio Di Somma, Fabio Giampaolo, Salvatore Cuomo, and Giancarlo Fortino. A survey on deep learning in medicine: Why, how and when? *Information Fusion*, 66:111–137, 2021.

[62] Sehrish Qummar, Fiaz Gul Khan, Sajid Shah, Ahmad Khan, Shahaboddin Shamshirband, Zia Ur Rehman, Iftikhar Ahmed Khan, and Waqas Jadoon. A deep learning ensemble approach for diabetic retinopathy detection. *IEEE Access*, 7:150530–150539, 2019.

[63] Hariharan Ravishankar, Prasad Sudhakar, Rahul Venkataramani, Sheshadri Thiruvenkadam, Pavan Annangi, Narayanan Babu, and Vivek Vaidya. Understanding the mechanisms of deep transfer learning for medical images. In *Deep learning and data labeling for medical applications*, pages 188–196. Springer, 2016.

[64] Ievgen Redko, Emilie Morvant, Amaury Habrard, Marc Sebban, and Younès Bennani. A survey on domain adaptation theory. *CoRR*, abs/2004.11829, 2020.

[65] Jonathan Richens, Ciaran Lee, and Saurabh Johri. Improving the accuracy of medical diagnosis with causal machine learning. *Nature Communications*, 11:3923, 08 2020.

[66] Kuniaki Saito, Donghyun Kim, Stan Sclaroff, Trevor Darrell, and Kate Saenko. Semi-supervised domain adaptation via minimax entropy. *CoRR*, abs/1904.06487, 2019.

[67] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?, 2019.

[68] Rory Sayres, Ankur Taly, Ehsan Rahimy, Katy Blumer, David Coz, Naama Hammel, Jonathan Krause, Arunachalam Narayanaswamy, Zahra Rastegar, Derek Wu, Shawn Xu, Scott Barb, Anthony Joseph, Michael Shumski, Jesse Smith, Arjun B. Sood, Greg S. Corrado, Lily Peng, and Dale R. Webster. Using a deep learning algorithm and integrated gradients explanation to assist grading for diabetic retinopathy. *Ophthalmology*, 126(4):552 – 564, 2019.

[69] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why

did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[70] Theodoridis. Sergios and Koutroumbas. Konstantinos. *Pattern Recognition*. Elsevier, 4 edition, 2009.

[71] K Shankar, Yizhuo Zhang, Yiwei Liu, Ling Wu, and Chi-Hua Chen. Hyperparameter tuning deep learning for diabetic retinopathy fundus image classification. *IEEE Access*, 8:118164–118173, 2020.

[72] Yaxin Shen, Bin Sheng, Ruogu Fang, Huating Li, Ling Dai, Skylar Stolte, Jing Qin, Weiping Jia, and Dinggang Shen. Domain-invariant interpretable fundus image quality assessment. *Medical Image Analysis*, 61:101654, 2020.

[73] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298, 2016.

[74] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[75] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos, 2014.

[76] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[77] Ruoxian Song, Peng Cao, Jinzhu Yang, Dazhe Zhao, and Osmar R Zaiane. A domain adaptation multi-instance learning for diabetic retinopathy grading on retinal images. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 743–750. IEEE, 2020.

[78] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012.

[79] Jost Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. 12 2014.

[80] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.

[81] James V. Stone. Information theory: A tutorial introduction. *CoRR*, abs/1802.05968, 2018.

[82] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[83] Ben Tan, Yu Zhang, Sinno Pan, and Qiang Yang. Distant domain transfer learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017.

[84] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[85] Teknologiraadet. Kunstig intelligens - muligheter, utfordringer og en plan for norge. 2018.

[86] Zhigang Tu, Wei Xie, Qianqing Qin, Ronald Poppe, Remco C. Veltkamp, Baoxin Li, and Junsong Yuan. Multi-stream cnn: Learning representations based on human-related regions for action recognition. *Pattern Recognition*, 79:32–43, 2018.

[87] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. *CoRR*, abs/1702.05464, 2017.

[88] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[89] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[90] IV Walton, O. Bennett, Robert B. Garoon, Christina Y. Weng, Jacob Gross, Alex K. Young, Kathryn A. Camero, Haoxing Jin, Petros E. Carvounis, Robert E. Coffee, and Yvonne I. Chu. Evaluation of Automated Teleretinal Screening Program for Diabetic Retinopathy. *JAMA Ophthalmology*, 134(2):204–209, 02 2016.

[91] Jing Wang, Yiwei Chen, Wanyue Li, Wen Kong, Yi He, Chuihui Jiang, and Guohua Shi. Domain adaptation model for retinopathy detection from cross-domain oct images. In Tal Arbel, Ismail Ben Ayed, Marleen de Bruijne, Maxime Descoteaux, Herve Lombaert, and Christopher Pal, editors, *Proceedings of the Third Conference on Medical Imaging with Deep Learning*, volume 121 of *Proceedings of Machine Learning Research*, pages 795–810. PMLR, 06–08 Jul 2020.

[92] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *CoRR*, abs/1802.03601, 2018.

[93] World Health Organisation (WHO). Global Report on Diabetes. Working Papers id:10553, eSocialSciences, April 2016.

[94] Garrett Wilson and Diane J. Cook. A survey of unsupervised deep domain adaptation. *ACM Trans. Intell. Syst. Technol.*, 11(5), July 2020.

[95] Zuxuan Wu, Yu-Gang Jiang, Xi Wang, Hao Ye, Xiangyang Xue, and Jun Wang. Fusing multi-stream deep networks for video classification. *CoRR*, abs/1509.06086, 2015.

[96] Xifeng Guo, Wei Chen, and Jianping Yin. A simple approach for unsupervised domain adaptation. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 1566–1570, 2016.

[97] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.

[98] Dalu Yang, Yehui Yang, Tiantian Huang, Binghong Wu, Lei Wang, and Yanwu Xu. Residual-cyclegan based camera adaptation for robust diabetic retinopathy screening. In Anne L. Martel, Purang Abolmaesumi, Danail Stoyanov, Diana Mateus, Maria A. Zuluaga, S. Kevin Zhou, Daniel Racoceanu, and Leo Joskowicz, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2020*, 2020.

[99] Yanchao Yang and Stefano Soatto. FDA: fourier domain adaptation for semantic segmentation. *CoRR*, abs/2004.05498, 2020.

[100] Yinsheng Zhang, Li Wang, Zhenquan Wu, Jian Zeng, Yi Chen, Ruyin Tian, Jinfeng Zhao, and Guoming Zhang. Development of an automated screening system for retinopathy of prematurity using a deep neural network for wide-angle retinal images. *IEEE Access*, 7:10232–10241, 2019.

[101] Mohammadreza Zolfaghari, Gabriel L. Oliveira, Nima Sedaghat, and Thomas Brox. Chained multi-stream networks exploiting pose, motion, and appearance for action classification and detection. *CoRR*, abs/1704.00616, 2017.