Convolutional Neural Networks for Multi-Label Image Classification

by

Jan André Marais

Thesis presented in partial fulfilment of the requirements for the degree of Master of Commerce (Mathematical Statistics) in the Faculty of Economic and Management Sciences at Stellenbosch University

Supervisor: Dr. S. Bierman

December 2017

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .....

Abstract

Convolutional Neural Networks for Multi-Label Image Classification

J. A. Marais

Thesis: MCom (Mathematical Statistics)

December 2017

Deforestation has devastaing effects on the environment, contributing to climate change and habitat loss amongst others. Better data about the location and cause of deforestation can help authorities improve their response time and understanding of the problem. Therefore, a classifier that can label satellite images of a rainforest with various classes of land cover and land use, would be very helpful. We pose this problem as a mutli-label image classification (MLIC) problem, where each satellite image can potentially be annotated with more than one label. Convolutional Neural Networks (CNNs) have consistently proven their superiority in single-label image classification problems. However, it is unclear how to best extend CNNs to effectively deal with the unique challenges that arise in the multi-label setting. This work critically reviews the proposals made in the literature and compares them on a dataset of > 100000 labelled satellite images of the Amazon rainforest. We found that (1) training the network with the label-wise binary cross-entropy loss function is sufficient in dealing with label imbalance; (2) making predictions from multi-scale feature maps helps to detect smaller objects; and (3) explicity modelling channel-wise dependencies of feature maps helps the network to exploit semantic and spatial label relationships. A bonus to the chosen network design is that it provides approximate label localisations despite only being trained with label supervision.

Uittreksel

Konvolusie Neurale Netwerke vir Multi-Etikel Beeldklassifikasie

("Convolutional Neural Networks for Multi-Label Image Classification")

J. A. Marais

Tesis: MCom (Wiskundige Statistiek)

Desember 2017

Afrikaans abstract

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations ...

Contents

List of Figures

List of Tables

Nomenclature

N	number of observations in a dataset
p	input dimension or the number of features for an observation
K	number of labels in a dataset
$\boldsymbol{x}$	p-dimensional input vector $(x_1, x_2, \ldots, x_p)^\intercal$
λ	label
$\mathcal{L}$	complete set of labels in a dataset $\mathcal{L} = \{\lambda_1, \lambda_2, \ldots, \lambda_K\}$
Y	labelset associated with $\boldsymbol{x}$, $Y \subseteq \mathcal{L}$
$\hat{Y}$	predicted labelset associated with $\boldsymbol{x}$, $\hat{Y} \subseteq \mathcal{L}$, produced by $h(\cdot)$
$\boldsymbol{y}$	K-dimensional label indicator vector, $(y_1, y_2, \ldots, y_K)^\intercal$, associated with observation $\boldsymbol{x}$
$(\boldsymbol{x}_i, Y_i)_{i=1}^N$	multi-label dataset with N observations
D	dataset
$h(\cdot)$	multi-label classifier $h : \mathbb{R}^p \to 2^{\mathcal{L}}$, where $h(\boldsymbol{x})$ returns the set of labels for $\boldsymbol{x}$
θ	set of parameters for $h(\cdot)$
$\hat{\theta}$	set of parameters for $h(\cdot)$ that optimise the loss function
$L(\cdot, \cdot)$	loss function between predicted and true labels
$f(\cdot)$	label prediction module, $f : \mathbb{R}^p \to \mathbb{R}^K$
$t(\cdot)$	thresholding function, $t : \mathbb{R}^K \to \{0, 1\}^K$
$\mathcal{N}(\boldsymbol{x})$	points in the input space neighbourhood of $\boldsymbol{x}$

Chapter 1

Introduction

1.1 The Task

The focus of this thesis is to produce a system that can identify multiple thoracic pathologies[1] from a chest X-ray image of a patient. Suppose this system is denoted by the function $h(\cdot)$, then it can be illustrated by the diagram in Figure 1.1.

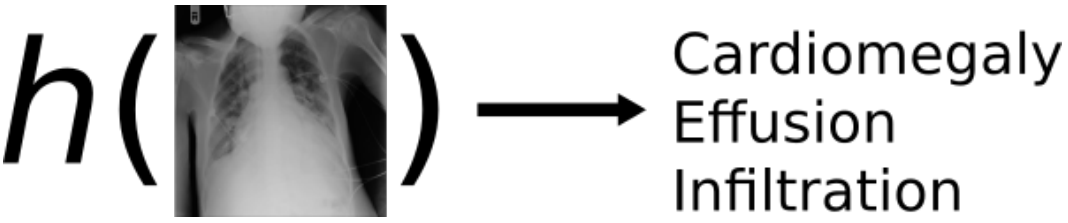

Figure 1.1: An illustration of the system $h(\cdot)$ that can read a chest X-ray and produce a list of relevant thoracic pathologies.

We formulate this task as a multi-label image classification task (Wang *et al.*, 2017), since the inputs are images and each image can be associated with more than one pathology, *i.e.* label. We denote an X-ray image by the vector

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{bmatrix},$$

[1]Pathologies that are located in the thorax region - between the neck and abdomen.

for $i = 1, 2, \ldots, N$, where N is the number of X-rays in the dataset and p the number of pixels in each X-ray image. We also refer to $\boldsymbol{x}$ as an observation. Suppose the set of all possible labels is defined as $\mathcal{L} = \{\lambda_1, \lambda_2, \ldots, \lambda_K\}$ then each observation is associated with a subset of the full labelset, $Y_i \subseteq \mathcal{L}$. Therefore we require a system that is a mapping from the input space to the labelspace (Zhang and Zhou, 2014), *i.e.*:

$$h : \mathbb{R}^p \to 2^{\mathcal{L}}.$$

We also refer to $h(\cdot)$ as the classifier or the model.

How do we model such an $h(\cdot)$ that produces accurate mappings? Following the *supervised learning* framework (see Hastie *et al.*, 2009, Ch. 2) for classification, $h(\cdot)$ is learned by example, *i.e.* from the data. The dataset consist of N input-output pairs (also known as examples) and is denoted by

$$(\boldsymbol{x}_i, Y_i)_{i=1}^N.$$

To learn $h(\cdot)$ from the data, we first need to assume its functional form (or its shape). Suppose the mapping performed by $h(\cdot)$ is defined by a set of parameters θ such that the configuration of θ determines how $h(\cdot)$ processes $\boldsymbol{x}$ to produce Y, however limited to the form we initially assumed for $h(\cdot)$. Suppose a specific paramater configuration $\hat{\theta}$ produces an output $\hat{Y}_i$ for $\boldsymbol{x}_i$, *i.e.*

$$h_{\hat{\theta}}(\boldsymbol{x}_i) \to \hat{Y}_i,$$

then we can measure how much the prediction made by $h_{\hat{\theta}}(\cdot)$ deviates from the ground truth by some *loss function*, $L(Y_i, \hat{Y}_i)$. The loss function should produce large values if the prediction is far from the truth and values close to zero if the prediction matches the true label(s). Therefore our task can now roughly[2] be reduced to the optimisation problem of finding the optimal parameters $\hat{\theta}$ such that the loss between the predictions and the ground truth is a minimum:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{N} L\left(Y_i, h_\theta(\boldsymbol{x}_i)\right)$$

This minimum can be found using any appropriate optimisation procedure.

The solution to our task can now be summarised by the following four main components:

[2]This is a slight simplification of the real task, used to convey a basic understanding. The actual optimisation problem includes a penalty term for regularisation, which we discuss later.

- The **data**, $(\boldsymbol{x}_i, Y_i)_{i=1}^N$,
- The **classifier**, $h_\theta(\cdot)$,
- The **loss function**, L, and
- The **optimisation procedure**.

In this work the data is the recently released ChestX-ray14 dataset (Wang *et al.*, 2017) which contains 112,120 frontal-view chest X-ray images individually labeled with up to 14 different thoracic diseases. An in-depth look at the data is given in Section 2.1.1.

For the classifier we use an artifical neural network (ANN) that specialises in detecting patterns from images, called a *convolutional neural network* (CNN). Exploring the different CNN architectures applicable to multi-label image classification will make up the main body of work for this thesis, as well as how these perform on our specific task. CNNs are described in Chapter 4 and how they can be adapted to better fit the multi-label image classification problem in Chapter 5. The results of the experiments done on our data can be found in Chapter 6. We attempt to cover most of the approaches proposed in the literature, in addition to some novel methods.

Not only is the evaluation of a multi-label classifier an ambiguous task, finding an appropriate surrogate loss function that can efficiently be minimised by the optimisation procdure is a challenging part of the problem. A handful of such loss functions have been proposed in the literature, which we also explore in Chapter 5 and experiment with in Chapter 6.

The most common optimisation procedure used along with neural networks is the *stochastic gradient descent* (SGD) algortihm. We provide a brief overview of this method in Section 3.5.4 and discuss some of the bells and whistles that can be added to improve the algorithm in Chapter 4. A detailed overview and experimentation of the best optimisation procedures for this problem is beyond the scope of this thesis. We will use the most robust, consistent and simplest method proved by the literature for our task.

remove outline details to outline section

An effective exploration and implementation concerning the above components should assist us in achieving the objectives described in the following section.

1.2 Objectives

The main target for this work is to achieve competive results with the state-of-the-art (SotA) approach on the ChestX-ray14 dataset reported in (Rajpurkar *et al.*, 2017). However, along the way we also aim to to accomplish the following sub-objectives:

- Develop a thorough understanding of CNNs and how they are applied to an image classification problem.
- Collect and review all the latest research on CNNs applied to multi-label image classification and develop a thorough understanding of how these methods compare to each other in addition to how they should perform in certain scenarios.
- Learn how the existing approaches can be tweaked or adapted to optimally fit to our data.
- Provide rigorous empirical evaluations of the selected existing approaches as well as the novel adaptions proposed in this thesis on the ChestX-ray14 dataset, such that the conclusions made here can safely be used in future research on this dataset and possibly other multi-label image classification problems.
- Design reproducible experiments for any researcher or person with an interest in the field to replicate the results and further build on the approaches.

Before we delve into the details of our work, we first want to motivate why each of these objectives are important to accomplish.

1.3 Motivation

The chest X-ray is one of the most commonly accessible radiological examinations for screening and diagnosis of many thoracic pathologies (Wang *et al.*, 2017). For diseases like pneumonia, it is currently the best available method for diagnosis (Organization *et al.*, 2001). However, the diagnosis of thoracic diseases from X-rays require the availability of expert radiologists. An effective *computer aided diagnosis* (CAD) system can help increase the accessibility to expert knowledge by breaking down some of the usual barriers facing patients, the major one being the financial cost of an examination. A CAD system

can assist radiologist to make more accurate diagnosis but at a quicker rate and therefore increasing the availability of such expert knowledge and thus driving down the cost of an examination. Maybe someday in the near future, such systems will become fully autonomous providing everyone with free, 100% trustworthy diagnosis. This is why we found the application in this domain to be important and worthy of study.

Multi-label image classification is not restricted to the application to the diagnosis of lung diseases from chest X-rays. In fact, it can be appplied to many important real-world problems. Teaching a computer to "see", *i.e.* parse complex real-world imagery, will assist in the automation of many tasks. An example of another interesting and important problem that fits into the multi-label image classification framework is in the analysis of satellite imagery. Recently, a Kaggle competition[3] was hosted where the task was to design a system that can identify multiple land cover and land use classes from satellite images of the Amazon Rainforest. The effective development of such a system would assist authorities to track the changes in the forest on a large scale and ultimately help with the fight against deforestation.

Although the original proposal for CNNs dates back to some time ago (based on a series of work done by Hubel and Wiesel, 1962, Fukushima (1980) & LeCun *et al.* (1999)) the first publication of a sucessful large-scale implementation was fairly recently (Krizhevsky *et al.*, 2012). Ever since, the SotA in most *computer vision* (CV) tasks are dominated by CNNs. Its success is attributed to its remarkable ability to learn useful representations from raw inputs (Lecun *et al.*, 2015) such as images in its pixel value form. However, up to now, the main focus of CNN research was on problems concerning the simpler scenario of multiclass image classification, where each image is only associated with a single label. See the annual large scale image classifcation competition known as ImageNet (Russakovsky *et al.*, 2015) as an example of such a problem. Figure 1.2 plots the misclassification error rate of the winning model for each year. Note the large drop in the error rate from 2011 to 2012 where a CNN was used for the first time in this competition.

Far less attention is given to the application of CNNs on a multi-label image classification problem. Which is arguebly a more relevant task since most of real-world images can be associated with more than one class. Although

[3]https://www.kaggle.com/c/planet-understanding-the-amazon-from-space

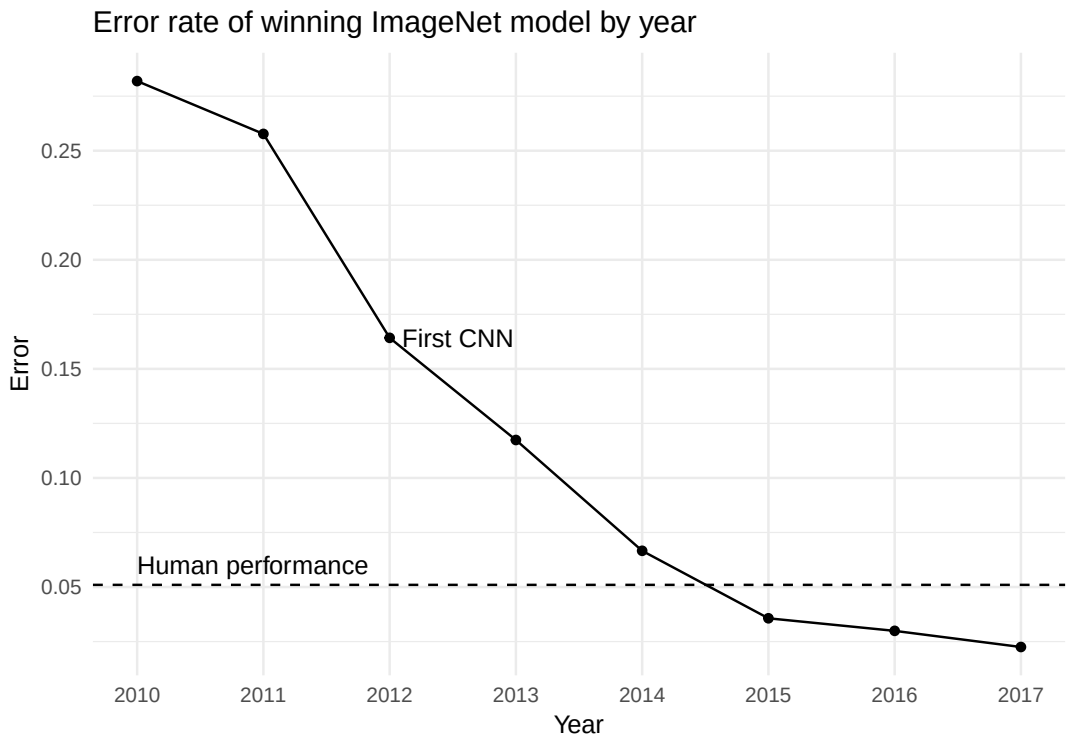

Figure 1.2: The misclassification error rate for the winning ImageNet model per year.

a fairly trivial adaption can be made to a single-label[4] CNN to apply it to a multi-label case (see Section 5.1), it does not explicitly deal with some of the unique challenges of modeling images with multiple labels like exploiting label correlation and detecting objects of different scales. Only a handful of publications address some of these issues (see for example Gong *et al.*, 2013, Wei *et al.* (2014), Wang *et al.* (2016), Zhu *et al.* (2017), Chen *et al.* (2017)). This is still a developing research direction and many of the publications are made in isolation, which warrants further study and comparisons of approaches in this area.

Since theory and practice does not always go hand-in-hand, it is usually advantageous to complement a theoretical study with empirical results. Another motivation for empirical study is that we regard the ability to implement an approach equally as important as understanding the theory behind it. We characterise a good empirical experiment as one that is *rigorous* and *reproducible*. Recently the field of *Deep Learning*, which is the class of statistical learning models multi-layer ANNs, including CNNs, belong to, has been criticised for

[4]Referring to binary or multiclass classification tasks.

the growing gap between the understanding of its techniques and its practical successes[5] where most of the recent focus was on the latter. The speakers urged the deep learning community to be more rigorous in their experiments where, for them, the most important part of rigor is better empiricism, not more mathematical theories. Better empiricism in classification may include for example practices such as using cross-validation to estimate the generalisation ability of a model and reporting standard errors. Empirical studies should be more than just trying to beat the benchmark and should also consist of simple experiments that aid in the understanding of how the techniques work.

On the other hand, we want all our experiments to be as reproducible as possible, *i.e.* provide all the code, data and necessary documentation to reproduce the experiments that were done in this thesis[6]. This is often an overlooked feature of experiments, but is however crucial for transparent and accountable reporting and making your work useful for others to build upon.

The task and framework for the research problem in this thesis has been discussed in addition to the further objectives of this study along with why they are important to achieve. We are now equiped to delve deeper into the details of convolutional neural networks for multi-label image classification and how it can be applied the problem of diagnosing thoracic diseases from chest X-rays. The structure for the remainder of the thesis is as follows.

1.4 Outline

In Chapter 2 we do a exploratory analysis of the data and highlight some of the challenges for learning from our specific dataset. We also have a look at ...

> give outline for rest of thesis once done. use the information already given at the end of task section.

[5] How do I cite the talk given at NIPS2017 - https://www.youtube.com/watch?v=Qi1Yry33TQE

[6] All of these are shared publicly at https://github.com/jandremarais/Thesis

Chapter 2

Data and Evaluation

In Section 1.1 we identified the data as one of the four main components of the multi-label image classification problem. Before any statistical learning algorithm is applied a thorough exploratory analysis of the data develops a better understanding of the problem. It is necessary to become familiar with the data and its properties since these should be taken into account in the modelling stage. In this chapter we explore the basic properties of the data, the important multi-label indicators and the nature of the input-output pairs. From this analysis we identify the possible challenges for applying multi-label CNNs to this dataset.

The performance evaluation of a multi-label learning system is much more complicated than in the single-label scenario. Section 2.2 zooms in on the importance of various multi-label metrics, how they are defined and which are important to our task. We also look at the considerations for splitting multi-label data into training and testing sets to ensure robust estimates.

2.1 The data

2.1.1 Description

This ChestX-ray14 dataset (Wang *et al.*, 2017) consists of 112,120 X-ray images with disease labels from 30,805 unique patients ($N = 112120$). To put it into perspective, the second largest publicly available chest X-ray dataset[1] to date has only 4,143 frontal view data points. The large number of X-rays in ChestX-ray14 better facilitates the data hungry process of deep neural network

[1] https://openi.nlm.nih.gov

training. All X-rays were resized to 1024×1024, meaning the total number of raw features (the pixels) for each X-ray is $p = 1048576$, since X-rays only have one color channel compared to color RGB images with 3 channels (see subsection 2.1.2). This is quite a large size for a typical image classification problem, for example the images in ImageNet are only of size 224×224.

The labels for each X-ray were obtained by using *Natural Languape Processing* techniques to text-mine disease classifications from the corresponding radiological reports. The authors expect the labels to be >90% accurate. The original publication of ChestX-ray14 had a total of eight possible disease labels but the latest version is labeled with up to 14 possible diseases ($K = 14$), they include: *Atelectasis*, *Consolidation*, *Infiltration*, *Pneumothorax*, *Edema*, *Emphysema*, *Fibrosis*, *Effusion*, *Pneumonia*, *Pleural_thickening*, *Cardiomegaly*, *Nodule*, *Mass* and *Hernia*. Note, that some X-rays are associated with no labels. This is not a large labelset compared to some of the other benchmark multi-label datasets (see *extreme multi-label learning* (Zhang *et al.*, 2017, Prabhu and Varma (2014)) where K sometimes reach into the millions).

Examples of the input-output pairs are given in Figure 2.1. An untrained eye will find it extremely difficult to detect the patterns that defines the relationship between the X-rays and their corresponding diseases. The beauty (and sometimes the danger) of CNNs for image classification, as we will see in Chapter 4, is that in ideal conditions, no prior knowledge of the application domain is necessary for the algorithm to learn the relationships between the inputs and the outputs. Current ideal conditions for deep learning has mostly to do with having enough examples. When the number of training examples are limited, an inspection of the data may provide clues to how a model can be adapted to get the most out of the data.

2.1.2 Images as Data

To understand how an algorithm can process an image, we need to know how an image is stored and how it is represented as a set of numbers, interpretable by a machine. An image is made up of a grid of many tiny squares (or cells) with different colors. These cells are known as pixels and one pixel represents one color. A grayscale image, 32 pixels wide and 32 pixels long, can be represented by a 32×32 matrix of integers, where each integer represents the 'brightness' (or intensity) of each pixel. These integers usually lie within $[0, 255]$, such that

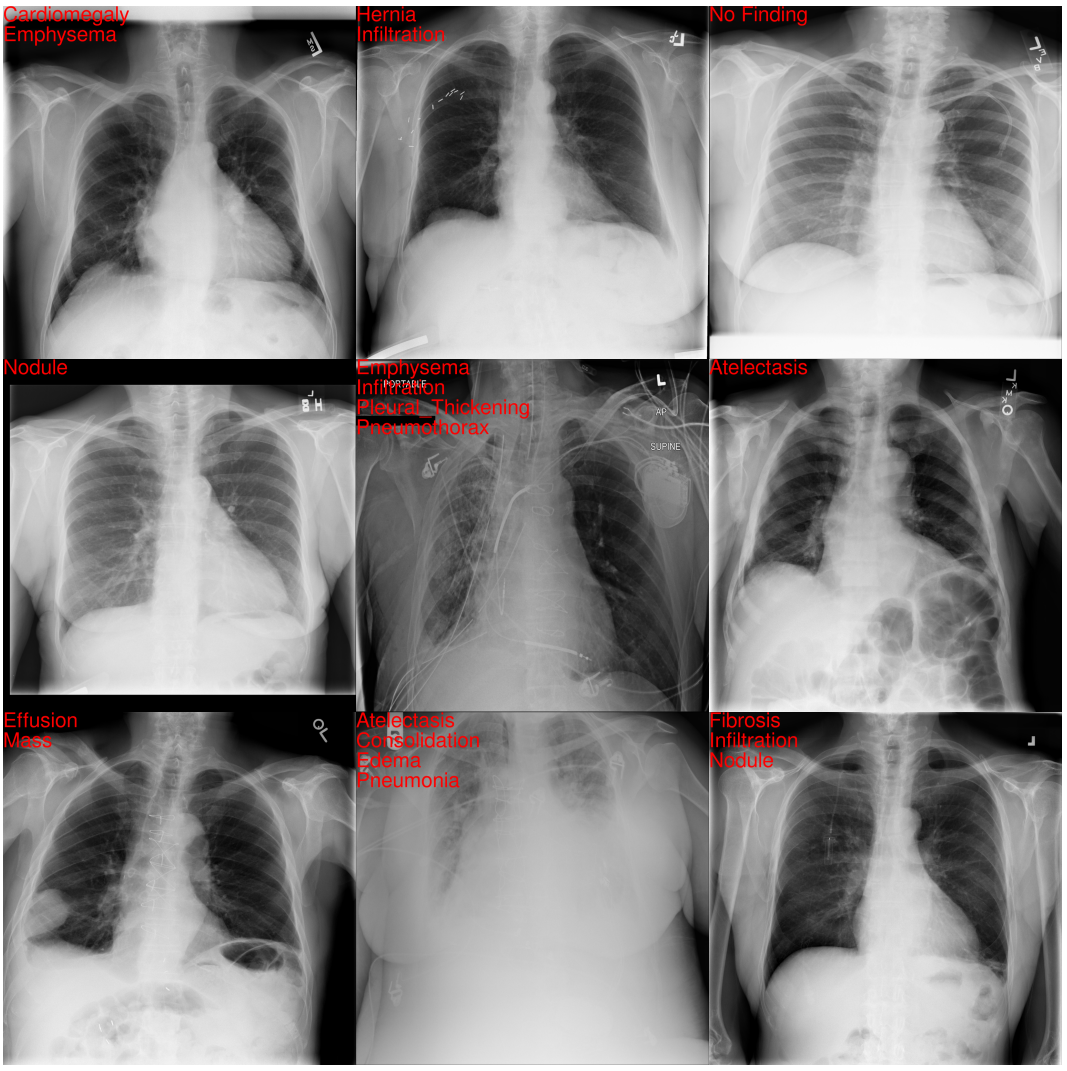

Figure 2.1: Examples of the X-rays and their disease labels contained in ChestX-ray14.

the greater the integer, the brighter the pixel. For example, a pixel with an intensity of 0 is totally black, and a pixel with an intensity of 255 is totally white. Figure 2.2 illustrates this representation by showing the pixel values from a zoomed in patch of a chest X-ray.

Note that a standard color image consists of three spectral bands (or channels), *viz.* red, green and blue (RGB). This means that the color of one pixel is determined by three integers in $[0, 255]$, each representing the intensity of the colors red, green and blue. Thus a 32×32 image is represented by a 3D-array of size $32 \times 32 \times 3$. X-rays are traditionally grayscale images and therefore only have one channel.

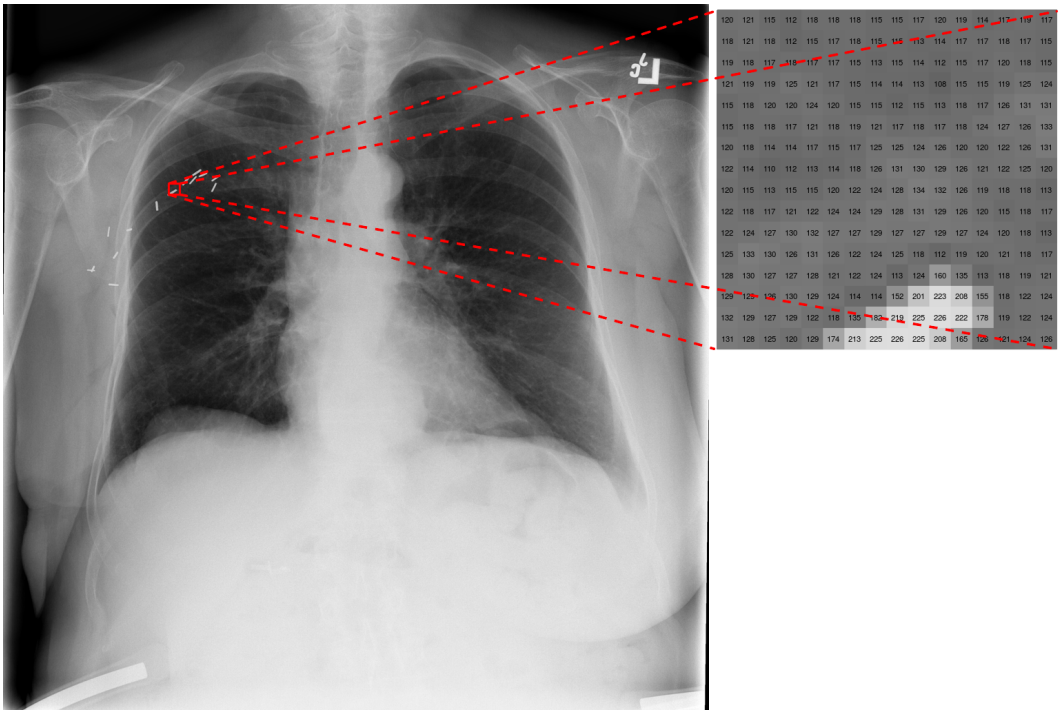

Figure 2.2: Illustration of a patch of an image as pixel values.

In summary, for each X-ray we have $1024 \times 1024 = 1048576$ values between 0 and 255 representing the brightness of a specific pixel in the image. This matrix can now be given to a learning algorithm. We refer to this representation of an image as its raw representation. Conventional classifiers struggle to make sense of images in its raw form and usually require some visual feature extraction process before classification (Sánchez *et al.*, 2013). We will see in Chapter 4 why this is not necessary for CNNs and how they effectively model images in their raw form.

Some of the reasons why it may be hard to distinguish the signal from the noise in the X-ray images in their raw form is as follows (adapted from (Karpathy)):

- **Scale variation**. The disease labels exhibit variation in their size. For example, the *Nodule* class is usually a very small part of the X-ray, whereas *Pleural_thickening* typically take up a much larger proportion.
- **Deformation**. One disease label can appear as many different shapes accross different X-rays. This is the case for the *Mass* label (Yao *et al.*, 2017).

- **Occlusion**. Some disease labels may only be partially visible, hidden behind other objects in the X-ray image.
- **Brightness and Contrast**. The X-rays were not all taken in the exact same lighting conditions and therefore vary in their brightness and contrast (see also Figure 2.5).
- **Background clutter**. The largest part of an X-ray in our dataset is background and totally irrelevant to the classification of disease labels. The background contains clutter such as markings and medical apparatus.

All of these factors may contribute to a relatively high intra-class variance and low inter-class variance. Diseases such as *Atelectasis*, *Consolidation*, *Pneumonia* and *Infiltration* are extremely difficult to distinguish between from X-ray images (Oakden-Rayner, 2017). Therefore it helps a classifier if the images in its raw form can be mapped to a space with more discriminative features between classes. Next, we explore more of the data in terms of its labels.

2.1.3 Label Distribution

The complete labelset for our problem is given by

$$\mathcal{L} = \{\text{Atelectasis}, \text{Consolidation}, \ldots, \text{Hernia}\},$$

where each X-ray is associated with a subset of $\mathcal{L}$ we denoted as Y_i for $i = 1, \ldots, N$. It is often also useful to denote the output of each observation in vector form, especially when defining statistical learning algorithms. Let the vector representation of the output for the i-th observation be

$$\boldsymbol{y}_i = \begin{bmatrix} y_{i1} \\ y_{i2} \\ \cdots \\ y_{iK} \end{bmatrix},$$

where $y_{ik} = 1$ if $\lambda_k \in Y_i$ (the k-th label is associated with the i-the observation) and $y_{ik} = 0$ otherwise, for $k = 1, 2, \ldots, K$. We index the labels as follows: λ_1: *Atelectasis*, λ_2: *Cardiomegaly*, λ_3: *Consolidation*, λ_4: *Edema*, λ_5: *Effusion*, λ_6: *Emphysema*, λ_7: *Fibrosis*, λ_8: *Hernia*, λ_9: *Infiltration*, λ_{10}: *Mass*, λ_{11}: *Nodule*, λ_{12}: *Pleural_thickening*, λ_{13}: *Pneumonia* and λ_{14}: *Pneumothorax*. See Table 2.1 for some label representation examples.

Table 2.1: Examples of label representations.

i	Y_i	y_{i1}	y_{i2}	y_{i3}	y_{i4}	y_{i5}	y_{i6}	y_{i7}	y_{i8}	y_{i9}	y_{i10}	y_{i11}	y_{i12}	y_{i13}	y_{i14}
1	{Cardiomegaly Emphysema}	0	1	0	0	0	1	0	0	0	0	0	0	0	0
2	{No Finding}	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	{Atelectasis, Consolidation, Edema, Pneumonia}	1	0	1	1	0	0	0	0	0	0	0	0	1	0

There are multiple properties of the label distribution that can be investigated. First, we can look at how the label count varies over each label in Figure 2.3 (a). The dataset has a fairly imbalanced label distribution, this is quite common in multi-label data. Only three of the labels (*Infiltration*, *Effusion*, *Atelectasis*) are present in more than 10% of the X-rays. Then there are labels such as *Pneumonia* and *Hernia* which occur very rarely, in 1.2% and 0.2% of the data, respectively.

Another insightful plot is the distribution of the number of labels per observation as in Figure 2.3 (b). We see that a large proportion (~54%) of the observations are associated with zero labels and roughly 27% of the observations with exactly one label. Thus only ~19% of the X-rays have multiple labels. The maximum number of labels observed is 9, occuring only twice in the dataset. This shows that the output for each observations is *sparse*, another type of imbalance where only a small proportion of the labels per observation is positive. We refer to all labels for which $y_{ik} = 0$ as negative labels.

Metrics summarising properties of the label distribution have been proposed in the literature. The two standard measures for the "multi-labeled-ness" of a data set are *label cardinality* and *label density*, introduced by (Tsoumakas and Katakis, 2007). The label cardinality of a multi-label dataset, D, set is the average number of labels per observation:

$$\text{LCard}(D) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}.$$

This measure can be normalised to be independent of the label set size, which results in the label density indicator:

$$\text{LDens}(D) = \frac{1}{K}\cdot\text{LCard}(D) = \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}.$$

According to (Tsoumakas and Katakis, 2007) it is important to distinguish between these two measures, since two data sets with the same label cardinality but with a great difference in the number of labels might not exhibit the same

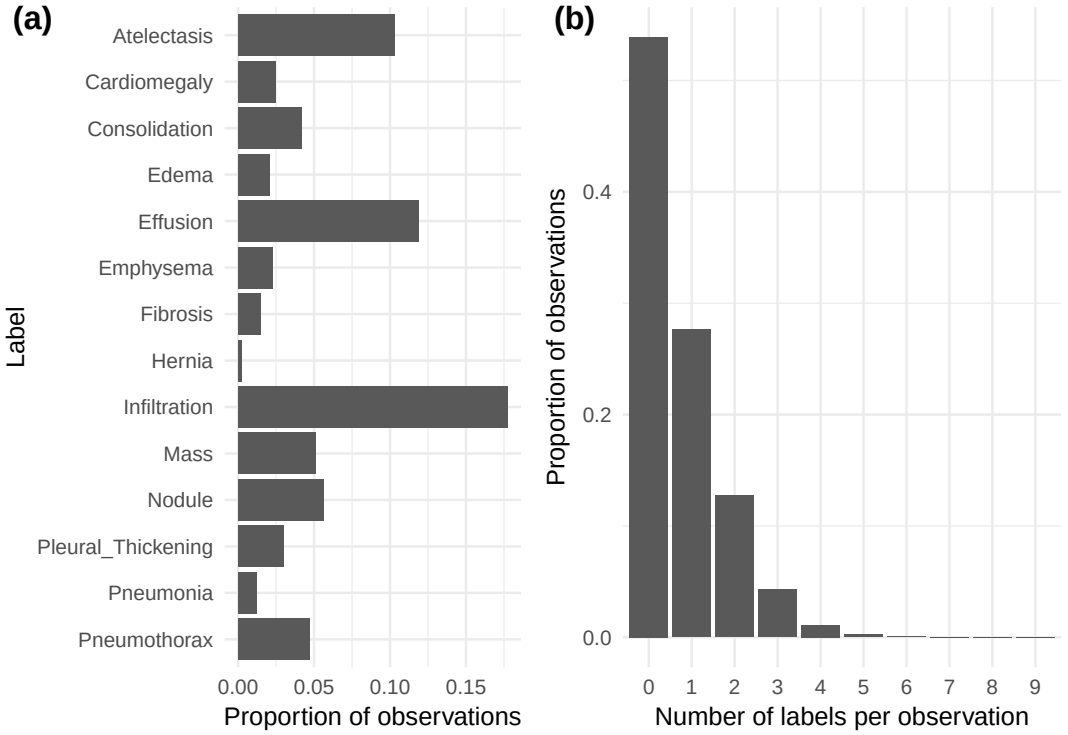

Figure 2.3: Bar plots exploring the label distribution in ChestX-ray14. (a) The proportion of observations associated with each label; (b) The distribution of the label count per observation.

properties and cause different behaviour to the *multi-label classification* (MLC) methods. These two measures give a good indication of the label frequency of a data set, but we are also interested in the uniformity and regularity of the labeling scheme. The authors of (Read *et al.*, 2011) suggested measuring the proportion of unique label combinations in the data and the proportion of observations in the data associated with the most frequently occuring label combination. Consider the number of unqiue label combinations, also referred to as the label diversity (Zhang and Zhou, 2014), which can be defined as:

$$\mathrm{LDiv}(D) = |\{Y | \exists \boldsymbol{x} : (\boldsymbol{x}, Y) \in D\}|.$$

The proportion of unique label combinations in D is then

$$\mathrm{PLDiv}(D) = \frac{1}{N} \cdot \mathrm{LDiv}(D).$$

The proportion of observations associated with the most frequently occuring label combination is defined by (Read *et al.*, 2011) as:

Table 2.2: Properties of popular multi-label image classification datasets.

Dataset	N	K	LDens	PLDiv	PMax
ChestX-ray14	**112120**	**14**	**0.0516**	**0.0070**	**0.5388**
Planet*	40479	17	0.1689	0.0102	0.3369
WIDER-Attribute	57517	14	0.2067	0.0206	0.0383
MSCOCO	123287	80	0.0363	0.2068	0.0219
NUS-WIDE	269648	81	0.0108	0.0440	0.5051
VOC2007	9963	20	0.0780	0.0309	0.0866
VOC2012	22263	20	0.0649	0.0181	0.4990

* Training set only

$$\text{PMax}(D) = \max_{Y} \frac{\text{count}(Y, D)}{N},$$

where $\text{count}(Y, D)$ is the frequency that label combination Y is found in data set D. Large values of PLDiv is an indication that there is a large variety of different label combinations accross the observations which would mean that there are fewer examples per unique label combination (note, the reverse is not necessarily true). Large values of PMax indicates a skewed labeling scheme, where most of the observations are associated with the most common label combination, which means that the other label combinations have fewer examples in the dataset.

We compare these multi-label indicators of the ChestX-ray14 dataset to those of the most popular multi-label image classification datasets. The results are summarised in Table 2.2. The datasets compared to are: *PASCAL VOC 2007 & 2012* (Everingham *et al.*, 2010, & Everingham *et al.* (2012)), *NUS-WIDE* (Chua *et al.*, 2009), *WIDER-Attribute* (Li *et al.*, 2016, preprocessed as in Zhu *et al.* (2017)), *MSCOCO* (Lin *et al.*, 2014) and *Planet*[2]. Some of these were actually designed for *object detection* learning, however they are also commonly used for multi-label image classification, just without the bounding box outputs.

ChestX-ray14 has a decent amount of data points. We note that although ChestX-ray14 has the smallest labelset, it still has fairly sparse output (small LDens), which we also see is not uncommon for images in multi-label datasets. The standout properties for ChestX-ray14 is its limited number of possible label

[2]https://www.kaggle.com/c/planet-understanding-the-amazon-from-space

combinations, indicated by PLDiv, and its large proportion of observations belonging to the most common label combination, which is the empty set in this case.

Very little research has been done on how all these multi-label indicators affect the performance of a multi-label classifier. (Chekina *et al.*, 2011) made a worthy attempt. Their goal was to find a way of determining which multi-label algorithm to use given a data set with specific properties and with a specific evaluation metric to optimise. They approached this problem by training a so called meta-learner on a meta-data set containing the performance of multiple multi-label algorithms on benchmark datasets with different properties. This trained meta-learner is then able to predict which multi-label algorithm is the most likely to give the best results in terms of a specific evaluation metric, given the properties of the data set to be analysed. We will not use their meta-learner for this thesis. However, it is useful to note that their meta-learner found a strong signal between the properties of the dataset and the best performing algorithm for that task. This strengthens the argument that it is important to take multi-label indicators into account before the training process. All of the properties reported in Table 2.2 proved to be important to the meta-learner.

One other important factor was the average χ^2-scores between pairs of labels. The χ^2-score between two labels can be viewed as a measure of dependence or correlation between them, since $\phi^2 = \frac{\chi^2}{N}$, where ϕ is the Pearson correlation between two binary variables. Label correlation is a very important consideration in MLC. We plot the pairwise sample Pearson correlation between labels in ChestX-ray14 in Figure 2.4 (a). Not many strong correlations are observed with the highest being between *Emphysema* and *Pneumothorax* (~0.178). This has partly to do with the low label cardinality of the dataset, also observed in (**?**). Therefore these correlations make more sense relatively than it does in an absolute sense.

There are two main limitations of using the Pearson correlation as a measure of label dependence: the sample correlation is a 'global' estimate and it assumes the correlation is symmetric. If for example a pair of labels are uncorrelated according to the sample correlation, that does not imply that they are independent for a given $\boldsymbol{x}$, since the sample correlation is the average correlation over all the data points. A distinction can be made between a *conditional dependence* and *unconditional dependence* (Dembszynski *et al.*, 2010). Two labels, λ_u, λ_v, are said to be unconditionaly dependent if

$$P(Y_u, Y_v) \neq P(Y_u) \times P(Y_v).$$

This dependence can be estimated from the data using any type of statistical correlation measure such as Pearson correlation. Despite being beneficial on average, modeling unconditional label dependence may not be optimal for a single point $\boldsymbol{x}$ since labels can also be conditionaly dependent, such that

$$P(Y_u, Y_v|\boldsymbol{X} = \boldsymbol{x}) \neq P(Y_u|\boldsymbol{x}) \times P(Y_v|\boldsymbol{x}).$$

This type of dependence is much harder to detect and model since it requires the conditioning on a typically large input space. Conditional dependence does not imply unconditional independence, nor the other way around. This should be taken into account before any label structure is imposed on the model.

The other drawback of the Pearson correlation as an estimate of the label dependence is that is assumes the dependence is symmetric. This is rarely the case in real life (Huang *et al.*, 2012) and it is more likely that

$$P(Y_u|Y_v) \neq P(Y_v|Y_u).$$

We can inspect these types of label relationships with a co-occurrence plot as in Figure 2.4 (b). Each cell in the plot is calculated by the formula:

$$\frac{\sum_{i=1}^{N} y_{iu} y_{iv}}{\sum_{i}^{N} y_{iu}}$$

where u indicates a label on the y-index and v a label on the x-axis, *i.e.* the proportion of times label λ_v is present, given λ_u is also present. This can be interpreted as an estimate for $P(Y_v|Y_u)$. Now the asymmetry of the label relations are more apparrent. We can see for example that *Effusion* is more likely to occur given *Cardiomegaly*, than *Cardiomegaly* given *Effusion*. Note however that these too are unconditional dependence estimates.

It would be reckless to further intepret these correlations without any medical expertise. The main purpose of this label correlation discussion was to communicate the different types of label dependence and how they may be estimated on average from the data. In a later discussion we will consider ways for a model to exploit these correlations to improve its prediction accuracy.

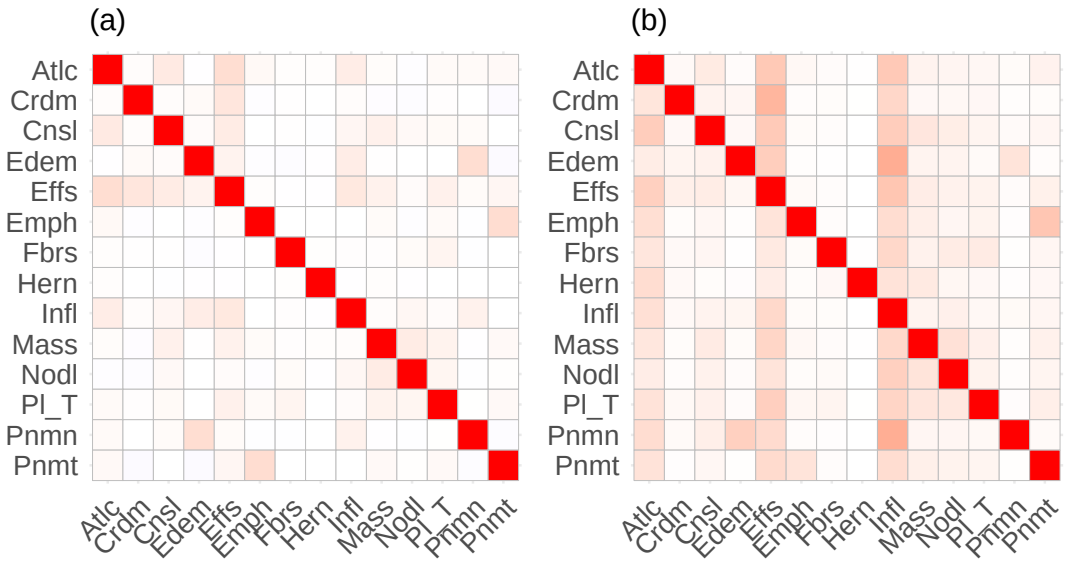

Figure 2.4: Visualisations of label dependence estimates. (a) The Pearson correlation matrix. (b) The co-occurrence matrix.

2.1.4 Challenges

From the analysis of the data we highlight the following factor which might make it hard for a CNN to learn discriminative visual features for MLC:

- **Number of observations**. Although this is by far the biggest dataset of its kind, it awaits to be seen if its sample size is sufficient for learning the complex disease patterns in the X-ray images. Combatting overfitting is a central theme in CNNs and statistical learning in general. Some approaches are discussed in througout the remainder of the thesis.

- **Image size**. The size of the X-ray images are large relative to other image classification datasets. This may require large amounts of computing power to process. We will need to find an image size the is small enough to process efficiently, without throwing away too much of the information in the image.

- **Image complexity**. Like most multi-label images, the images in ChestX-ray14 is complex. Labels such as *Mass* have a large amount of withing class variation. Some labels are extremely small, like *Mass* and *Nodule*. Then there are also images that are of poor quality, see Figure 2.5, and on top of that, some diseases are indistinguishable with the naked eye, for example *Atelectasis*, *Consolidation*, *Pneumonia* and *Infiltration*.

- **Class imbalance and label sparsity**. Classes such as *Pneumonia* and *Hernia* have very little examples in the dataset which makes it harder for a classifier to identify its patterns. Additionaly, there are few positive labels per observation and it might be difficult for a classifier to learn from such sparse outputs.
- **Dealing with label Correlation**. It is difficult to determine the exact label relationships from the data. We need to design a classifier that is able to take information from other labels into account when it is necessay, but at the same time ignore them when it is not.

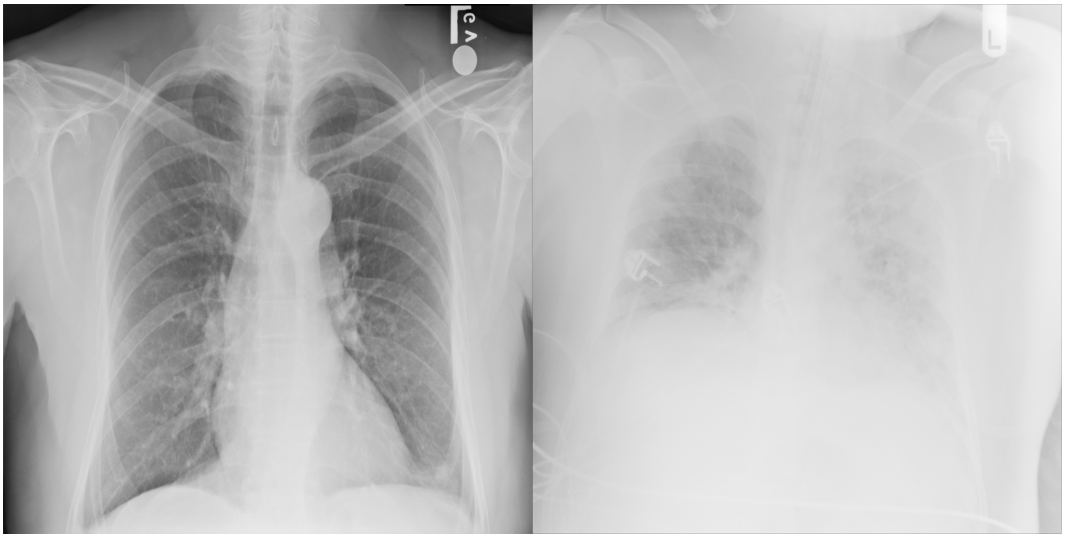

Figure 2.5: Comparing a good (left) vs poor (right) quality X-ray.

To explore the performance of different approaches and how they may overcome these challenges, we first need a measure of such performance for multi-label classifiers. The next section discuss how multi-label classifiers are evaluated.

2.2 Evaluating Multi-Label Models

2.2.1 Classification vs Ranking

Recall that the multi-label classifier is of the following form:

$$h_\theta(\boldsymbol{x}_i) \to Y_i$$

and that the set of labels for observation $\boldsymbol{x}_i$, Y_i, can also be denoted as an indicator vector $\boldsymbol{y}_i$. Although a multi-label classifier is trained against the binary ground truth output, it is often not possible for it to directly output a vector of 0's and 1's and returns a real value for each label instead. In the exact same way that, for example, the *k-nearest neighbour* algorithm for classification returns the proportions of the classes associated with the observations in a neighbourhood of the input space and the *linear model* returns a weighted average (real) of the input features. Most of the time the algorithm is designed in such a way that the real outputs lie in $[0, 1]$, so that it looks like posterior class probabilities, $P(\boldsymbol{y}|\boldsymbol{x})$. Thus we will need some thresholding function to map the real values for each label to binary output. This is similar to binary classification where it is common to map all outputs larger than a threshold of say 0.5 to 1 and the rest to 0, and multiclass classification where the largest output for a given observation is mapped to 1 and the rest to 0.

Thus a multi-label classifier can also be given by:

$$h(\boldsymbol{x}) = t\left(\left(f(\boldsymbol{x})\right)\right),$$

where $f(\boldsymbol{x})$ is the *label prediction module*, $f : \mathbb{R}^p \to \mathbb{R}^K$, and $t(\cdot)$ the *thresholding function* or label decision module that takes the real-values for each label and outputs a binary value, *i.e.* $t : \mathbb{R}^K \to \{0, 1\}^K$.

The output of the label prediction module are also referred to as class scores. Suppose we can write

$$f(\boldsymbol{x}) = \begin{bmatrix} f(\boldsymbol{x})_1 \\ f(\boldsymbol{x})_2 \\ \cdots \\ f(\boldsymbol{x})_K \end{bmatrix},$$

then we expect $f(\boldsymbol{x})_k$ to be large if the classifier is confident that λ_k is associated with $\boldsymbol{x}$ and small otherwise, since it is trained to match the binary ground truth outputs. The absolute values of these scores are less important, but we at least want $f(\boldsymbol{x})_u > f(\boldsymbol{x})_v$ for any $\lambda_u \in Y$ and $\lambda_v \notin Y$. From this output we can then obtain a ranking of the labels from high to low according to how confident the classifier is that each label is present. This is done by a ranking

function, $rank(\cdot)_k$, which maps the outputs of f, $\{f(\boldsymbol{x})_1, f(\boldsymbol{x})_2, \ldots, f(\boldsymbol{x})_K\}$, to $\{1, 2, \ldots, K\}$ such that if $f(\boldsymbol{x})_u > f(\boldsymbol{x})_v$ then $rank(\boldsymbol{x})_u < rank(\boldsymbol{x})_v$.

However, in MLC, we ultimately want to know which labels are associated with each input and thus the need for $t(\cdot)$. A typical choice for $t(\cdot)$ is

$$t(a) = \begin{cases} 1 & \text{if} \quad a > 0.5 \\ 0 & \text{otherwise} \end{cases}.$$

In some cases the threshold is chosen to optimise certain metrics (Read *et al.*, 2011). This optimal threshold may vary across different labels. Estimating $t(\cdot)$ can also be treated as a learning problem, *i.e.* estimate it from the data (Zhang *et al.*, 2006, RamóN Quevedo *et al.* (2012), Li *et al.* (2017)). These approaches are discussed in **??**. An example multi-label classifier pipeline may look something in the lines of:

$$\boldsymbol{x} \rightarrow \begin{bmatrix} 0.7 \\ 0.1 \\ 0.2 \\ 0.9 \\ 0.4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

We may choose to evaluate a multi-label classifier based on its real output (ranking measures) or on its binary output (classification measures). These are discussed next.

2.2.2 Metrics

There are a plethora of metrics to evaluate the performance of multi-label classifiers. According to (Dembszynski *et al.*, 2010), it is essential to evaluate multi-label learners on multiple and contrasting measures because of the additional degrees of freedom that the multi-label setting introduces. This provides a better evaluation of the capabilities of an algorithm and helps identifying algorithms which perform well across a range of evaluation measures.

For the following definitions, recall that $\boldsymbol{y}_i$ is the indicator vector of true labels for observation $\boldsymbol{x}_i$ and $\hat{\boldsymbol{y}}_i$ the indicator vector of predicted labels for the same observation, produced by $h_{\hat{\theta}}(\boldsymbol{x}_i)$, which we will also denote as $\hat{h}$ for brevity. Also, let the predicted output of the label prediction module, $\hat{f}$, be

denoted as $\boldsymbol{p}_i$, *i.e.*:

$$f_{\hat{\theta}}(\boldsymbol{x}_i) = \begin{bmatrix} p_{i1} \\ p_{i2} \\ \vdots \\ p_{iK} \end{bmatrix} = \boldsymbol{p}_i.$$

Suppose the output corresponding to the k-th label is contained in a N-sized vector indexed by (k), *i.e.*

$$\boldsymbol{y}_{(k)} = \begin{bmatrix} y_{1k} \\ y_{2k} \\ \vdots \\ y_{Nk} \end{bmatrix}, \quad \boldsymbol{p}_{(k)} = \begin{bmatrix} p_{1k} \\ p_{2k} \\ \vdots \\ p_{Nk} \end{bmatrix}.$$

In binary classification, the (dis)similarity between the ground truth binary vector, $\boldsymbol{y}$, and the predicted vector (binary, $\hat{\boldsymbol{y}}$, or real-valued, $\boldsymbol{p}$) is computed. For our discussion the following binary classification- and ranking-based evaluation metrics computed on n-sized vectors, are of importance:

- *error rate*:
$$\text{err}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_i),$$
where $I(\cdot)$ is the indicator function.

- *F_1-score*:
$$F_1(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{2 \sum_{i=1}^{n} y_i \hat{y}_i}{\sum_{i=1}^{n} y_i + \sum_{i=1}^{n} \hat{y}_i}$$

- *Average Precision (AP)*:
$$\text{AP}(\boldsymbol{y}, \boldsymbol{p}) = \frac{1}{\sum_{i=1}^{n} y_i} \sum_{i;y_i=1} \frac{\sum_{j;y_j=1} I(p_j \geq p_i)}{rank(p_i)},$$
where $rank(\cdot)$ sorts the elements in $\boldsymbol{p}$ from highest to lowest, returning values in $\{1, 2, \ldots, n\}$. It evaluates the average fraction of relevant labels ranked above a specific relevant label, which equals to one if no negative labels are ranked higher than positive labels.

- *Area under the ROC curve (AUC)*:
$$AUC(\boldsymbol{y}, \boldsymbol{p}) = \frac{\sum_{u;y_u=1} \sum_{v;y_v=0} I(p_u \geq p_v)}{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} (1 - y_i)},$$
which calculates the proportion of correctly ordered positive-negative label pairs.

Since the output of a MLC problem is also just a set of binary outcomes, binary classification evaluation metrics can be harnessed to evaluate multi-label classifiers. It just needs to be decided how the binary measures are aggregated over the entire dataset.

There are three possible aggregation schemes: per *example*, *micro*- and *macro*-averaging. In some literature (Zhang and Zhou, 2014), micro- and macro-averaging are grouped into *label*-based averaging, however we find the example-micro-macro grouping more descriptive.

Suppose a binary performance metric comparing (dis)similarity between two binary vectors is denoted as B, *i.e.* $B \in \{\text{err}, F_1, \text{AUC}, \dots\}$. Example-based multi-label evaluation metrics can then be given as:

$$\frac{1}{N}\sum_{i=1}^{N} B(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i),$$

which is the metric computed for each observation (or example) and then averaged over all observations in the dataset. An example of an example-based metric is the so called *Hamming Loss* for which B is the binary missclassification rate computed per observation, *i.e.*

$$\text{hloss}(\hat{h}) = \frac{1}{N}\sum_{i=1}^{N} \text{err}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{K}\sum_{k=1}^{K} I(y_{ik} \neq \hat{y}_{ik}).$$

Thus the Hamming loss computes the average number of incorrectly predicted labels for each observation and takes the average over all observations. The Hamming loss is not very informative when the ratio of positive to negative labels is imabalanced. For example, if we predict all zero's for ChestX-ray14, the Hamming loss is overly confident at 0.0516 (the same as the label density). For cases like these it may be more suitable to use $B = F_1$ which is known to be less sensitive to class imbalance.

A much stricter example-based metric is the *Subset Accuracy* which is defined as:

$$\text{subsetacc}(\hat{h}) = \frac{1}{N}\sum_{i=1}^{N} I(\boldsymbol{y}_i = \hat{\boldsymbol{y}}_i)$$

and thus computes the proportion of observations that were perfectly predicted by $\hat{h}$. This gives no in-between reward if $\hat{h}$ is 'almost' correct for a given observation.

Ranking based example-based metrics are any binary metrics measuring the ranking performance of a classifier, averaged over the examples in the dataset, *i.e.* $B \in \{AUC, AP\}$.

Macro-averaged multi-label metrics is given by:

$$\frac{1}{K}\sum_{k=1}^{K} B(\boldsymbol{y}_{(k)}, \hat{\boldsymbol{y}}_{(k)})$$

and thus calculates B for each label and averages it accross all labels. The micro-averaging process treats the multiple labels as a single variable and then computes a binary measure, *i.e.*

$$B(\boldsymbol{y}^*, \hat{\boldsymbol{y}}^*),$$

where

$$\boldsymbol{y}^* = \begin{bmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{1K} \\ y_{21} \\ y_{22} \\ \vdots \\ y_{NK} \end{bmatrix}, \quad \hat{\boldsymbol{y}}^* = \begin{bmatrix} \hat{y}_{11} \\ \hat{y}_{12} \\ \vdots \\ \hat{y}_{1K} \\ \hat{y}_{21} \\ \hat{y}_{22} \\ \vdots \\ \hat{y}_{NK} \end{bmatrix}.$$

Note that the hamming loss is equivalent to the err^{macro} and err^{micro}. A more complete taxonomy of multi-label evaluation metrics can be found in (Wu and Zhou, 2016).

The most common measures reported in multi-label image classification are F_1^{micro}, F_1^{macro} and the *mean average precision* (mAP), which is the name given to macro-averaged AP (Zhu *et al.*, 2017). It may also be useful to evaluate a multi-label classifier on a per label level, depending on the application. For example in (Wang *et al.*, 2017, Yao *et al.* (2017), Rajpurkar *et al.* (2017)) they reported the per-label AUC scores for ChestX-ray14. Therefore, in our experiments we will report the per-label AUC along with the F_1^{macro}, F_1^{micro} and mAP scores so that we can compare our methods to existing benchmarks and also provide a greater variety of metrics to compare against for future work on ChestX-ray14.

In order to get a feel for some of the discussed multi-label metrics, we provide the following toy example. We evaluate the predictions of three classifiers,

Table 2.3: Toy predictions.

i	${}_1\hat{f}(\boldsymbol{x}_i)$	${}_2\hat{f}(\boldsymbol{x}_i)$	${}_3\hat{f}(\boldsymbol{x}_i)$
1	$(0.1, 0.9, 0.4, 0.3, 0.2)^\intercal$	$(0.2, 0.4, 0.1, 0.4, 0.3)^\intercal$	$(0.1, 0.4, 0.1, 0.7, 0.2)^\intercal$
2	$(0.6, 0.7, 0.8, 0.9, 0.6)^\intercal$	$(0.3, 0.2, 0.7, 0.9, 0.8)^\intercal$	$(0.7, 0.1, 0.9, 0.8, 0.7)^\intercal$
3	$(0.1, 0.4, 0.2, 0.1, 0.3)^\intercal$	$(0.1, 0.9, 0.2, 0.3, 0.8)^\intercal$	$(0.9, 0.8, 0.1, 0.2, 0.6)^\intercal$

${}_1\hat{h}$, ${}_2\hat{h}$ and ${}_3\hat{h}$ on three data points with ground-truth labels, $Y_1 = \{\lambda_2\}$, $Y_2 = \{\lambda_1, \lambda_3, \lambda_4, \lambda_5\}$ and $Y_3 = \{\lambda_2, \lambda_5\}$, with $K = 5$. Suppose the three classifiers produced the predictions contained in Table 2.3 and that we choose to threshold the values at 0.5 to determine the final binary output. Their performance in terms of multiple multi-label metrics are reported in Table 2.4, with the best performance in terms of each metric is highlighted in bold.

Table 2.4: Performance of the toy predictions in Table 2.3.

	${}_1\hat{h}$	${}_2\hat{h}$	${}_3\hat{h}$
hloss	**0.2**	**0.2**	**0.2**
subsetacc	**0.3333**	0	**0.3333**
AP^{example}	0.5153	0.6667	**0.6778**
F_1^{micro}	0.7692	0.7273	**0.8**
F_1^{macro}	0.8333	**1**	0.8

Some important observations from this example:

- No one classifier is best in terms of all metrics.
- There is no difference between the classifiers in terms of the hamming loss.
- ${}_1\hat{h}$ and ${}_3\hat{h}$ is equivalent in terms of both hamming loss and subset accuracy, whereas ${}_3\hat{h}$ has the better $AP^{example}$ and ${}_1\hat{h}$ the better F_1^{micro} and F_1^{macro}.
- F_1^{micro} and F_1^{macro} measure different aspects of the predictions.

A final comment about multi-label metrics is that they are usually non-convex and discontinuous (Zhang and Zhou, 2014) and therefore multi-label classifiers resort to optimising surrogate metrics which are easier to optimise. Most multi-label classifiers learn from the training observations by explicity or implicitly optimising one specific metric (Zhang and Zhou, 2014). That is why in (Dembcz *et al.*, 2012) the authors reccomended specifying which of the metrics a new proposed algorithm aims to optimise in order to show if it is succesful. But at the same time it is important to test the algorithm on numerous metrics for fair comparisons against other algorithms (Zhang and Zhou, 2014, Madjarov *et al.* (2012)).

Other than predictive performance, there are other aspects on which multi-label classifiers can be evaluated, such as its efficiency. Multi-label algorithms

should be efficient in the sense that it strives to take the least amount of computational power for a given level of predictive performance (Madjarov *et al.*, 2012). These classifiers can take a considerable amount of time to train when complicated ensembles are being implemented on datasets with huge labelsets. In cases where live updating and predictions are needed, this may be a problem.

So far we have only defined the evaluation metrics on an arbitrary set of points, $i = 1, 2, \ldots, N$. However, on which data points a metric is evaluated is an extremely important consideration. The interpretation of a metric computed on the same data used to build the model is vastly different than on an independent dataset. We discuss these differences and how to obtain these sets for multi-label data in the next section.

2.2.3 Splitting Multi-Label Data

The collection of data points used to train an algorithm is called the *training* dataset. We can evaluate the performance of the classifier on the training set, but this will however give an overly confident estimate of the prediction accuracy, since the classifier was specifically trained to minimise its error on the training set. What we are more interested in is the prediction accuracy on a totally new set of data points, unseen by the classifier during training. This set of points we refer to as the *test* dataset and a a model's prediction performance on such an independent test set is known as its *generalisation* ability (Hastie *et al.*, 2009, Chp. 7).

Thus given a dataset for a learning problem, we must decide how to split the data into a training and a test set. It is often useful to set aside an additional set of data points for estimating the peformance of different models in order to choose the best one. Such as a set is known as the *validation* set. We should avoid using the test set for this task, since if we specifically choose a model with the best test performance, the performance will be biased towards the test set. Thus the test should ideally only be used at the end of a data analysis for estimating the final chosen model's generalisation performance.

It is difficult to determine how many (and which) observations should be assigned to each set. We want the training set to be large for better model training, but we also want the test and validation sets to be large for more accurate performance estimates. In all the previous work on ChestX-ray14

(Wang *et al.*, 2017, Yao *et al.* (2017), Rajpurkar *et al.* (2017)), the data was randomly split into a training, validation and test set according to the following proportions: 70%, 10% and 20%.

Another way of estimating a model's generalisation ability is to use *cross-validation*. Cross-validation divides the available data points into equal groups, usually chosen as 5 or 10. In turn, each group acts as the test set whilst the observations in the remaining groups are used in the training set. The performance estimates on each holdout group are then averaged to obtain a final estimate of a model's perfomance. One of the advantages of this approach is that is possible to also compute a standard deviation for the error estimate.

Cross-validation is typically used when the dataset is small, since it allows us to allocate more data to the training set, whilst testing the model on all the available observations. The disadvantage is that it requires a model to be fitted multiple times (as many times as the number of groups), not ideal when the training process is computationaly expensive. Note the notion of what is "small" data is not defined and depends completely on the task and the nature of the data.

The number of observations in each set is not the only consideration when deciding on how to split the data. A key property of the validation and test sets is that they must be representative of the new data coming from the same underlying distribution. In our case, new X-rays, either from patients already in the dataset or from totally new patients. Therefore to emulate this process, patient ID should be taken into account when splitting the data. We must ensure that X-rays in the validation set are either not from the same patients that are in the training set or if from the same patients, taken at a future date. The same should hold for the test set, with respect to the training and validation sets. (Wang *et al.*, 2017) and (Yao *et al.*, 2017) both ignored patient ID when distributing the X-rays. (Rajpurkar *et al.*, 2017) took an overly conservative approach and ensured that there were no patient overlaps between the sets. We found this unecessary since we would also want to evaluate a model's perfomance on new X-rays but from the same patients.

One final consideration for splitting multi-label data is the label distribution in each set. If the split is done randomly, it may construct groups lacking sufficient positive examples of rare labels, due to class imbalance problems (Szymański and Kajdanowicz, 2017). Not only does this cause calculation problems for some evaluation metrics, but also limits the generalisation abilities

of a model trained on a fold that did not include classes available in the test set. In addition, evaluating a model on a fold lacking rare, difficult to detect labels, may report biased results.

To prevent this from happening (Sechidis *et al.*, 2011) introduced an iterative stratified sampling approach which takes into account the existence of disjoint groups within a population and produces samples where the proportion of these groups is maintained. This algorithm was extended in (Szymański and Kajdanowicz, 2017) to take into account second-order relationships between labels. To validate the effectiveness of their stratified sampling approach, they compared the label distributions in each group based on some statistical properties and also checked the performance of classifiers trained on these different groups. They found that their approach exhibits greater stability on these fronts.

The largest dataset used in the experiments in (Szymański and Kajdanowicz, 2017) had $N = 43907$. This is less than half the size of ChestX-ray14. We argue that with the larger the dataset and the smaller the PLDiv (number of label combinations per observation), stratified sampling becomes less needed. In fact, (Wang *et al.*, 2017) found insignificant performance difference of their classifiers with different random splits and (Yao *et al.*, 2017) found that the classifier performance on their validation and test sets were consistent with each other. If we assume the test set that we want to generalise to is also a random subset from all available data, it seems unecessary to introduce structure in the splits based on target info.

To test the stability of a random split for ChestX-ray14, we conduct a small experiment comparing the statistical properties of such splits, using the metrics proposed in (Sechidis *et al.*, 2011) and (Szymański and Kajdanowicz, 2017). Our splitting procedure for this experiment is a 5-fold cross-validation and repeated 10 times. The metrics we calculate for each split is as follows:

- *label distribution.*

$$LD = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{J}\sum_{j=1}^{J}\left|\frac{\sum_{i\in F_j} y_{ik}}{N_j - \sum_{i\in F_j} y_{ik}} - \frac{\sum_i y_{ik}}{N - \sum_i y_{ik}}\right|\right),$$

  where F_j, $j = 1, 2, \ldots, J$, is the index of observations in the j-th fold and N_j the number of observations in the j-th fold. LD evaluates the extent to which the distribution of positive and negative examples of each label

in each subset, follows the distribution of that label in the whole dataset. A value close to zero indicates that the distribution is similar across folds.

- *label pair distribution.*

$$LPD = \frac{1}{K^*}\sum_{k=1}^{K^*}\left(\frac{1}{J}\sum_{j=1}^{J}\left|\frac{\sum_{i\in F_j} y^*_{ik}}{N_j - \sum_{i\in F_j} y^*_{ik}} - \frac{\sum_i y^*_{ik}}{N - \sum_i y^*_{ik}}\right|\right),$$

where y^*_{ik} indicates the presence of the k-th label-pair for observation i, for $k = 1, 2, \ldots, K^*$. This is an extension of LD to compare the positive-negative ratio of label-pairs accross the folds. Another suggested statistic is the *example distribution* (ED) which evaluates how much a given fold's size deviates from the desired number of observations in that fold. This statistic is not relevant in our case, since with random 5-fold cross-validation and N dividable by 5, the size of each fold will always be exaclty what we desire, *i.e.* $ED = 0$.

- *FZ* - the number of folds that contain at least one label with no positive examples.

- *FLZ* - the number of fold-label pairs with no positive examples.

- *FLPZ* - the number of fold-*label-pair* pairs with no positive examples. The convention is to not count the label-pairs with a number of positive examples less than the number of splits.

For all of these statistics, close to zero is better. In Table 2.5 we summarise the results of these experiments by reporting the mean and standard deviation of these metrics accros each splitting iteration. Comparing these results to (Sechidis *et al.*, 2011) and (Szymański and Kajdanowicz, 2017) we find that the statistics are extremely small with a small standard deviation, meaning that random splitting for ChestX-ray14 produces stable and consistent folds.

Table 2.5: Statistical properties of the subsets produced by 5-fold cross-validation, repeated 10 times.

	mean	std
LD	0.0011	0.0001
LPD	0.0016	0.0001
FZ	0.0000	0.0000

	mean	std
FLZ	0.0000	0.0000
FLPZ	0.0231	0.0042

For all of the above mentioned reasons, we decide to use random sampling when splitting the data into different sets. When the process of training a classifier is not to computationaly demanding, we will use 5-fold cross-validation to estimate the generalisation perfomance of a model. Otherwise, we will split the data according to previous work, *i.e.* train: 70%, validation: 10% and test: 20%. We also ensure that if there is patient overlap between a test and training set, the X-ray(s) in the test set is taken at a date after the X-ray(s) taken in the training set.

In this chapter we did a thorough exploration of the ChestX-ray14 data and highlighted the some of the challenges we might face in future analyses. We also looked at how models on this dataset should be evaluated and described the metrics and splitting procedures we will use in following analyses. With this foundation we are now ready to explore approaches for MLC and image classification, which is the main focus of the remainder of the thesis.

Chapter 3

Multi-Label Classification Methods and Neural Networks

Multi-label classification can be viewed as a generalisation of the conventional single-label classification task (binary or multiclass). In a MLC problem, each observation in the dataset may be associated with more than one label and the task is to predict a label set, whose size is unknown *a priori*, for each unseen observation. There are plenty of real-world applications that fit into this framework: an image annotation problem where each image contains more than one semantic object (Zhu *et al.*, 2017), a text classification task where each document has multiple topics (Liu *et al.*, 2017) or an acoustic classification task where the recordings contain the sounds of multiple bird species (Zhang *et al.*, 2016), to name a few. The realisation of the field's applicability to real-world problems is probably what drives the increasing research interest (see Figure 3.1).

The generality of MLC naturally introduces more complexity to the classification task. We have already seen some of the challenges in Chapter 2 such as: evaluating multi-label models, dealing with label correlation, label imbalance, etc. In the first half of this chapter we investigate some of the representative approaches to solving the MLC problem. Thie goal of this discussion is to put our chosen approach into perspective with other existing approaches. Our approach is based on neural networks for MLC and therefore, in the second half of this chapter, we zoom in on neural networks and how they are adapted to predict multiple outputs.

MLC methods are in abundance and developing at a rapid pace. Approaches

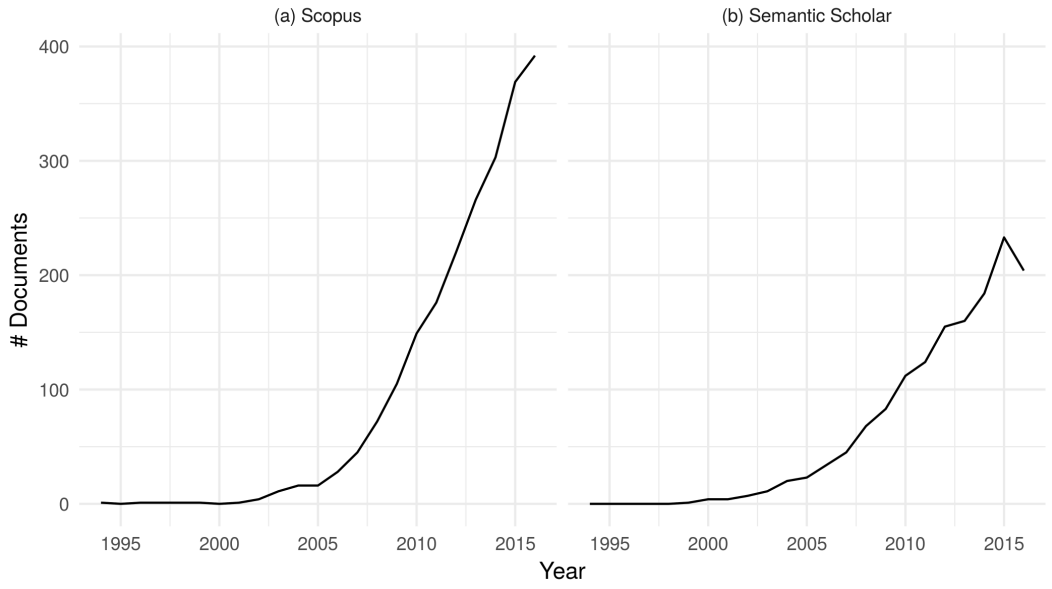

Figure 3.1: Line graphs illustrating the rise in multi-label learning publications per year for two databases. The database searches were done on 24-03-2017. The searches were not identical since they were limited to the search features of the databases. (a) The search on Scopus (cite) was for all documents (conference papers, articles, conference, articles in press, reviews, book chapters and books) in any subject area with either the words *multi-label* or *multilabel* and either the words *learning* or *classification* found in either their titles, abstracts or keywords. (b) The search on Semantic Scholar was based on machine learning principles and thus automatically decides which research documents are relevant to a specific search query. The query used was *multilabel multi-label learning classification*. The search only returns research in the computer science and neuroscience fields of study. More technical details can be found on the respective engine's websites (probably put details in appendix).

to MLC are generally categorised into *Problem Transformation* (PT) and *Algorithm Adaptation* (AA) methods (Zhang and Zhou, 2014). A further grouping is often given, namely *Ensemble Methods* (See Figure 3.2). These are respectively discussed in Section 3.1, Section 3.2 and **??**. In Section 3.3 we look at more complicated approaches to thresholding the real output of a classifier. In Section 3.4 we introduce artificial neural networks and discuss there structure, how they are trained and how they are adapted for the MLC problem.

- The question is whether or not harnessing label dependencies will improve predictive accuracy. Unfortunately this question cannot be answered in a blanket way since it is totally dependent on the problem, influenced

by factors such as the properties of the data and the loss function to be minimised [Dembszynski2010]. If we inappropriately model label dependence it can add unecessary complexity and additional noise to the problem.

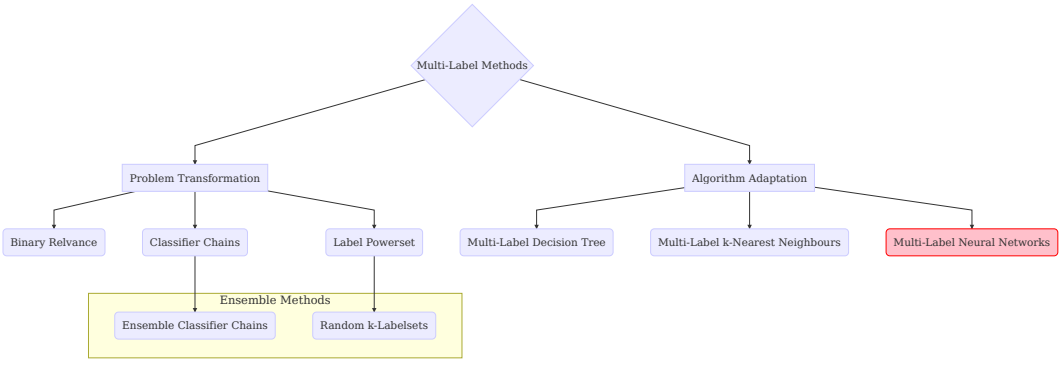

Figure 3.2: Diagram of the categorisation of MLC methods.

3.1 Problem Transformation

PT methods consist of first transforming the multi-label problem into one or more single-label problem(s) and then fitting any standard supervised learning algorithm(s) to the single-label data. For that reason, PT methods are called algorithm independent, since once the data is transformed, any single-label classifier can be used (Tsoumakas and Vlahavas, 2007).

The two main problem transformation algorithms are the *binray relevance* (BR) and *label powerset* (LP) transformations. They form the basis of most problem transformation methods. The SotA problem transformations algorithms are most of the times extensions of either the standard BR or LP algorithms (Alazaidah and Ahmad, 2016).

3.1.1 Binary Relevance

The BR algorithm transforms the mutli-label task into K binary classification problems by modelling each of the labels separately. Thus K single label datasets are constructed from the multi-label dataset,

$$(\boldsymbol{x}_i, y_{ik})_{i=1}^{N},$$

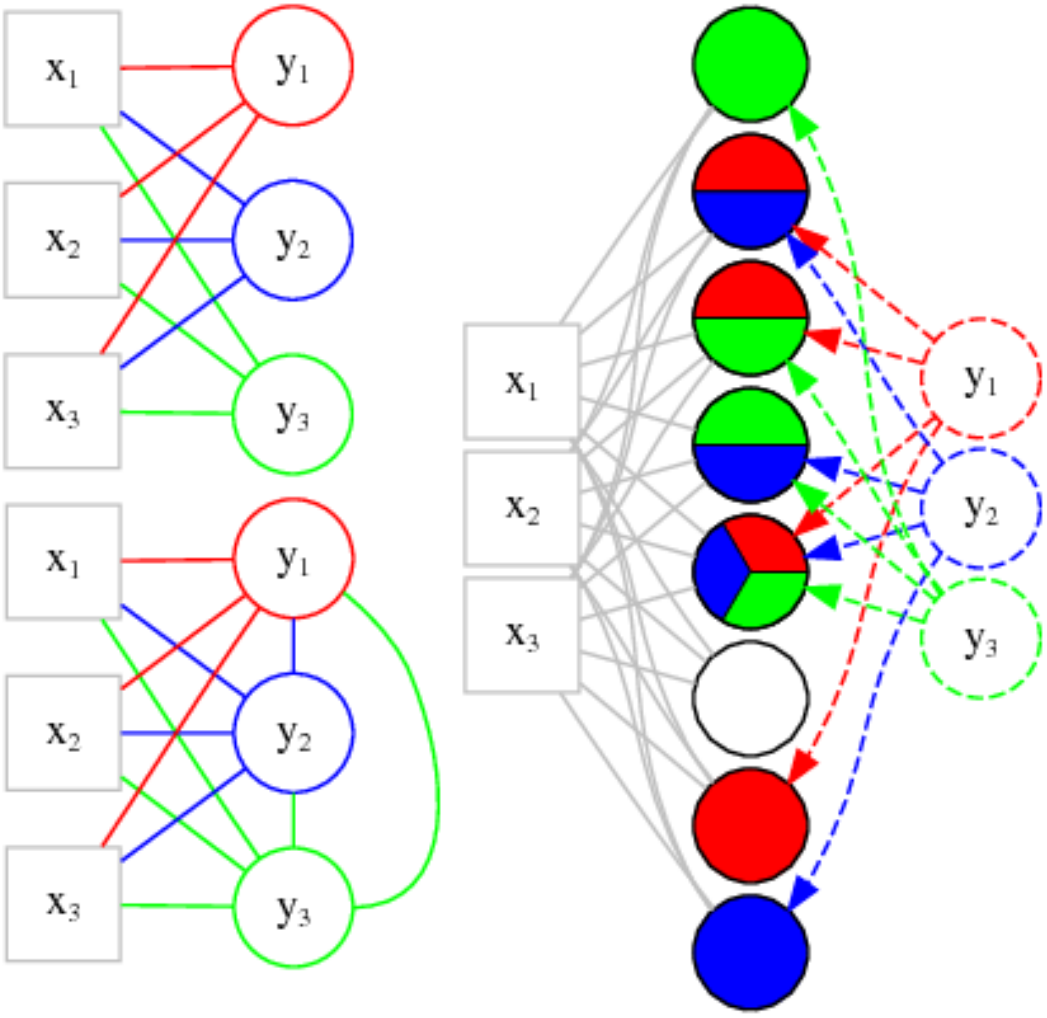

Figure 3.3: Diagrams illustrating various problem transformation algorithms; (a) Binary relevance, (b) Label powerset and (c) Classifier chains.

for $k = 1, 2, \ldots, K$ and a classifier of the form $h_k : \mathbb{R}^p \to \{0, 1\}$ is trained on each dataset separately. To predict the labels for an observation $\boldsymbol{x}$, the input is fed to each of the K classifiers and their outputs are combined to determine the multi-label output, *i.e.* $\left(\hat{h}_1(\boldsymbol{x}), \hat{h}_2(\boldsymbol{x}), \ldots, \hat{h}_K(\boldsymbol{x})\right)^\intercal$. See Figure 3.3 (a) for an illustration of the BR transformation applied to a scenario with $p = 3$ and $K = 3$. Notice how there are no connections between the labels.

The BR transformation makes the very strong assumption of total label independence, *i.e.* the presence or absence of a label is not influenced by the presence or absence of other labels. Obviously this assumption limits the performance of a classifier when the labels are strongly correlated. There are some benefits to this assumption however. Firstly, the simplicity of the

approach protects against overfitting label dependencies. The complexity of BR scales linearly with increasing K, which compares favourably against some of the other PT approaches. Nevertheless, it remains inconvenient to separately fit K different models for large K.

It is possible for BR to predict any combination of labels for an input, in contrast to some of the other methods such as LP which we will discuss shortly. This can be seen as an advantage since it is not restricted to label combinations that are only in the training set. But this may also lead to label combination predictions that would never occur in real life. (Read *et al.*, 2011) mentions that the label cardinality of the predicted labels produced by BR may differ greatly from the cardinality of the actual labels. But this flexibility of the BR predictions make it ideal for dynamic learning evironments, where the labels to be predicted may change over time, for instance.

In scenarios of high label cardinality, large K and large label correlatinos, (Luaces *et al.*) found that BR performed surprisingly well empirically against other methods that take label correlations into account, probably because of its simplicity. This was especially true when looking at macro-averaged metrics, since these metrics evaluate the model on a per label basis. (Dembcz *et al.*, 2012) proved that the BR approach is the risk minimiser of the Hamming Loss but performs poorly in terms of the subset accuracy, which evaluates the quality of the predicted label combinations as a whole.

BR is considered the baseline method for MLC. Most of the modern work suggest how label correlations can be taken into account and show how it outperforms BR on certain datasets and specific metrics. But again, there is no one solution that outperforms BR consistently.

Direct extensions of BR that can learn from label correlations but preserves some of the simplicity of BR have been suggested (see **?**, Tsoumakas *et al.* (2009), Godbole and Sarawagi (2004), Pachet and Roy (2009)). The basic idea is a two-stage approach. In the first stage a standard BR algorithm is applied to obtain predictions for each label and then in the second stage those predictions are used as input to for the following round of training. The finer detail is beyond the scope of this thesis. These extensions do not guarrantee an improvement. Some label pairs might have no correlation and adding predictions for those labels as inputs to the second round of training will add noise to the model and waste computation time. (Tsoumakas *et al.*, 2009) suggests a solution called correlation-based pruning. They estimate the pairwise

correlations between labels, ϕ, and only include initial predictions of labels as inputs that are highly correlated with the specific label to be predicted.

One PT algorithm that can also be seen as a variation of BR that take label dependencies into account is the *Classifier Chains* (CC) algorithm (Read *et al.*, 2011). This is a common algorithm for MLC, also with some interesting extensions, and therefore we discuss it in the next section.

3.1.2 Classifier Chains

The CC algorithm also consists of transforming the mutli-label data set into K single-label data sets but the transformations are done sequentially in the sense that the label previously treated as a target will be added as a feature for predicting the next label. The produced datasets can be given as follows:

$$(\boldsymbol{x}_i^*, y_{ik})_{i=1}^N,$$

where $\boldsymbol{x}_i^* = (x_{i1}, x_{i2}, \ldots, x_{ip}, y_{i1}, y_{i2}, \ldots, y_{i(k-1)})$. To each of these single-label data sets a classifier, $h_k : \mathbb{R}^{p+k-1} \to \{0, 1\}$, is trained and then their predictions are combined in the same fashion as BR. See Figure 3.3 (b) for an illustration of this transformation. Notice how the previously predicted labels are used as inputs to predict a given label in a chain like fashion, illustrated by the colored connections between the output nodes. CC thus assumes a conditional label dependence between some labels.

To make a prediction with CC, we cannot use y_{ik} as inputs to the classifier, since this information is typically not available in a test setting. Therefore we use $\hat{y}_{ik}$, obtained by h_k. It is also possible to use the real output f_k instead of h_k as input to the classifiers later in the chain (Dembczy, 2010).

Notice that if a label used early in the chain is poorly predicted it may have a negative influence on the predictions of the labels in the remainder of the chain. In addition, the labels at the end of the chain is provided information on all the labels in the chain before it, whereas the labels at the start of the chain do not have that privillege. Ideally we would want the labels that are easy to predict and not dependent on other labels to be at the start of the chain and the labels that depend on other labels to be placed after those labels in the chain. However there is not obvious way to find the optimal label order. The original proposal suggested a random order and claimed that in many cases

this has little influence. However in an experiment [Dutoit2017] found evidence of the contrary.

It also not feasible to experiment with all the possible label orders for large datasets. Therefore it is useful if we can estimate the best label order from the data. Some suggestions from the literature is to order the labels based on the initial accuracy of each label using BR, placing the easily predicted labels first [Keika2016, ordered ccs] and ordering the labels based on feature importance and correlation scores [Dutoit2017]. We are not going into the detail here.

On way to reduce the effect of label order is to randomly try different label orders and combine the predictions obtained from the CC for each permutation. This method is called *Ensemble of Classifier Chains* (ECC) (Read *et al.*, 2011). ECC also trains each CC on a different subset of the training dataset to increase the uniqueness of each classifier, helping with variance reduction when aggregating the output of each CC in the ensemble. Ensembling is popular technique used to increase the accuracy of statistical learning algorithms, therefore it is hard to determine whether ECC handles the label correlations better than CC or that the performance improvement is actually due to the ensembling procedure. More on CC based methods can be found in [Dutoit2017].

3.1.3 Label Powerset

On the other side of the spectrum of problem transformation approaches, is the label powerset (LP) algorithm. LP treats each combination of labels as a distinct class and then a standard multiclass classifier can be applied to learn theses classes (Tsoumakas and Katakis, 2007). The transformation is illustrated in Figure 3.3 (c). See how the original labels are mapped to a space where there is a single point for each possible label combination. Thus no observation can belong to more than one class in the transformed space and therefore a multiclass problem. There are 2^K possible label combinations but using the LP algorithm for MLC restricts one to only learn label combinations that exist in the training data, which may be a lot less than 2^K. Nevertheless, the number of classes after the powerset transformation can be very large for large K which makes multiclass classifiers inefficient. It also quite common that some label combinations appear relatively infrequently in the training set and thus introduces an imbalance class distribution, again difficult for classifiers to

handle (Xu *et al.*, 2016).

The advantage of using the LP algorithm is that it models the full conditional dependencies of the labels. (Dembcz *et al.*, 2012) shows that LP optimises the subset accuracy but usually fails for losses like the Hamming Loss.

Again there are multiple proposals on how to extend LP to keep its advantages and reduce its limitations. (Read, 2008) suggested to prune the number of classes after the powerset transformation, *i.e.* eliminate some of the infrequent label combinations. This reduces the complexity of the multiclass problem but not without other drawbacks. Firstly it is difficult to determine the threshold for which of the label combinations to be eliminated and by eliminating these combinations introduces bias in the predictions.

RAKEL (Random k Labelsets) and *HOMER* (Hierachy of multi-label classifiers) (Tsoumakas *et al.*, 2008) are both examples of ensembling approaches which attempt to improve the base LP algorithm by combining the output of simpler classifiers. They achieve this by first clustering the labels and then fitting classifiers to these clusters. The advantages of these methods typically lie in the reduction of the output space dimension of the fitted classifiers and the variance reduction effects of ensembling. However ensembling typically requires more computing time. Luckily some of the models can be trained in parallel.

The previous sections roughly cover the representative PT approaches to MLC. Next we will look at algorithm adaptation approaches and see where neural networks for MLC fit in.

3.2 Algorithm Adaptation

AA methods tackle the MLC task by adapting, extending and/or customising an existing supervised learning algorithm (Madjarov *et al.*, 2012). The main weakness of AA methods is that they are mosty tailored to suit a specific model, whereas problem transformation methods are more general and allows for the use of many well-known and effective single-label models (Systems and Aviv-yafo, 2014). We place neural networks for MLC in this category. Here we look at a simple example of another algorithm in this category.

3.2.1 Multi-Label k-Nearest Neighbour (ML-kNN)

There are multiple proposals to adapt the kNN algorithm to suit MLC - the most common of these methods is the *Multi-Label k-Nearest Neighbour* (ML-kNN) algorithm (Zhang and Zhou, 2007). Suppose we are interested in predicting the labels for observation $\boldsymbol{x}$. Just like the conventional kNN algorithm the ML-kNN algorithm first determines the k closest traininf points to $\boldsymbol{x}$ in the input space, which we denote as $\mathcal{N}(\boldsymbol{x})$. In a binary or multiclass problem, the output of kNN is a simple aggregation (majority vote of average) of the label counts accros the observations in $\mathcal{N}(\boldsymbol{x})$. The output of ML-kNN is also based on the label counts, but aggregated in a different way to accomodate for multiple labels.

Suppose C_k denotes the number of points in $\mathcal{N}(\boldsymbol{x})$ associated with label λ_k and let $P(y_{ik} = b|C_j)$ be the probability that $y_{ik} = b$ for $b \in \{0, 1\}$ given that there are exactly C_k points in $\mathcal{N}(\boldsymbol{x}_i)$ with label λ_k. Then the output of ML-KNN can be given by:

$$\begin{aligned}\hat{y}_{ik} &= \arg\max_{b\in\{0,1\}} P(y_{ik} = b|C_k) \\ &= \arg\max_{b\in\{0,1\}} P(y_{ik} = b)P(C_k|y_{ik} = b) \quad \text{(Bayes' Rule)}\end{aligned}$$

Therefore all we need to predict the relevance of label λ_k for observation $\boldsymbol{x}_i$ is the prior probability $P(y_{ik} = b)$ and the posterior probability $P(C_k|y_{ik} = b)$. Both of these quantities can be estimated from the data using label frequencies in the training set. For example $P(y_{ik} = 1)$ can be estimated by calculating the proportion of observations in the training set associated with λ_k. More detail can be found in (Zhang and Zhou, 2007).

Some of the advantages of using ML-kNN is that it complexity scales linearly with increase in K and that it is easy to implement. Otherwise it suffers from the same limitations of ordinary kNN such as the large memory requirements for prediction and the sensitivity to the *curse of dimensionality* (Hastie *et al.*, 2009, p. 20 ??).

The above sections served as an overview of the MLC methods taxonomy. An often overlooked aspect of MLC is the thresholding of the continuous outputs to obtain binary outputs. Most of the time a threshold of 0.5 is used, which is rarely optimal. In the next section we look at more advanced ways to go from continuous outputs to multiple label predictions which can considerably improve the performance of a classifier.

3.3 Thresholding Strategies

We learned that a multi-label classifier, h, consists of a label prediction module, f, and a thresholding function, t. $t : \mathbb{R}^K \to \{0, 1\}^K$ maps the real-output of the label prediction module (usually in $[0, 1]$) for all labels to binary values. In multiclass classification, the convention is to assign a 1 to the label with highest score given by f and zero the rest of the labels. In binary classification this translates to mapping the continuous output to 1 if it exceeds 0.5. Since MLC can be decomposed into multiple binary classification tasks, a threshold of 0.5 is usually used for each label to obtain binary values from the continuous output.

Depending on the desired metric to be optimised, this thresholding strategy may not be optimal. For instance, if recall is important to the application, it will probably be better to chose a smaller threshold to make sure less of the positive labels are missed. The results may improve even further if different thresholds are used for each label. Suppose for some reason the classifier learned to be biased towards a certain label, then a threshold for this label can be chosen to be larger than the other labels' thresholds to try and correct the bias. However finding these optimal thresholds are tricky which is probably why most users resort to the simple 0.5 threshold.

The simplest method to finding a possibly better threshold than the default 0.5 is by a simple search over a pre-selected set of threshold values (for example $\{0.1, 0.2, \ldots, 0.8, 0.9\}$). The binary predictions obtained using each of these threshold values can be evaluated using any desired multi-label metric and then we can choose the threshold that results in the best performance. The evaluation is typically done on a validation set or incorporating it into the cross-validation procedure to prevent biased error estimates. (Read *et al.*, 2011) found that choosing a threshold so that the label cardinality of the test set is the same as that of the training set, also produces comparable performace at a lower cost. This would only work if a full testing set is available and that its true label distribution is similar to that of the training set.

Instead of finding an optimal threshold value, the thresholding can be done by mapping the top m labels with the highest predicted score to 1 and the rest to zero (Yang, 2001). Again m can be found using a search strategy or alternatively setting it to the integer closest to the label cardinality in the training set. However, assuming that each observation has the same number

of labels is very unrealistic in most cases. An alternative could be to find the top m_k for $k = 1, 2, \ldots, K$ observations with the highest score for label λ_k and map them to 1 and the rest to zero. m_k can be found using any search strategy or possibly to match the label proportions of the training set.

For a more flexible approach when using threshold values, we can search for a threshold value for each label separately. In (Yang, 2001) the thresholds are sequentially tuned independently of one another using a validation set or cross-validation. For micro-averaged and example based metrics the tuning process requires more than one pass over all labels until convergence. This label dependent threshold strategy requires more computation and increases the risk of overfitting the thresholds to a subset of the data.

Furthermore, the threshold may even depend on the input or the predicted scores from the other labels (RamóN Quevedo *et al.*, 2012). This can conveniently be posed as a learning problem, where the threshold value or the number of labels for a given observation can be treated as the response variable and then using a statistical learning algorithm to predict these values. (Zhang *et al.*, 2006, and Nam *et al.* (2013)) used a regularised linear model to predict the thresholds for each label given $\boldsymbol{x}$. Other design choices can be to inlcude the predicted label scores as inputs or to pose it as a multiclass classification problem, modelling the number of labels (Ioannou *et al.*, 2010).

In Chapter 5 we will see that this learnable threshold strategy can easily be incorporated into a neural network, producing the class scores and optimal thresholds simultaneously, as in (Li *et al.*, 2017), with no need for an extra stage as described in the previous paragraph. We compare this with other more conventional thresholding strategies empirically in Chapter 6. In the next section we introduce neural networks and how they are adapted for MLC.

3.4 Artifical Neural Networks

An ANN maps inputs to outputs according to series of simple functions or transformations stacked on-top of each other. One of the main building blocks of an ANN is the very common linear transformation. For an easy introduction to ANNs we first discuss the linear model. Consider the linear model for the task of binary classification where the target is coded as zeros and ones and thus $f : \mathbb{R}^p \to \{0, 1\}$. The linear model assumes the the output, y, can be

obtained from a weighted average of the input $\boldsymbol{x}$ and thus we can write:

$$f(\boldsymbol{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p = w_0 + \boldsymbol{w}^\intercal \boldsymbol{x},$$

where w_j, $j = 1, 2, \ldots, p$, is the *weight* applied to the j-th feature, also referred to as *coefficients* in classical statistics, and w_0 a scalar added to the weighted combination, known as the *bias* term in machine learning or the *intercept* in statistics. For convenience we usually write

$$f(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x},$$

where we include the bias term in the weight vector $\boldsymbol{w}$ and add a constant feature to the inputs:

$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}.$$

The true values for $\boldsymbol{w}$ are unknown and therefore we need to estimate them from the data. The estimated weights are denoted as $\hat{\boldsymbol{w}}$ and the output obtained using the estimated weights (*i.e.* the predictions) is given by:

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{w}}^\intercal \boldsymbol{x}$$

Similar to the description in Section 1.1, we want the prediction $\hat{y}$ to be as close as possible to the true value y, measured by a loss function, L, and therefore choose $\hat{\boldsymbol{w}}$ as:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}^*} \sum_i L(y_i, \boldsymbol{w}^{*\intercal} \boldsymbol{x}),$$

which can be found using an optimisation algorithm, discussed shortly. A common loss function, typically used for regression but used here for illustration purposes, is the *squared error* loss:

$$L_{MSE}(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

The simplest form (and also the origin) of DNNs is a *feedforward neural network*, also known as the *multilayer perceptron* (MLP). They are called *feedforward* because information flows through the function being evaluated from the inputs $\boldsymbol{X}$, through the intermediate computations used to define f, and finally to the output $\boldsymbol{Y}$ (Goodfellow *et al.*, 2016). The *network* in the name refers to the structure of this type of model which is most naturally visualised as a network of inter-connected nodes.

3.4.1 Single Layer Perceptron

Like most other supervised learning models, a neural network is a mapping from an input to an output. The central idea of a neural network is to extract linear combinations of the inputs as derived features, and then model the target as a non-linear function of these features (Hastie *et al.*, 2009, Ch. 11). This idea was developed separately in the fields of statistics and artificial intelligence. In statistics, the first methods built on this idea was called the Projection Pursuit Regression (PPR) model (see Hastie *et al.*, 2009, pp. 389-392). This model can be written as

$$f(\boldsymbol{X}) = \sum_{m=1}^{M} g_m(\boldsymbol{\omega}_m^T \boldsymbol{X}),$$

where $\boldsymbol{X}$ is the usual input vector of p components and $\boldsymbol{\omega}_m$, $m = 1, \ldots, M$, p-sized vectors with unknown parameters. Thus, the PPR model is an additive model in the derived features, $V_m = \boldsymbol{\omega}_m^T \boldsymbol{X}$. $g_m(\cdot)$ is called a ridge function and is to be estimated. V_m is the projection of $\boldsymbol{X}$ onto the unit vector $\boldsymbol{\omega}_m$, and we seek $\boldsymbol{\omega}_m$ such that the model fits well, hence the name, Projection Pursuit. The details of this method is beyond the scope of this thesis and can be found at the reference above.

The term neural networks is used for a large class of models and learning methods. First, consider the "vanilla" neural network, known as the single layer perceptron. It is a neural network with a single hidden layer and trained by backpropogation. It can be applied to both regression and classification. It takes an input, $\boldsymbol{X} : 1 \times p$, transforms it to a hidden layer $\boldsymbol{Z} : 1 \times M$ and then uses $\boldsymbol{Z}$ as input to model the target, $\boldsymbol{Y} : 1 \times K$. This structure can be represented as a network as shown in Figure 3.4.

The number of units in the final layer matches the dimensionality of the output, denoted by K. Thus for classic regression, $K = 1$, and for multiclass classification, K is the number of possible categories, where unit k, $k = 1, \ldots, K$, represents the score for class k. For this discussion we will describe neural networks for multiclass classification. Thus there are K target measurements, $\boldsymbol{Y} = \{Y_1, Y_2, \ldots, Y_K\}$. Y_k is coded as 1 when class k is present and as 0 otherwise.

The hidden layer units, $\boldsymbol{Z} = \{Z_1, Z_2, \ldots, Z_M\}$, are a set of features derived from the input. They are created by first taking a linear combination of the inputs and then sending it through a non-linear *activation function*, $a(\cdot)$,

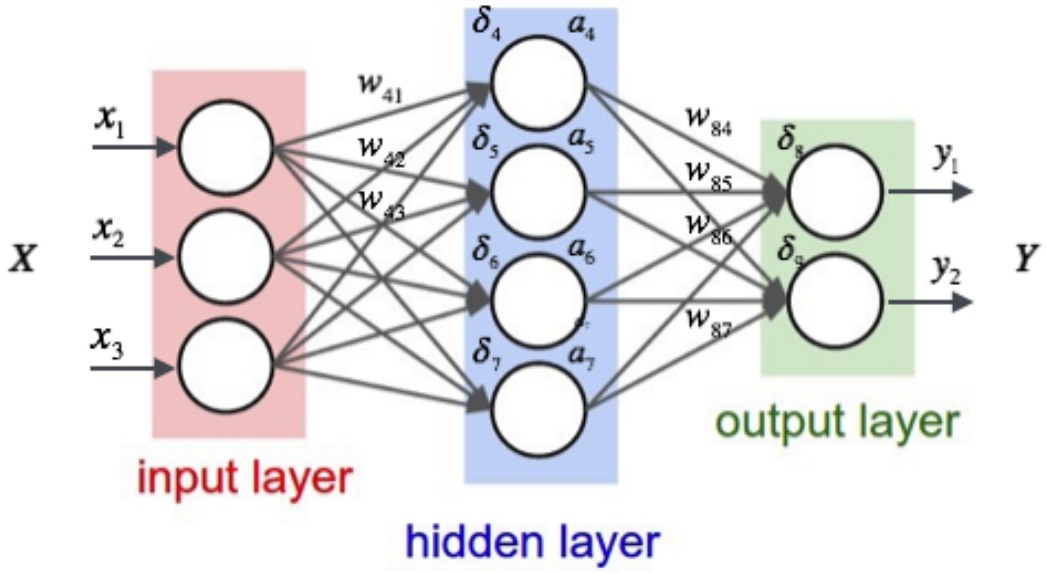

Figure 3.4: Graph structure of a vanilla neural network.

$$Z_m = a\left(\alpha_{0m} + \boldsymbol{\alpha}_m^T \boldsymbol{X}\right),$$

for $m = 1, \ldots, M$. α_{0m} and $\boldsymbol{\alpha}_m$ are the coefficients of the linear mapping. Note that a layer that outputs a linear transformation of its inputs in this fashion is also called a *fully-connected* or *dense* layer. The activation function, $a(\cdot)$, was usually chosen to be the sigmoid function, $a(v) = \frac{1}{1+e^{-v}}$. However these days, there are many, more effective activation functions used in deep neural networks which we discuss in Section 3.4.2.

The output units of the neural network can then be expressed as

$$f_k(\boldsymbol{X}) = g_k\left(\beta_{0k} + \boldsymbol{\beta}_k^T \boldsymbol{Z}\right),$$

for $k = 1, \ldots, K$. Here, the β's are the coefficients of the linear combination of the derived features, $\boldsymbol{Z}$, and $g_k(\cdot)$ is another activation function. Originally, for both regression and classification, $g_k(\cdot)$ was chosen to be the identity function, but they later found that the softmax function was better suited for multiclass classification, defined as

$$g_k(\boldsymbol{T}) = \frac{e^{T_k}}{\sum_k e^{T_k}}.$$

This function is exactly the transformation used in the multilogit model discussed in **??**. It produces output in the range [0,1], summing to 1, similar to

the properties of conditional class probabilities.

The units in $\boldsymbol{Z}$ are called hidden since they are not directly observed. The aim of this transformation is to derive features, $\boldsymbol{Z}$, so that the classes become linearly separable in the derived feature space (Lecun *et al.*, 2015). Many more of these hidden layers (combination of linear and non-linear transformations) can be used to derive features to input into the final classifier. This is what we refer to as deep neural networks (DNNs) or deep learning methods.

Note, that if the $a(\cdot)$ activation function was the identity function or another linear function, the whole network would collapse into a single linear mapping from inputs to outputs. By introducing the non-linear activations, it greatly enlarges the class of functions that can be approximated by the network (see universal approximator).

In a statistical learning sense, the hidden units can be thought of as a basis function expansion of the original inputs. The neural networks is then a standard linear (multilogit) model with the basis expansions as inputs. The only difference to the conventional basis function expansion techinique in Statistical Learning (Hastie *et al.*, 2009, Ch. 5) is that the parameters of the basis functions are learned from the data.

One can now also see the relationship between a neural network and the PPR model. If the neural network has one hidden layer, it can be written in the exact same form as the PPR model. The difference is that the PPR uses a nonparametric function $g_m(v)$, while the neural network uses far simpler non-linear activation functions, like $a(\cdot)$.

The number of units in the hidden layer, M, is also a value to be decided on. Too few units will not allow the network enough flexibility to model complex relationships and too many takes longer to train and increases the chance of overfitting. M is mostly chosen by experimentation. A good starting point would be to choose a large value and training the network with regularisation (discussed shortly).

The difference between the above discussed neural networks and current state-of-the-art deep learning methods, is the number and type of hidden layers. The following section discusse the popular activation functions used in DNNs.

3.4.2 Activation Functions

In the previous section, we introduced activation functions, which are simple non-linear functions of its input. These are usually applied after a fully connected layer (linear transformation) and are crucial for the flexibility of a deep neural network. We also mentioned that the sigmoid activation, which was originally the go-to activation, is currently not the most popular choice. Another activation function originally thought to work well was, $a(x) = \tanh(x)$. However, by far the most common activation function used at the time of writing is the Rectified Linear Units (ReLU) non-linearity. Its definition is much simpler than its name and is defined as $a(x) = \max(0, x)$. It was introduced in (Krizhevsky *et al.*, 2012) and they showed that using ReLUs in their CNNs reduced the number of training iterations to reach the same point by a factor of 6 compared to using $\tan(x)$.

There are a plethora of proposals for activation functions, since any simple non-linear (differentiable?) function can be used. Some of the recent most popular choices are exponential linear units (ELUs) (Clevert *et al.*, 2015) and scaled exponential linear units (SELUs) (Klambauer *et al.*, 2017). The choice of activation function usually influences the convergence time and some might protect the training procedure from overfitting in some cases. The different activation functions can be experimented with, however it would be sufficient in most cases to use ReLUs. The other mentioned proposals have inconsistent gains over ReLUs and therefore it remains the standard choice.

However, very recently (Ramachandran *et al.*, 2017) used automated search techniques to discover novel activation functions. The exhaustive and reinforcement learning based searched identified a few promising novel activation functions on which the authors then did further empirical evaluations. They found that the so-called *Swish* activation function,

$$a(x) = x \cdot \sigma(\beta x),$$

where β is a constant (can also be a trainable parameter), gave the best empirical results. It consistently matched or outperformed ReLU's on deep networks applied to the domains of image classification and machine translation.

3.5 Training a Neural Network

3.5.1 Backpropogation

In **??** we discussed how to fit a linear model using the Stochastics Gradient Descent optimisation procedure. Currenlty, SGD is the most effective way of training deep networks. To recap, SGD optimises the parameters θ of a networks to minimise the loss,

$$\theta = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} l(\boldsymbol{x}_i, \theta).$$

With SGD the training proceeds in steps and at each step we consider a mini-batch of size $n \leq N$ training samples. The mini-batch is used to approximate the gradient of the loss function with respect to the paramaters by computing,

$$\frac{1}{n} \frac{\partial l(\boldsymbol{x}_i, \theta)}{\partial \theta}.$$

Using a mini-batch of samples instead of one at a time produces a better estimate of the gradient over the full training set and it is computationally much more efficient.

This section discusses the same procedure, but applied to a simple single hidden layer neural network. This is made possible by the *backpropogation* algorithm. Note, this process extends naturally to the training of deeper networks.

The neural network described in the previous section has a set of unknown adjustable weights that defines the input-output function of the network. They are the $\alpha_{0m}, \boldsymbol{\alpha}_m$ paramters of the linear function of the inputs, $\boldsymbol{X}$, and the $\beta_{0k}, \boldsymbol{\beta}_k$ paramaters of the linear transformation of the derived features, $\boldsymbol{Z}$. Denote the complete set of parameters by θ. Then the objective function for regression can be chosen as the sum-of-squared-errors:

$$L(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(\boldsymbol{x}_i))^2$$

and for classification, the cross-entropy:

$$L(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(\boldsymbol{x}_i),$$

with corresponding classifier $G(\boldsymbol{x}) = \arg\max_k f_k(\boldsymbol{x})$. Since the neural network for classification is a linear logistic regression model in the hidden units, the paramaters can be estimated by maximum likelihood. (I'm not sure if this is possible with deeper networks, and with the non-linear activations?). According to Hastie *et al.* (2009, p. 395), the global minimiser of $L(\theta)$ is most likely an overfit solution and we instead require regularisation techniques when minimising $L(\theta)$.

Therefore (?), one rather uses gradient descent and backpropogation to minimise $L(\theta)$. This is possible because of the modular nature of a neural network, allowing the gradients to be derived by iterative application of the chain rule for differentiation. This is done by a forward and backward sweep over the network, keeping track only of quantities local to each unit.

In detail, the backpropogation algorithm for the sum-of-squared error objective function,

$$\begin{aligned} L(\theta) &= \sum_{i=1}^{N} L_i \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(\boldsymbol{x}_i))^2, \end{aligned}$$

is as follows. The relevant derivatives for the algortihm are:

$$\begin{aligned} \frac{\partial L_i}{\partial \beta_{km}} &= -2(y_{ik} - f_k(\boldsymbol{x}_i)) g_k'(\boldsymbol{\beta}_k^T \boldsymbol{z}_i) z_{mi}, \\ \frac{\partial L_i}{\partial \alpha_{ml}} &= -\sum_{k=1}^{K} 2(y_{ik} - f_k(\boldsymbol{x}_i)) g_k'(\boldsymbol{\beta}_k^T \boldsymbol{z}_i) \beta_{km} \sigma'(\boldsymbol{\alpha}_m^T \boldsymbol{x}_i) x_{il}. \end{aligned}$$

Given these derivatives, a gradient descent update at the $(r+1)$-th iteration has the form,

$$\begin{aligned} \beta_{km}^{(r+1)} &= \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial L_i}{\partial \beta_{km}^{(r)}}, \\ \alpha_{ml}^{(r+1)} &= \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial L_i}{\partial \alpha_{ml}^{(r)}}, \end{aligned}$$

where γ_r is called the learning rate. Now write the gradients as

$$\begin{aligned} \frac{\partial L_i}{\partial \beta_{km}} &= \delta_{ki} z_{mi}, \\ \frac{\partial L_i}{\partial \alpha_{ml}} &= s_{mi} x_{il}. \end{aligned}$$

The quantities, δ_{ki} and s_{mi} are errors from the current model at the output and hidden layer units respectively. From their definitions, they satify the following,

$$s_{mi} = \sigma'(\boldsymbol{\alpha}_m^T \boldsymbol{x}_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki},$$

which is known as the backpropogation equations. Using this, the weight updates can be made with an algortihm consisting of a forward and a backward pass over the network. In the forward pass, the current weights are fixed and the predicted values $\hat{f}_k(\boldsymbol{x}_i)$ are computed. In the backward pass, the errors δ_{ki} are computed, and then backpropogated via the backpropogation equations to give obtain s_{mi}. These are then used to update the weights.

Backpropogation is simple and its local nature (each hidden unit passes only information to and from its connected units) allows it to be implented efficiently in parallel. The other advantage is that the computation of the gradient can be done on a batch (subset of the training set) of observations. This allows the network to be trained on very large datasets. One sweep of the batch learning through the entire training set is known as an epoch. It can take many training epochs for the objective function to converge.

3.5.2 Learning Rate

The convergence times also depends on the learning rate, γ_r. There are no easy ways for determining γ_r. A small learning rate slows downs the training time, but is safer against overfitting and overshooting the optimal solution. With a large learning rate, convergence will be reached quicker, but the optimal solution may not have been found. One could do a line search of a range of possible values, but this usually takes too long for bigger networks. One possible strategy for effective training is to decrease the learning rate every time after a certain amount of iterations.

Recently, in (https://arxiv.org/abs/1711.00489) (no bibtex entry), the authors found that, instead of learning rate decay, one can alternatively increase the batch size during training. They found that this method reaches equivalent test acccuracies compared to learning rate decay after the same amount of epochs. But their method requires fewer parameter updates.

3.5.3 Basic Regularisation

There are many ways to prevent overfitting in deep neural networks. The simplest strategies for single hidden layer networks are by early stopping and weight decay. Stopping the training process early can prevent overfitting. When to stop can be determined by a validation set approach. Weight decay is the addition of a penalty term, $\lambda J(\theta)$, to the objective function, where,

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2.$$

This is exactly what is done in ridge regression (Hastie *et al.*, 2009, Ch. 4). $\lambda \geq 0$ and larger values of λ tends to shrink the weights towards zero. This helps with the generalisation ability of a neural network, but recently more effective techniques to combat overfitting in DNNs have been developed. These are dicussed in Section 4.3.

It is common to standardise all inputs to have mean zero and standard deviation of one. This ensures that all input features are treated equally. Now we have covered all of the basics for simple (1-layer) neural networks.

3.5.4 Optimisation

Chapter 4

Convolutional Neural Networks

4.1 Introduction

Representation learning is a set of methods that can take raw unstructured data as input and automatically learn the optimal representations from the data for the specific task, *e.g.* classification. Deep learning methods are representation learning methods, explaining their superiority in image classification and related tasks. The 'deep' of deep learning refers to the multiple layers of a deep learning network stacked on top of each other. Each layer transforms a representation at one level (starting at the input) to a slightly higher level of abstraction, until a level is reached sufficient for classification (or any other task). These layers are a combination of simple linear and non-linear functions and together (if the network is deep enough) it can approximate any function, no matter how complex (Hornik, 1991).

For many years, dating back to the late 1950s, researches have tried to find ways to replace hand-crafted feautres with multilayer networks (Selfridge, 1959; Rosenblatt, 1957). The first real progress was made when they found that the networks can be trained by simple gradient descent and the backpropogation algorithm (Rumelhart *et al.*, 1988). Until the early 2000s, research communities related to statistics and artificial intelligence did not have much hope for neural networks. They believed training the network by gradient descent will result in solutions stuck in a poor local minima. In practice this is not true and it has actually been shown that the solutions tend to get stuck in saddle points, which are not that problematic (Dauphin *et al.*, 2014; Choromanska *et al.*, 2014).

Hope was restored when unsupervised methods were developed to pretrain

networks on unlabelled data to obtain a weight initialisation for the supervised learning training process (Hinton *et al.*, 2006; Bengio *et al.*, 2006). This helped the backpropogation algorithm to find good solutions especially when the number of labelled data points were limited. More efficient ways of training the networks were developed by making use of GPUs, decreasing the average training time of networks by at least a factor of 10. Finally, a general consensus on the power of deep learning methods were reached when a CNN was trained on a large-scale image classification data set to beat previous state-of-the-art by a large margin.

This chapter discusses deep neural networks (DNNs) focussing on image classification. Convolutional Neural Networks (CNNs), a specific type of DNN, is the state-of-the-art model in single label image classification. Therefore the aim of this chapter is to gain an understanding of CNNs such that we can later extend it to handle multi-label image classification. First, **??** introduces Neural Networks. It looks at its structure and the various strategies regarding its optimisation. **??** is on CNNs, which are especially useful for image classification. The section focusses on the unique components of a CNN and why it works so well. Then in **??** we will briefly discuss Recurrent Neural Networks (RNNs). These type of networks are especially well suited for sequential input, but they have also been used for multi-label classification and therefore it is included in this chapter. Section 4.3 discusses the important problem of reducing overfitting of DNNs. It reviews the most important strategies for improving the generalisation ability of DNNs. Section 4.4 on transfer learning. Maybe also something on attention and then a conclusion.

As mentioned before, Deep Neural Networks are extensions of Neural Networks. The extensions consist of adding more hidden layers and the use of more advanced layers. The type of DNN best suited for image classification is called a Convolutional Neural Network (CNN). The identifying feature of a CNN is its convolutional layer.

4.2 Convolutional Layer

4.3 Reducing Overfitting

The relationship between the input and the true output in an image classification problem is usually complicated. CNNs generally have millions of trainable

paramaters and therefore there will typically be many different settings of these parameters that allow the model to fit the training data almost perfectly, especially if the amount of training data is limited. However, a network with weights tuned to fit the training data perfectly is very likely to perform much worse on test data not used for training, since the weights are specifically suited for the examples in the training set. This is what we call overfitting.

The bigger the network the more prone it becomes to overfitting. Luckily there are several ways to combat overfitting. Some of the more important strategies are introduced here.

4.3.1 Data Augmentation

The simplest way to reduce overfitting is to get more labelled data. But in many cases this is not possible for several reasons including time and budget constraints. The next best approach is to artificially enlarge the dataset using label-preserving transformations. This is called data augmentation (Krizhevsky *et al.*, 2012) and can naturally be applied to image classification datasets.

There are many possible transformations (or augmentations) that can be applied to images including: rotating, mirroring, cropping, zooming, *etc.* A combination of these transformations can be performed randomly on images each time its shown to the network when training. Therefore every time a different version of the same image is shown to the network, which has a similar effect to showing it a new labelled image.

poorly written. give more resources

4.3.2 Dropout

Overfitting can be reduced by using dropout (Hinton *et al.*, 2012) to prevent complex co-adaptions on the training data. Dropout consists of setting the output of a hidden unit to zero with a probability p (in the original paper they used $p = 0.5$). The units which are set to zero do not contribute to the forward pass and do not participate in backpropogation. Every time an input is presented, the neural network samples a different set of units to be dropped out.

This technique ensures that a unit does not rely on the presence of a particular set of other units. It is therefore forced to learn more robust features

that are useful in conjunction with many different random subsets of the other units (Krizhevsky *et al.*, 2012).

At test time, no units are dropped out and their output is multiplied by $1-p$ (make sure) to compensate for the fact that all of the units are now active. Dropout does tremendously well to combat overfitting, but it slows down the covergence time of training.

- in the original paper they also compare the technique to ensebmling

4.3.3 Batch Normalisation

One of the things that complicate the training of neural networks is the fact that hidden layers have to adapt to the continuously changing distribution of its inputs. The inputs to each layer are affected by the paramaters of all its preceding layers and a small change in a preceding layer can lead to a much bigger difference in output as the network becomes deeper. When the input distribution to a learning system changes, it is said to experience covariate shift (Shimodaira, 2000).

Using ReLUs, carefull weight initialisation and small learning rates can help a network to deal with the internal covariate shift. However, a more effective way would be to ensure that the distribution of non-linearity inputs remains more stable while training the network. (Ioffe and Szegedy, 2015) proposed *batch normalisation* to do just that.

A batch normalisation layer normalises its inputs to a fixed mean and variance (similar to how the inputs of the network is normalised) and therefore it can be applied before any hidden layer in a network to prevent internal covariate shift. The addition of this layer dramatically accelerates the training of DNNs, also because it can be used with higher learning rates. It also helps with regularisation (Ioffe and Szegedy, 2015), therefore in some cases dropout is not necessary.

The batch normalising transform over a batch of univariate inputs, $x_1, \ldots, x_n$ is done by the following steps:

1. Calculate the mini-batch mean, μ, and variance, σ^2:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$
$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - u)^2$$

2. Normalise the inputs,

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}},$$

where ϵ is a constant to ensure numerical stability.

3. Scale and shift the values,

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

where γ and β are the only two learnable paramaters of a batch normalisation layer.

The reason for the scale and shift step is to allow the layer to represent the identity transform if the normalised inputs are not suitable for the following layer, *i.e.* the scale and shift step will reverse the normalisation step if $\gamma = \sqrt{\sigma^2 + \epsilon}$ and $\beta = \mu$.

4.4 Transfer Learning

The major critique against DNNs are that they require a huge amount of training data and that they take extremely long to train. This is somewhat true, however, *transfer learning* provides an effective solution to these problems. Recall that DNNs are examples of representation learning algorithms. Consider the case where a CNN was sucessfully trained on ImageNet. For any input image, each layer of the CNN produces some feature representation of the input image. (Not sure where Zeiler paper is going to be discussed).

Chapter 5

Convolutional Neural Networks for Multi-Label Image Classification

One potential concern with this approach is the risk of learning biased interde-pendencies from a limited training set which does not accurately represent a realistic distribution of pathologies – if every example of cardiomegaly is also one of cardiac failure, the model may learn to depend too much on the presence of other patterns such as edemas which do not always accompany enlarge-ent of the cardiac silhouette ((Yao *et al.*, 2017) on modeling label correlations).

5.1 Baseline

5.2 From Single-Label to Multi-Label CNNs

It is very simple to adapt a DNN to suit a MLC problem. Recall that in a multiclass classification problem we would use a softmax layer activation on the final layer of a DNN. The softmax transforms the output so that the class scores are squeezed into the range of 0-1 and so that the class scores sum to 1. This is done so that the class scores better imitate class probabilities. However, when an observation can be associated with more than one label, we wouldn't expect the class probabilities to sum to 1. Therefore, a sigmoid activation is used on the output layer of a DNN for MLC. The sigmoid squeezes the class scores in the range of 0-1 without the constraint that the values should sum

to 1. See Figure @ref(fig:sigsof) for an illustration of the differences between sigmoid and softmax.

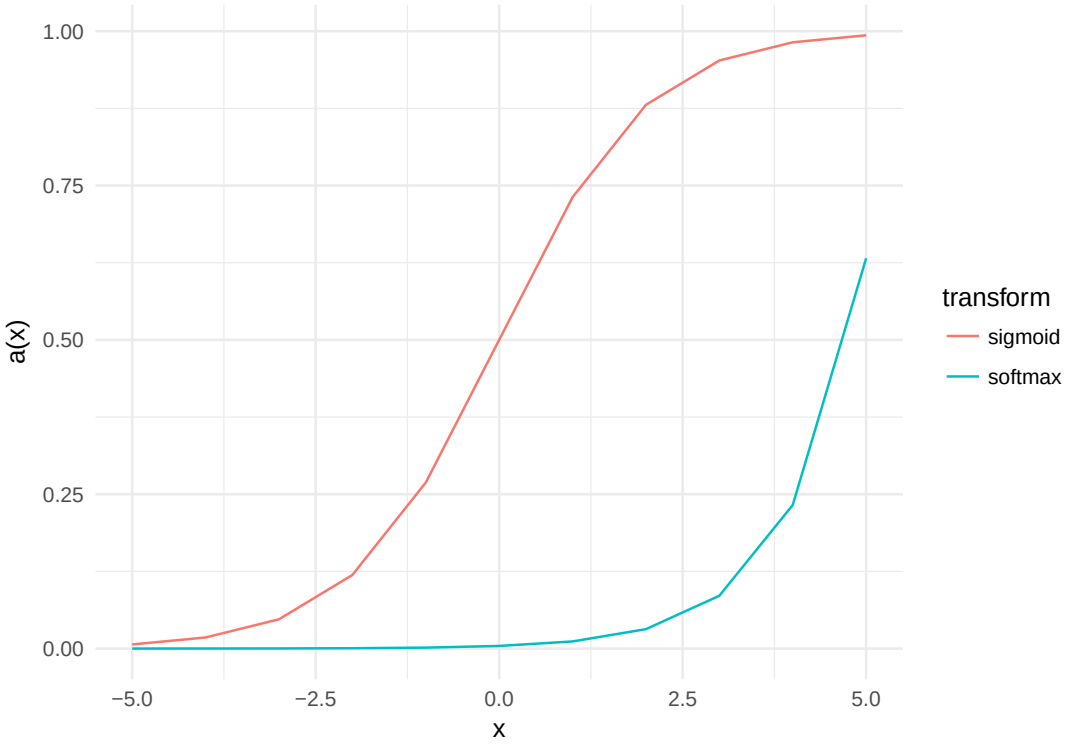

Figure 5.1: Difference between the sigmoid and softmax transformation

It turns out that a DNN with a sigmoid output layer is a very good baseline model for MLIC. It is only necessary to train one network to predict all labels and the network implicitly takes label correlations into account since the weights were trained to optimise all labels. For a MLIC problem, one can use a pretrained single-label classification network, say on ImageNet, swap out its softmax layer with a sigmoid layer and then fine-tune the network (with binary cross-entropy) on your specific data and achieve very good results. Although this method may be sufficient in some cases, there are ways to improve on this architecture. They are discussed below, sorted in order of publishing date. Remember, we are only considering end-to-end network approaches.

5.3 Multi-Label Loss functions

One of the main considerations when switching from single-label to multi-label CNNs is the choice of loss function to minimise, *i.e.* how the network penalises

the deviation between the predicted and true labels. Recall that when training a CNN, $f(x, \theta) \in \mathbb{R}^K$, we solve the optimisation problem

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} l(f(x_i, \theta), y_i) + \mathcal{R}(\theta),$$

where $l(f(x_i, \theta), y_i)$ is the loss for observation i and $\mathcal{R}(\theta)$ is a regularisation term. The loss function needs to suit the multi-label output. We divide the loss functions used for training multi-label CNNs into two classes: cross-entropy based and ranking based.

5.3.1 Cross-Entropy Based

The simplest and standard choice is to use the *binary cross-entropy loss*. This is simple the binary cross-entropy calculated for each label separately and then summed over all labels. Suppose the output from the network given the i-th input is the K-dimensional vector $\boldsymbol{p}_i = (p_{i1}, p_{i2}, \ldots, p_{iK})$, each element corresponding to the network's confidence that a certain label is associated with the input image. Assume that the final layer of the network is a sigmoid activation and therefore $p_{ij} \in (0, 1)$. Let the true labels for that image be represented by the K-dimensional vector $\boldsymbol{y}_i = (y_{i1}, y_{i2}, \ldots, y_{iK})$ where $y_{ij} = 1$ if label j is associate with image i and $y_{ij} = 0$ if not. Then the cross-entropy for binary output j can be given as

$$\mathrm{CE}(p_{ij}, y_{ij}) = -\left(y_{ij} \log(p_{ij}) + (1 - y_{ij}) \log(1 - p_{ij})\right),$$

which can also be given by

$$\mathrm{CE}(p_{ij}, y_{ij}) = \begin{cases} -\log(p_{ij}) & \text{if } y_{ij} = 1 \\ -\log(1 - p_{ij}) & \text{if } y_{ij} = 0. \end{cases}$$

Thus the cross-entropy penalises p_{ij} close to zero when $y_{ij} = 1$ (label j is present) and p_{ij} near 1 when $y_{ij} = 0$. The loss for a single observation can then be given by:

$$l_{CE}(\boldsymbol{p_i}, \boldsymbol{y_i}) = \sum_{j=1}^{K} \mathrm{CE}(p_{ij}, y_{ij})$$

This is what is referred to when reading "trained with binary cross-entropy", for example in (Rajpurkar *et al.*, 2017).

The authors of (Wang *et al.*, 2017) found that their network had trouble learning positive labels *i.e.* the network had a low recall. They argued that it was because of the small proportion of positive labels per image. To counter this imbalance, they proposed a balancing factor to give more weight to missclassified positive labels. Let β_i be the proportion of positive labels for image i, *i.e* $\beta_i = \frac{\sum_{j=1}^{K} y_{ij}}{K}$. They then reweighted the cross-entropy loss as follows:

$$\text{W-CE}(p_{ij}, y_{ij}) = \begin{cases} -\frac{1}{\beta_i} \log(p_{ij}) & \text{if } y_{ij} = 1 \\ -\frac{1}{1-\beta_i} \log(1 - p_{ij}) & \text{if } y_{ij} = 0 \end{cases}$$

This formulation gives a larger penalty to a missclassified positive label if the proportion of positive labels for the corresponding output is small. For example, if an input image is labeled with only 1 out 10 possible labels, the loss contributed if it was missclassified will be scaled by a factor of 10 (from $\frac{1}{0.1}$), whereas the loss contributed by the missclassified negative labels will only be scaled by a factor of 1.1111 (from $\frac{1}{1-0.1}$). The reverse would be true if for example 9 out the 10 possible labels are associated with an image. In (Wang *et al.*, 2017) they found that the W-CE loss gave the best results in terms of the area under the ROC curve (AUC) for each label.

Of course there are other ways of defining the balancing factor. Another common convention to deal with class imbalance is to weight the contribution made by each label separately according to their proportions of positive instances, instead of a per observation weighting as was done previously.

Although these weighting schemes can give rare classes greater weight in the loss calculcation, it cannot differentiate between easy and hard examples, *i.e.* the weights are independent of how wrong or right the network is. We refer to the correctly predicted labels with high confidence as easy examples and those incorrectly predicted with high confidence as hard examples.

Observe the negative log function of values between 0 and 1 in Figure 5.2. Notice that for values greater than 0.5 the negative log function still produces a relatively large non-zero value. This means that correctly classified examples ($p > 0.5$) will still make a significant contribution to the overall loss incurred. We actually want a curve that is relatively flatter for values greater than roughly 0.6 and steeper for values less than say 0.4. This will ensure that the loss focusses more on the so-called hard examples.

One such loss function is proposed in the field of object detection, called the *Focal loss* (Lin *et al.*, 2017). The focal loss is defined as

$$\mathrm{FL}(p_{ij}, y_{ij}) = \begin{cases} -(1-p_{ij})^{\gamma}\log(p_{ij}) & \text{if } y_{ij} = 1 \\ -p_{ij}^{\gamma}\log(1-p_{ij}) & \text{if } y_{ij} = 0, \end{cases}$$

where γ is tunable focussing parameter $\gamma \geq 0$. This can be regarded as another form of the weighted cross-entropy, only this time, the weight is dependent on the confidence the network has in each label, p_{ij}. Consider the case where image i is tagged with label j ($y_{ij} = 1$). If the network incorrecly has a low confidence that label j is relevant (p_{ij} close to zero), the scaling factor of the cross-entropy will be close to 1 and will have virtually no reduction in the cross-entropy. However, as the label confidence grows and the network becomes more certain that label j is relevant to image i ($p_{ij} \to 1$), the scaling factor reduces the cross-entropy by a greater amount. Therefore, when $p_{ij} > 0.6$, the loss contribution will have a relatively much smaller contribution to the overall loss. A similar interpretation can be given for the case when $y_{ij} = 0$.

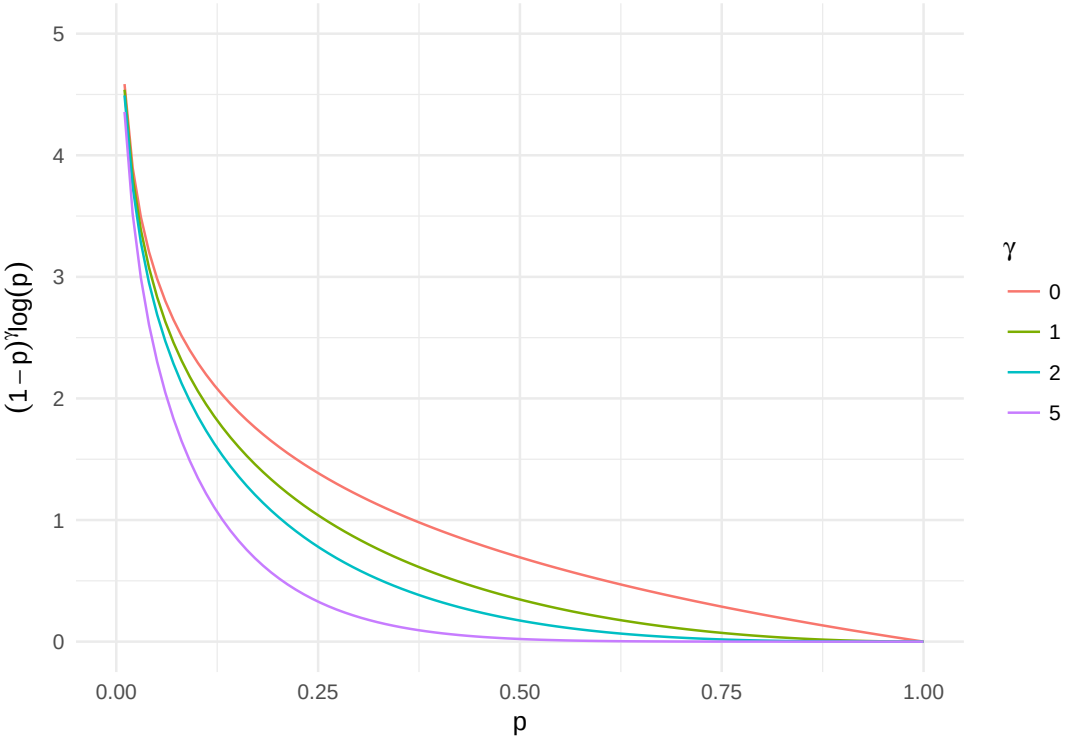

Figure 5.2: The loss contributions made by the focal loss.

Figure 5.2 shows how the focal loss has a flatter penalty for correctly classified labels with high confidence. The effect of this down weighting is controlled by γ. If $\gamma = 0$ the focal loss equivalent to the cross-entropy loss. The greater γ the greater the down weighting. The creators of the focal loss found that $\gamma = 2$ gave satifying results but that the focal loss is not that sensitive to this choice. The metric of interest was the average precision.

To the best of our knowledge, the focal loss has never before been used for multi-label classification, only for object detection. We report some of our findings in the next chapter along with experiments using a combination of the weighted cross-entropy and the focal loss.

5.3.2 Ranking Based

While in multi-label image classification we care most about correctly classifying positive labels, it is equally important for the classifier to make sensible mistaktes, *i.e.* even if it fails to classify any positive labels it should still give higher scores to the positive labels compared to the negative labels. Thus it is desired for $p_{iu} > p_{iv}$, $\forall u \in L_i$, $v \notin Y_i$, where Y_i is the labelset associated with the i-th image. To help ensure this we can choose to train the CNN using a rank based loss function. These may also act as better surrogates for some rank based multi-label evaluation metrics like AUC.

The first use of a ranking based loss in the context of multi-label image classification using CNNs can be found in (Gong *et al.*, 2013), which is also the first published work on multi-label image classification with CNNs. The *pairwise-ranking loss* can be given as

$$l_{rank}(\boldsymbol{p_i}, \boldsymbol{y_i}) = \sum_{v \notin Y_i} \sum_{u \in Y_i} \max(0, 1 - p_{iu} + p_{iv}))$$

Thus a non-zero loss will be contributed by a positve label, p_{iu}, and negative label, p_{iv}, if $p_{iu} - p_{iv} < 1$. This condition will always be true if the final layer of the newtork is a sigmoid activation. Then we would want p_{iu} to be as close as possible to 1 and p_{iv} as close as possible to zero. This is calculated for each positive and negative label pair of an image and for each image in a batch to determine the final pairwise rank loss. However, here it is not necessary for sigmoid activation at the output. It is also possible to swap the constant 1 with any other constant to change the margin.

The authors of (Gong *et al.*, 2013) found that l_{rank} does not optimise top-k accuracy and therefore proposed the use of the *Weighted Approximate Ranking* (WARP) loss, defined as:

$$l_{WARP}(\boldsymbol{p_i}, \boldsymbol{y_i}) = \sum_{v \notin Y_i} \sum_{u \in Y_i} w(r_i^u) \max(0, 1 - p_{iu} + p_{iv}))$$

where $w(\cdot)$ is a weighting function and r_i^u the estimated rank for positive label u. They used the following weighting function:

$$w(r) = \sum_{j=1}^{r} \frac{1}{j}$$

The idea is that the loss contribution for a label pair should have a greater weight if the positive label is not ranked near the top of the label list, *i.e.* when r is large. r_i^u is an estimation of the label u's rank for image i. This estimation is based on the output of the network, $\boldsymbol{p_i}$. (Gong *et al.*, 2013) used a sampling approach to estimate the rank, based on the number of trials it took to sample a negative label given a positive label for which $p_{iu} - p_{iv} < 1$. This adds extra computation to the loss calculation and gets very expensive as the number of labels become larger. In addition, the hinge function used inside the ranking loss is not smooth everywhere and is therefore difficult to optimise.

Recently, (Li *et al.*, 2017) proposed a smooth approximation of l_{rank}, which also does not require a rank estimation stage. This approximation uses the log-sum-exp pairwise function and therefore they called it the LSEP loss:

$$l_{LSEP}(\boldsymbol{p_i}, \boldsymbol{y_i}) = \log \left(1 + \sum_{v \notin Y_i} \sum_{u \in Y_i} \exp(p_{iv} - p_{iu}) \right)$$

See Figure 5.3 how the contribution made by each pair of labels differ between the pairwise ranking loss and the LSEP loss. Note, that $\exp(p_{iv} - p_{iu})$ will give a much higher loss for large values of $p_{iv} - p_{iu}$ compared to $\max(1 - p_{iu} + p_{iv})$. However, this difference is reduced by the log function in l_{LSEP}.

The authors of (Li *et al.*, 2017) argue that it is not necessary to add a weighting mechanism as in l_{WARP} since weighting is done implicity by l_{LSEP}. l_{LSEP} is also quite similar to the *BP-MLL* loss proposed in (Nam *et al.*, 2013) for text classification and genomics:

$$l_{BP-MLL}(\boldsymbol{p_i}, \boldsymbol{y_i}) = \sum_{v \notin Y_i} \sum_{u \in Y_i} \exp(p_{iv} - p_{iu})$$

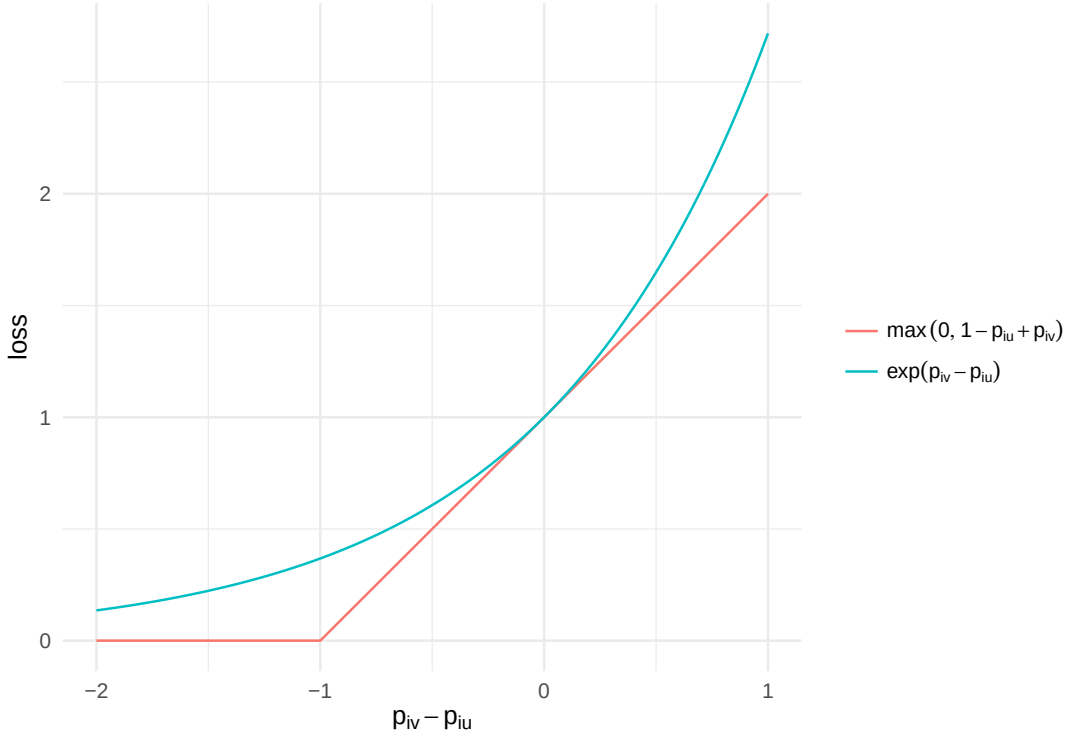

Figure 5.3: The hinge function compared to the log-sum-exp function for different margins.

According to (Li *et al.*, 2017) l_{LSEP} is numerically more stable and focuses more on the violating cases ($p_{iv} > p_{iu}$), whereas l_{BP-MLL} keeps pushing $p_{iu} - p_{iv}$ to ∞, because it lacks the $\log(1 + \text{pairwise-loss})$ form of l_{LSEP}.

Computing the loss contributed by each pair of labels can become computationaly infeasible if K is large. Therefore (Li *et al.*, 2017) also introduced a sampling trick to let the loss function scale linearly with increase in K. Instead of calculating the loss for each positive-negative label pair, they only sample a maximum of t pairs from the Cartesian product, where they set $t = 1000$.

In their results they found that l_{LSEP} performed better than l_{rank}, l_{WARP} and l_{BP-MLL} in terms of the macro F_1-score and exact match measures, evaluated on 3 multi-label image benchmark datasets. It will be interesting to see how it compared to the cross-entopy based loss functions. We will investigate this in the next chapter.

5.4 Modelling Label Correlations

As mentioned in **??** exploiting label correlations is a key challenge in multi-label classificaiton. The standard way of adapting a CNN for multi-label classification does not explicitly model label correlation, however we can think of it as implicitly dealing with it.

- chaining
- context gating
- SE-Net
- mention RNN: (https://nurture.ai/nips-challenge/p/c074cc00-3f61-476b-8f0c-83795f0229ed)

The winning solutions to the YouTube video classification challenge proposed interesting and ways to model label dependence for a MLC task. I identified two approaches worth exploring to see how well it works on MLIC.

The first approach is called **context gating** (CG) by (Miech *et al.*, 2017). The idea of the CG module is to capture nonn-linear interedependencies between labels as well as among features. It does this by transforming an input representation X into a new representation Y, by

$$Y = \sigma(WX + b) \circ X,$$

where $\circ$ is the elementwise multiplication operation. $WX + b$ is the linear transformation of X and thus W and b are arrays of learnable parameter. The sigmoid activation, $\sigma(.)$, transforms the linear transform of X to values between 0 and 1 and thus act as weights (or gates) which are then multplied with X.

The motivation behind this is to introduce non-linear interactions among activations of the input representation and to recalibrate the strengths of different activations of the input representation through a self-gating mechanism. The authors used CG first to transform an intermediate feature vector before passing it to the classification module. Secondly, they applied it after the classification layer to capture the prior structure of the output label space.

By sending a feature vector through the context gate, dependencies among features can be captured. For example, the context gating unit can learn to suppress features likely to be in the background and emphasise the foreground objects. For instance, if features corresponding to 'Trees', 'Skier' and 'Snow' have high co-occurring activations in a skiing video, context gating could learn

to suppress the background features such as 'Trees' and 'Snow', which are less important for the classification. By applying the CG unit after the initial classification layer it can learn to downweight unlikely combinations of labels. For example we saw previously that 'Male' and 'Skirt' does not appear often in the WIDER-attribute images. If this was predicted by the initial classificaiton layer, CG would then be able to tweak these predictions.

The authors saw a significant performance increase in their model after incorporating CG.

The other approach from the YouTube challenge is called **chaining** by Wang *et al.* (2017). The idea is based on CC to model label correlations (although I think it resembles BR+ and stacked BRs more). A chaining unit accepts one feature vector and multiple model predictions as input and the produces a new model prediction. The model predictions are embedded into a lower dimensional space if K is large. These units can be stacked on top of each other and then at each step the model can learn from its previous mistakes. See Figure @ref(fig:chaining) for an illustration of the chaining approach.

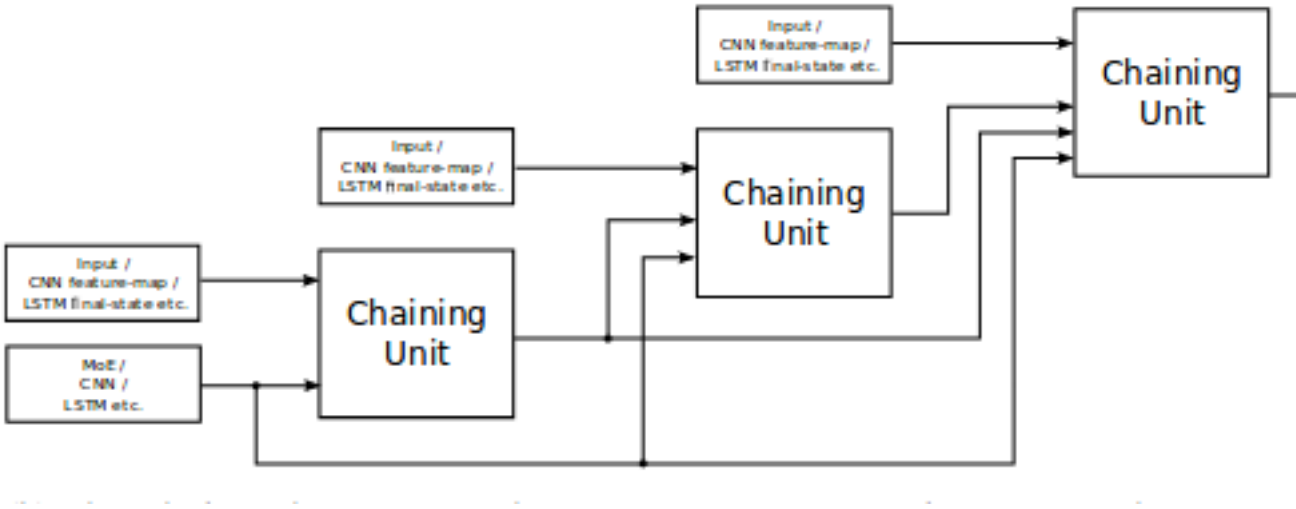

Figure 5.4: An illustration of the chaining layers.

To accelarate the training process, auxiliary losses were given to the intermediate predictions after each chaining unit. The auxiliary losses only contributed 10-20% of the total loss when training. The number of chaining units to stack

needs to be decided. The authors show that the addition of the chaining units increased the network's performance significantly.

Chaining has some similarities with recurrent nets, but where at each iteration all labels are predicted and no memory mechanism is involved and with a fixed number of iterations.

5.5 Learning from Complex Images

- using RNN attention

uses RL to train (https://nurture.ai/nips-challenge/p/2e855b9a-2e78-4788-98b2-6634301f35ab)

5.5.1 Spatial Relations

The Spatial Regularisation Network (SRN) is the first network to propose to model spatial relations between labels. They achieve this by training a subnet to learn attention maps representing the image regions for each label, from which a series of convolutional layers can then learn the label spatial relations.

As the backbone network, they first train a ResNet 101 on the relevant data. Then they use a set of lower level features from the backbone network to give as input to the SRN subnet. From these lower level visual features, the SRN subnet first learns a attention map for each label in a fully convolutional fashion, after which a series of convolutional layers take the attention maps as input and output the relevant labels for an image. This output is then added to the backbone net's predicted probabilities to obtain a final prediction. Figure @ref(fig:srn) provides an illustration.

They tested the network on MSCOCO, NUS-WIDE and WIDER-Attribute datasets and achieved very good results, far better that any other method. They also evaluated without first selecting a fixed number of labels. Although the authors attribute the SRN's success to its ability to model spatial relation, there are other reasons why this network performs so well.

Firstly, the ResNet is a more advanced network than VGG which was used by CNN-RNN and WARP. Secondly, when training, the authors of SRN used a more sophisticated form of data augmentation. They used a random scale augmentation amongst others, which allows the network to "see" an image at different sizes, usually help to recognise objects of different scales. Lastly, the

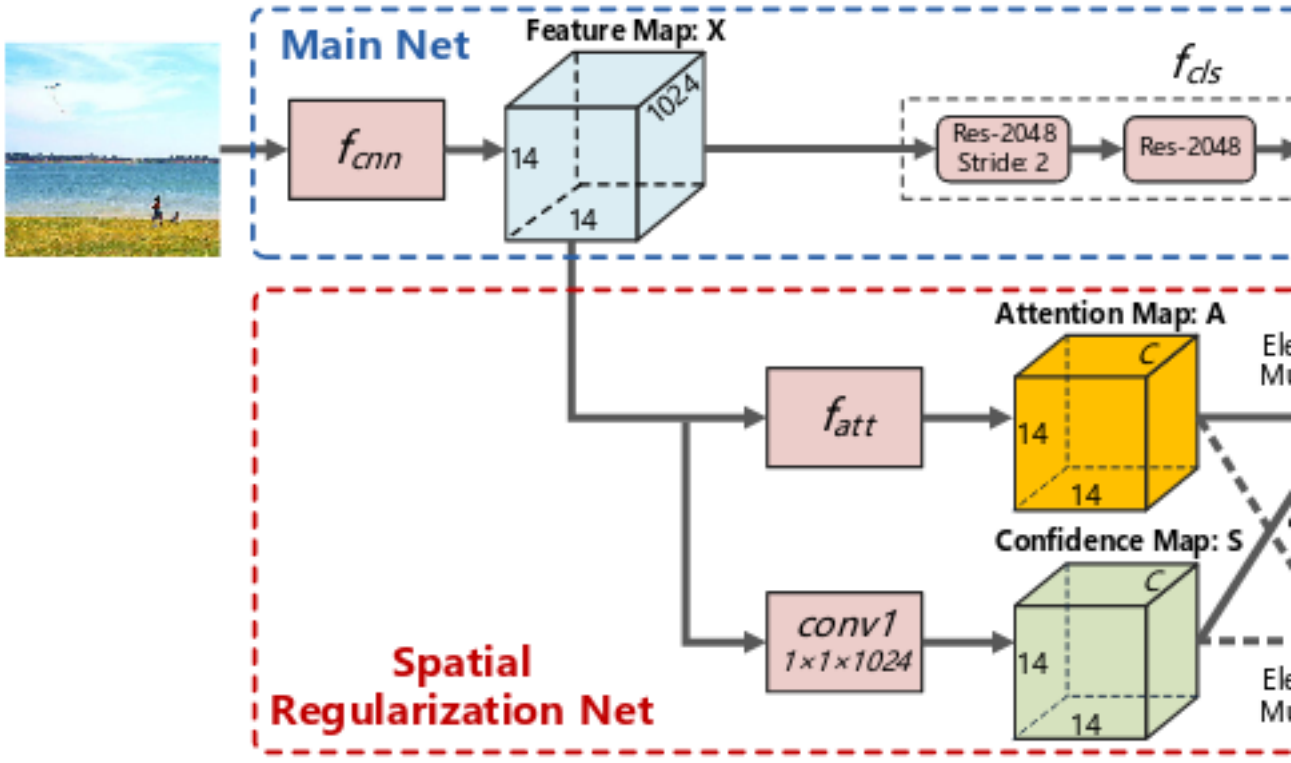

Figure 5.5: An illustration of the SRN framework.

fact that the SRN is built on top of a lower level visual feature map which is of higher resolution, also makes the detection of smaller objects easier. Thus it might be valuable to compare the SRN with the CNN-RNN on equal footing, where in addition the CNN part of CNN-RNN is allowed to be fine-tunend.

The SRN is trained using binary cross-entropy loss. One inconvenience is that the network is trained in multiple steps. We still consider this an unified network, since the steps only consist of freezing and unfreezing weights during training.

A bonus is that the SRN's attention maps can be visualised, giving localisation information on each image.

A challenge in MLIC not receiving nearly enough attention is the challenge of recognising objects of vastly different scales. Although the CNN-RNN architecture can do this. Fortunately in object detection this issue has been attended to.

See also: https://arxiv.org/pdf/1712.00433.pdf

5.5.2 Multi-Level Predictions

In brief, FPN augments a standard convolutional network with a top-down pathway and lateral connections so the network efficiently constructs a rich, multi-scale feature pyramid from a single resolution input image. Each level of the pyramid can be used for detecting objects at a different scale.

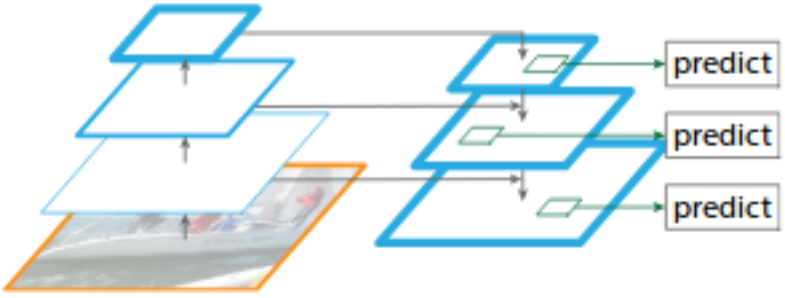

Figure 5.6: An illustration of the FPN concept in @Lin2016.

5.6 Automatic Thresholding

- multi-task learning
- top-k
- threshold

This contribution made in this work is two-fold: providing a novel loss function for MLC and introducing the idea of a learnable label decision module.

Recall, that a standard CNN returns scores for each class and that we are ultimately interested in the binary output. Usually we would use a thresholding function or select the top-k labels to determine which labels to include in the final predicted set. This approach does not take the input into account.

The learnable decision module proposed in this work takes the image into account when determining which labels to include in the final set. The decision module is a multi-layer perceptron (MLP) built on top of the penultimate layer of the classification network, which can either output the label count of the input image or the threshold to select the labels by. See Figure @ref(fig:imprank) for an illustration. If we want the label decision module to output the label count we can treat it as a n-way classification problem, where n is the maximum number of labels an image can have. The reason we choose to model it as

a classification task is so that integers are returned. Since it is a multiclass classification problem, the module is trained using a categorical cross-entropy loss.

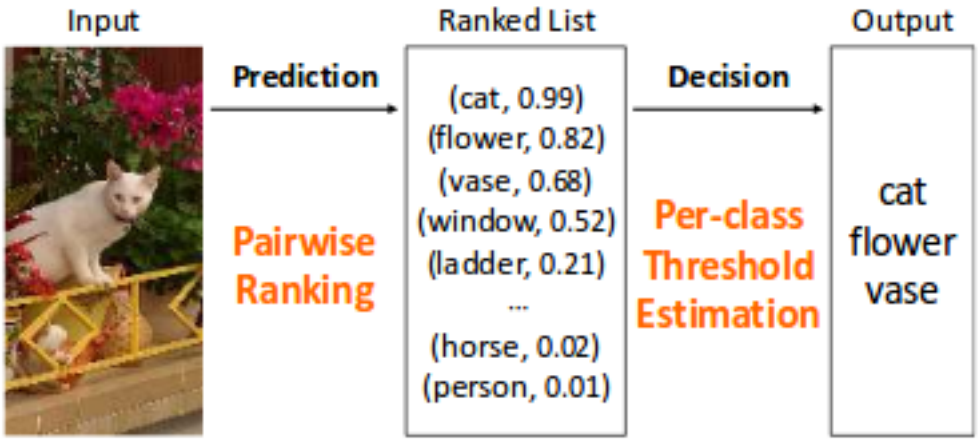

Figure 5.7: An illustration of the label decision module concept.

If we want the label decision module to return optimal thresholds for each label, we can treat it as a K dimensional regression task. The loss function then becomes more complicated and can be given by:

$$L_{thresh} = -\sum_{K=1}^{K} Y_{i,k} \log(\delta_\theta^k) + (1 - Y_{i,k}) \log(1 - \delta_\theta^k),$$

where $\delta_\theta^k = \sigma(f_k(\boldsymbol{x}_i) - \theta_k)$, with $\sigma(.)$ the sigmoid function and θ_k the predicted threshold for label k. Thus if the k-th label of observation i is positive, then we want $\delta_\theta^k > 0$ and therefore $f_k(\boldsymbol{x}_i) > \theta_k$, *i.e.* the score of class k to be higher than the threshold. A similar explanation can give for the negative label case.

The authors trained the classification network first and then trained the label decision module with the classification network weights frozen. The authors mention the full network can be trained in a multi-task learning manner (*i.e.* both networks simulataneously) but they observed better results doing it sequentially. They used a VGG pretrained on ImageNet as the classification CNN.

Their evaluations were done on all the major MLIC benchmark datasets (NUS-WIDE, MSCOCO, PASCAL VOC 2007) and reported the performance using the standard MLIC metrics along with the exact match. They compared their approach to using WARP and CNN-RNN, but not SRN. One of the other

baselines they used as a comparison is a multi-label adaption of the categorical cross-entropy loss which does not make sense to me. They found that the thresholding decision module combined with LSEP loss got the best results of all (even better that cross-validated select thresholds or top-k's). Although it does not seem to do better than SRN.

5.7 Weakly-Supervised Object Detection

- class activation maps
- fully convolutional networks

5.8 Conclusion

Chapter 6

Experiments and Results

"For us, the most important part of rigor is better empiricism, not more mathematical theories."

— Ali Rahimi and Ben Recht, *NIPS 2017*

6.1 Introduction

The main aim of this chapter is to empirically compare some of the deep learning for multi-label image classification approaches proposed in the literature in a standardised fashion. We will also attempt to empirically answer some of the questions that arose in the literature study.

Multi-label image classification with CNNs is still a relatively new research area. No work has been done to provide an extensive and robust comparison of the existing approaches in the literature. Typically, when a new approach is proposed it is empirically compared to other previous proposed approaches. But these evaluations of the approaches are not in a standardised fashion. The base networks and optimisation procedures are just some of the learning components that vary accross the proposed approaches. This makes it difficult to determine whether or not a proposed approach performs empirically better than another because of its ability to model multi-label images or because of the latest general developments of training CNNs.

Take the Spatial Regularisation Network (SRN) in the previous chapter as an example. The SRN is an extension of a base CNN that is supposed to help exploiting spatial relations amongs labels. The SRN shows favourable empirical results over all other proposed approaches. However, it also uses a much deeper base CNN (ResNet-101) than the other approaches in the literature. This

makes it difficult to determine whether or not the performance boost comes from the SRN or the deeper CNN.

For this reason, we want to provide a standardised and robust comparison of the some the most promising approaches in the literature. To standardise the comparisons, we will evaluate the chosen approaches using the same base CNN and optimisation procedure. To ensure robustness, we will evaluate the methods on two very distinct multi-label image datasets (described in Appendix **??**), using multiple diverse evaluation metrics and using cross-validation for a better estimate of generalisation ability which will also allow us to report standard deviations of errors.

There are 4 main question we attempt to answer in this chapter. They are:

1. How do the different loss functions act as a surrogate for the micro and macro F-score? (bce vs weighted bce vs rank loss vs retina loss)
2. Does multi-level predictions help to detect small objects?
3. Which extension works best to explicitly model label correlations? CG vs chaining vs SE-module
4. How does learnable label calibration modules compare to brute force search?

After getting closer to the answers of these questions we will train a final model taking into considerations the empirical findings to see how accurate we can get on both datasets.

- what am I going to do with SRN?

6.2 General Methodology

Unless explicitly stated otherwise, the general experimental methodology will be as follows. Consider the hypothetical case where we want to compare Approach A with Approach B:

- Evaluate Approach A and Approach B on both the Satellite Images and Chest X-Rays datasets. These datasets are vastly different in terms of the nature of the images and the size of the datasets. However they have

roughly the same number of possible labels and therefore our conclusions cannot be assumed to hold for task where K is much larger.

- For each dataset, evaluate the approaches with 5-fold cross-validation and report each evaluation metric as the average over the 5-folds along with its standard deviation. No other results in the literature are reported in this rigorous way.

- Compare approaches in terms of the following metrics:

  - label-based macro F_1-score (F_1^{macro})
  - label-based micro F_1-score (F_1^{micro})
  - example-based F_1-score (F_1^{exam})
  - example-based average precision (AP).

  These are chosen since they are the most popular metrics for multi-label image classification in the literature.

- Report the time taken for each approach to complete training where relevant. This measure is neglected in the literature but is an important factor, especially for use cases where resources are limited.

- When transfer learning is used, the same preprocessing of the images are done as was for in the training of the original network being transferred from.

- If no thresholding function is mentioned, then you may assume that a threshold of 0.5 for all labels was used.

6.3 Loss Functions for Multi-Label Image Classification

The goal of this experiment is to find out which multi-label loss function is a more suitable surrogate for our chosen multi-label evaluation metrics. We compare the binary cross-entropy loss, weighted binary cross-entropy loss, focal loss and LSEP loss. We only use one ranking based loss function since there was significant proof given in (Li *et al.*, 2017) that LSEP loss outperforms the other ranking based loss functions. We will also experiment with the focal

loss weighted as in W-CE. This is the first time the focal loss is used with multi-label image classification and the first time LSEP loss is compared to cross-entropy based loss functions.

For comparing the above mentioned loss functions, we will follow a transfer learning like approach. We will use a ResNet-50 pretrained on ImageNet to extract features for each image (of size 224×224), where the features we choose to extract is in the last layer before the classification layer of the ResNet. Thus we will get a 2048-dimensional vector for each image. The only training we will do is of a fully connected layer connecting the 2048-dimensional input to a K-dimensional output which is then passed through a sigmoid activation function (Revise number of layers during experiments). This network will be trained using SGD with an initial learning rate of 0.1 which is reduced by a factor of 10 when the loss plateaus. For each loss function we will only run 50 epochs. The rest of the details follow as in Section 6.2. Although, we will not look at the precision and recall, because they are summarised by the F_1-scores.

6.3.1 Results and Discussion

6.4 Multi-Level Predictions to Detect Small Objects

6.4.1 Method

6.4.2 Results

6.4.3 Discussion

6.5 Exploiting Label Correlations

6.5.1 Method

6.5.2 Results

6.5.3 Discussion

6.6 Label Calibration

6.6.1 Method

6.6.2 Results

6.6.3 Discussion

6.7 The Final Model

6.7.1 Method

6.7.2 Results

6.7.3 Discussion

6.8 Notes

- maybe also shrink images for faster computations.
- add time taken for learning as a metric.

Chapter 7

Conclusion

Appendices

Appendix A

Software and Code

- Deep Learning Library: Keras
- Hardware: AWS p2-instance
- R and Python
- github
- compiling thesis with Rmarkdown with version control with Github

A.1 Code and Reproducibility

Note that all of the code used in the thesis, including the source documents, is made available in the Thesis Github repository [1]. More instructions on how to implement the code is contained in the file named `README.md`, in the repository.

[1] https://github.com/jandremarais/Thesis

Bibliography

Alazaidah, R. and Ahmad, F.K. (2016). Trending Challenges in Multi Label Classification. *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 7, no. 10.
Available at: www.ijacsa.thesai.org

Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H. (2006). Greedy layer-wise training of deep networks. In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pp. 153–160. MIT Press, Cambridge, MA, USA.
Available at: http://dl.acm.org/citation.cfm?id=2976456.2976476

Chekina, L., Rokach, L. and Shapira, B. (2011). Meta-learning for selecting a multi-label classification algorithm. In: *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 220–227. ISBN 9780769544090. ISSN 15504786.

Chen, S.-F., Chen, Y.-C., Yeh, C.-K. and Wang, Y.-C.F. (2017 July). Order-free rnn with visual attention for multi-label classification. *ArXiv e-prints*. 1707.05495.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G.B. and LeCun, Y. (2014). The loss surface of multilayer networks. *CoRR*, vol. abs/1412.0233.
Available at: http://arxiv.org/abs/1412.0233

Chua, T.-S., Tang, J., Hong, R., Li, H., Luo, Z. and Zheng, Y. (2009). Nus-wide: A real-world web image database from national university of singapore. In: *Proceedings of the ACM International Conference on Image and Video Retrieval*, CIVR '09, pp. 48:1–48:9. ACM, New York, NY, USA. ISBN 978-1-60558-480-5.
Available at: http://doi.acm.org/10.1145/1646396.1646452

Clevert, D.-A., Unterthiner, T. and Hochreiter, S. (2015 November). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ArXiv e-prints*. 1511.07289.

Dauphin, Y., Pascanu, R., Gülçehre, Ç., Cho, K., Ganguli, S. and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, vol. abs/1406.2572.
Available at: http://arxiv.org/abs/1406.2572

Dembcz, K., Waegeman, W., Cheng, W., Hüllermeier, E., Tsoumakas, G., Zhang, M.-L., Zhou, Z.-H., Dembczyski, K., Waegeman, W., Cheng, W. and Hüllermeier, E. (2012). On label dependence and loss minimization in multi-label classification. *Mach Learn*, vol. 88, pp. 5–45.

Dembczy, K. (2010). Bayes Optimal Multilabel Classification via Probabilistic Classifier Chains. *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 279–286.
Available at: http://machinelearning.wustl.edu/mlpapers/paper{_}files/icml2010{_}DembczynskiCH10.pdfhttp://www.uni-marburg.de/fb12/kebi/people/cheng/cheng-icml10c.pdf

Dembszynski, K., Waegeman, W., Cheng, W. and Hüllermeier, E. (2010). On label dependence in multilabel classification. In: *LastCFP: ICML Workshop on Learning from Multi-label data*. Ghent University, KERMIT, Department of Applied Mathematics, Biometrics and Process Control.

Everingham, M., Gool, L., Williams, C.K., Winn, J. and Zisserman, A. (2010 June). The pascal visual object classes (voc) challenge. *IJCV*, vol. 88, no. 2, pp. 303–338. ISSN 0920-5691.
Available at: http://dx.doi.org/10.1007/s11263-009-0275-4

Everingham, M., Gool, L., Williams, C.K., Winn, J. and Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Fukushima, K. (1980 Apr). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202. ISSN 1432-0770.
Available at: https://doi.org/10.1007/BF00344251

Godbole, S. and Sarawagi, S. (2004). Discriminative Methods for Multi-labeled Classification. *Lecture Notes in Computer Science*, vol. 3056, pp. 22–30. ISSN 03029743. 978-3-540-24775-3{_}5.
Available at: http://link.springer.com/10.1007/978-3-540-24775-3{_}5

Gong, Y., Jia, Y., Leung, T., Toshev, A. and Ioffe, S. (2013). Deep convolutional ranking for multilabel image annotation. *CoRR*, vol. abs/1312.4894.
Available at: http://arxiv.org/abs/1312.4894

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

Hastie, T., Tibshirani, R. and Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction*. 2nd edn. Springer.
Available at: http://www-stat.stanford.edu/~tibs/ElemStatLearn/

Hinton, G.E., Osindero, S. and Teh, Y.-W. (2006 July). A fast learning algorithm for deep belief nets. *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554. ISSN 0899-7667.
Available at: http://dx.doi.org/10.1162/neco.2006.18.7.1527

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, vol. abs/1207.0580.
Available at: http://arxiv.org/abs/1207.0580

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, vol. 4, no. 2, pp. 251 – 257. ISSN 0893-6080.
Available at: http://www.sciencedirect.com/science/article/pii/089360809190009T

Huang, S.-j., Yu, Y. and Zhou, Z.-h. (2012). Multi-label hypothesis reuse. *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*, p. 525.
Available at: http://dl.acm.org/citation.cfm?id=2339530.2339615

Hubel, D.H. and Wiesel, T.N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154. ISSN 1469-7793.
Available at: http://dx.doi.org/10.1113/jphysiol.1962.sp006837

Ioannou, M., Sakkas, G., Tsoumakas, G. and Vlahavas, I. (2010). Obtaining bi-partitions from score vectors for multi-label classification. In: *Proceedings of the 2010 22Nd IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, ICTAI '10, pp. 409–416. IEEE Computer Society, Washington, DC, USA. ISBN 978-0-7695-4263-8.
Available at: http://dx.doi.org/10.1109/ICTAI.2010.65

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, vol. abs/1502.03167.
Available at: http://arxiv.org/abs/1502.03167

Karpathy, A. (). Stanford University CS231n: Convolutional Neural Networks for Visual Recognition.
Available at: http://cs231n.stanford.edu/syllabus.html

Klambauer, G., Unterthiner, T., Mayr, A. and Hochreiter, S. (2017 June). Self-Normalizing Neural Networks. *ArXiv e-prints*. 1706.02515.

Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012). Imagenet classification with deep convolutional neural networks. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pp. 1097–1105. Curran Associates Inc., USA.
Available at: http://dl.acm.org/citation.cfm?id=2999134.2999257

Lecun, Y., Bengio, Y. and Hinton, G. (2015 5). Deep learning. *Nature*, vol. 521, no. 7553, pp. 436–444. ISSN 0028-0836.

LeCun, Y., Haffner, P., Bottou, L. and Bengio, Y. (1999). *Object Recognition with Gradient-Based Learning*, pp. 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-46805-9.
Available at: https://doi.org/10.1007/3-540-46805-6_19

Li, Y., Huang, C., Loy, C.C. and Tang, X. (2016). Human attribute recognition by deep hierarchical contexts. In: *European Conference on Computer Vision*.

Li, Y., Song, Y. and Luo, J. (2017). Improving pairwise ranking for multi-label image classification. *CoRR*, vol. abs/1704.03135. 1704.03135.
Available at: http://arxiv.org/abs/1704.03135

Lin, T., Goyal, P., Girshick, R.B., He, K. and Dollár, P. (2017). Focal loss for dense object detection. *CoRR*, vol. abs/1708.02002. 1708.02002.
Available at: http://arxiv.org/abs/1708.02002

Lin, T., Maire, M., Belongie, S.J., Bourdev, L.D., Girshick, R.B., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L. (2014). Microsoft COCO: common objects in context. *CoRR*, vol. abs/1405.0312.
Available at: http://arxiv.org/abs/1405.0312

Liu, J., Chang, W.-C., Wu, Y. and Yang, Y. (2017). Deep learning for extreme multi-label text classification. In: *Proceedings of the 40th International ACM*

SIGIR Conference on Research and Development in Information Retrieval, SIGIR '17, pp. 115–124. ACM, New York, NY, USA. ISBN 978-1-4503-5022-8.
Available at: http://doi.acm.org/10.1145/3077136.3080834

Luaces, O., Díez, J., Barranquero, J., Del Coz, J.J. and Bahamonde, A. (). Binary Relevance Efficacy for Multilabel Classification.

Madjarov, G., Kocev, D., Gjorgjevikj, D. and Džeroski, S. (2012). An extensive experimental comparison of methods for multi-label learning.
Available at: http://www.elsevier.com/copyright

Miech, A., Laptev, I. and Sivic, J. (2017 June). Learnable pooling with context gating for video classification. *ArXiv e-prints*. 1706.06905.

Nam, J., Kim, J., Gurevych, I. and Fürnkranz, J. (2013). Large-scale multi-label text classification - revisiting neural networks. *CoRR*, vol. abs/1312.5419.
Available at: http://arxiv.org/abs/1312.5419

Oakden-Rayner, L. (2017 Nov). Quick thoughts on chestxray14, performance claims, and clinical tasks.
Available at: https://lukeoakdenrayner.wordpress.com/2017/11/18/quick-thoughts-on-chestxray14-performance-claims-and-clinical-tasks/

Organization, W.H. *et al.* (2001). Standardization of interpretation of chest radiographs for the diagnosis of pneumonia in children.

Pachet, F. and Roy, P. (2009). Improving Multilabel Analysis of Music Titles: A Large-Scale Validation of the Correction Approach. *IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, vol. 17, no. 2.

Prabhu, Y. and Varma, M. (2014). Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pp. 263–272. ACM, New York, NY, USA. ISBN 978-1-4503-2956-9.
Available at: http://doi.acm.org/10.1145/2623330.2623651

Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., Ding, D., Bagul, A., Langlotz, C., Shpanskaya, K., Lungren, M.P. and Ng, A.Y. (2017). Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *CoRR*, vol. abs/1711.05225. 1711.05225.
Available at: http://arxiv.org/abs/1711.05225

Ramachandran, P., Zoph, B. and Le, Q.V. (2017). Searching for activation functions. *CoRR*, vol. abs/1710.05941. 1710.05941.
Available at: http://arxiv.org/abs/1710.05941

RamóN Quevedo, J., Luaces, O. and Bahamonde, A. (2012 February). Multilabel classifiers with a probabilistic thresholding strategy. *Pattern Recogn.*, vol. 45, no. 2, pp. 876–883. ISSN 0031-3203.
Available at: http://dx.doi.org/10.1016/j.patcog.2011.08.007

Read, J. (2008). A pruned problem transformation method for multi-label classification. *New Zealand Computer Science Research Student Conference, NZCSRSC 2008 - Proceedings*, , no. April, pp. 143–150.
Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84880106608{&}partnerID=40{&}md5=ccdd19f7d0f111a2fdda8df7af0a8075

Read, J., Pfahringer, B., Holmes, G., Frank, E., Brodley Read, C.J., Pfahringer, B., Holmes, G. and Frank, E. (2011). Classifier chains for multi-label classification. *Mach Learn*, vol. 85, no. 85, pp. 333–359. ISSN 08856125. arXiv:1207.6324.
Available at: http://download.springer.com/static/pdf/44/art{%}253A10.1007{%}252Fs10994-011-5256-5.pdf?originUrl=http{%}3A{%}2F{%}2Flink.springer.com{%}2Farticle{%}2F10.1007{%}2Fs10994-011-5256-5{&}token2=exp=1490608886{~}acl={%}2Fstatic{%}2Fpdf{%}2F44{%}2Fart{%}25253A10.1007{%}25252Fs10994-011-5256-

Rosenblatt (1957). *The perceptron, a perceiving and recognizing automaton, Project Para.* Cornell Aeronautical Laboratory.
Available at: /reference-material/rosenblatt1957perceptron.pdf

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1988). Neurocomputing: Foundations of research. chap. Learning Representations by Back-propagating Errors, pp. 696–699. MIT Press, Cambridge, MA, USA. ISBN 0-262-01097-6.
Available at: http://dl.acm.org/citation.cfm?id=65669.104451

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C. and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252.

Sánchez, J., Perronnin, F., Mensink, T. and Verbeek, J. (2013 December). Image classification with the fisher vector: Theory and practice. *Int. J. Comput. Vision*, vol. 105, no. 3, pp. 222–245. ISSN 0920-5691.
Available at: http://dx.doi.org/10.1007/s11263-013-0636-x

Sechidis, K., Tsoumakas, G. and Vlahavas, I. (2011). On the Stratification of Multi-label Data.
Available at: http://download.springer.com/static/pdf/229/chp{%}253A10.1007{%}252F978-3-642-23808-6{_}10.pdf?originUrl=http{%}3A{%}2F{%}2Flink.springer.com{%}2Fchapter{%}2F10.1007{%}2F978-3-642-23808-6{_}10{&}token2=exp=1489589222{~}acl={%}2Fstatic{%}2Fpdf{%}2F229{%}2Fchp{%}25253A10.1007{%}25252F978-3-64

Selfridge, O.G. (1959). Pandemonium: a paradigm for learning. In The mechanisation of thought processes.

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227 – 244. ISSN 0378-3758.
Available at: http://www.sciencedirect.com/science/article/pii/S0378375800001154

Systems, E. and Aviv-yafo, T. (2014). Ensemble Methods for Multi-label Classification Ensemble Methods for Multi-label Classification. , no. July 2013.

Szymański, P. and Kajdanowicz, T. (2017 April). A Network Perspective on Stratification of Multi-Label Data. *ArXiv e-prints*. 1704.08756.

Tsoumakas, G., Dimou, A., Spyromitros, E., Mezaris, V., Kompatsiaris, I. and Vlahavas, I. (2009). Correlation-based pruning of stacked binary relevance models for multi-label learning. *Proceedings of the Workshop on Learning from Multi-Label Data (MLD'09)*, pp. 101–116. ISSN 1475-925X.
Available at: http://www.ecmlpkdd2009.net/wp-content/uploads/2008/09/learning-from-multi-label-data.pdf{#}page=102

Tsoumakas, G. and Katakis, I. (2007). Multi-Label Classification : An Overview. ISSN 1548-3924.

Tsoumakas, G., Katakis, I. and Vlahavas, I. (2008). Effective and efficient multilabel classification in domains with large number of labels. *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, pp. 30–44.
Available at: http://lpis.csd.auth.gr/publications/tsoumakas-mmd08.pdf

Tsoumakas, G. and Vlahavas, I. (2007). Random k-labelsets: An Ensemble Method for Multilabel Classification. *European Conference on Machine Learning*, pp.

406–417. ISSN 01681605.
Available at: http://link.springer.com/chapter/10.1007/978-3-540-74958-5{_}38

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C. and Xu, W. (2016). CNN-RNN: A unified framework for multi-label image classification. *CoRR*, vol. abs/1604.04573.
Available at: http://arxiv.org/abs/1604.04573

Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M. and Summers, R. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3462–3471.

Wei, Y., Xia, W., Huang, J., Ni, B., Dong, J., Zhao, Y. and Yan, S. (2014). CNN: single-label to multi-label. *CoRR*, vol. abs/1406.5726.
Available at: http://arxiv.org/abs/1406.5726

Wu, X. and Zhou, Z. (2016). A unified view of multi-label performance measures. *CoRR*, vol. abs/1609.00288. 1609.00288.
Available at: http://arxiv.org/abs/1609.00288

Xu, C., Tao, D. and Xu, C. (2016). Robust Extreme Multi-Label Learning. *KDD*, pp. 421–434. ISSN 0146-4833. arXiv:1602.05561v1.

Yang, Y. (2001). A study of thresholding strategies for text categorization. In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, pp. 137–145. ACM, New York, NY, USA. ISBN 1-58113-331-6.
Available at: http://doi.acm.org/10.1145/383952.383975

Yao, L., Poblenz, E., Dagunts, D., Covington, B., Bernard, D. and Lyman, K. (2017). Learning to diagnose from scratch by exploiting dependencies among labels. *CoRR*, vol. abs/1710.10501. 1710.10501.
Available at: http://arxiv.org/abs/1710.10501

Zhang, L., Towsey, M., Xie, J., Zhang, J. and Roe, P. (2016). Using multi-label classification for acoustic pattern detection and assisting bird species surveys. *Applied Acoustics*, vol. 110, no. Supplement C, pp. 91 – 98. ISSN 0003-682X.
Available at: http://www.sciencedirect.com/science/article/pii/S0003682X16300603

Zhang, M.-L. and Zhou, Z.-H. (2007). ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, vol. 40, pp. 2038–2048.
Available at: www.elsevier.com/locate/pr

Zhang, M.L. and Zhou, Z.H. (2014). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1819–1837. ISSN 10414347.

Zhang, M.-l., Zhou, Z.-h. and Member, S. (2006). Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization. vol. 18, no. 10, pp. 1338–1351.

Zhang, W., Wang, L., Yan, J., Wang, X. and Zha, H. (2017). Deep extreme multi-label learning. *CoRR*, vol. abs/1704.03718. 1704.03718.
Available at: http://arxiv.org/abs/1704.03718

Zhu, F., Li, H., Ouyang, W., Yu, N. and Wang, X. (2017). Learning spatial regularization with image-level supervisions for multi-label image classification. *CoRR*, vol. abs/1702.05891. 1702.05891.
Available at: http://arxiv.org/abs/1702.05891