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

Constants

g =	$9.81\,\mathrm{m/s^2}$	

Variables

Re_D	Reynolds number (diameter)	[ ]
x	Coordinate	[m]
$\ddot{x}$	Acceleration	[$\mathrm{m/s^2}$]
θ	Rotation angle	[rad]
τ	Moment	[N·m]

Vectors and Tensors

$\overrightarrow{v}$	Physical vector, see equation ...

Subscripts

a	Adiabatic
a	Coordinate

Chapter 1

Introduction

1.1 Motivation

Image Classification concerns the task of assigning one (or more) label(s) to an input image. This seemingly simple task is one of the core problems in *Computer Vision*, and also has a large variety of practical applications. Examples of such applications include detecting deforestation in the Amazon from satellite images [1], or autonomous lung disease diagnosis from chest X-rays (Wang *et al.*, 2017). Image classification is a thoroughly researched topic and already regarded by many to be a 'solved' problem. This progress is mainly attributed to the yearly large-scale image classification competition, called *ImageNet*[2], providing researchers with millions of labelled images to train their image classification models.

The other driver of the recent success of image classification systems is the development of *Deep Learning* (Lecun *et al.*, 2015), a subfield of Machine Learning. Deep Learning (by means of *deep neural networks*, or DNNs) extends upon artificial neural networks, which were designed in an attempt to mimic the structure and function of the human brain. The type of DNNs proven best suited for image classification problems is the class of *Convolutional Neural Networks* (CNNs). The distinguishing feature of CNNs is its ability to 'learn' highly discriminative feature representations of input images, whereas more conventional approaches require one to manually hand-craft image features for each specific task (for example in (Lowe, 2004)).

Consider Figure 1.1 for an illustration of the impact that CNNs have made in image classification. Notice the large decrease from 2011 to 2012 when the first CNN (Krizhevsky *et al.*, 2012) was used in this competition. Thereafter all of the winning solutions made use of CNNs. CNNs are now able to surpass human level performace on this task.

Until recently, the main focus of deep learning approaches for image clas-

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

[2] http://www.image-net.org/

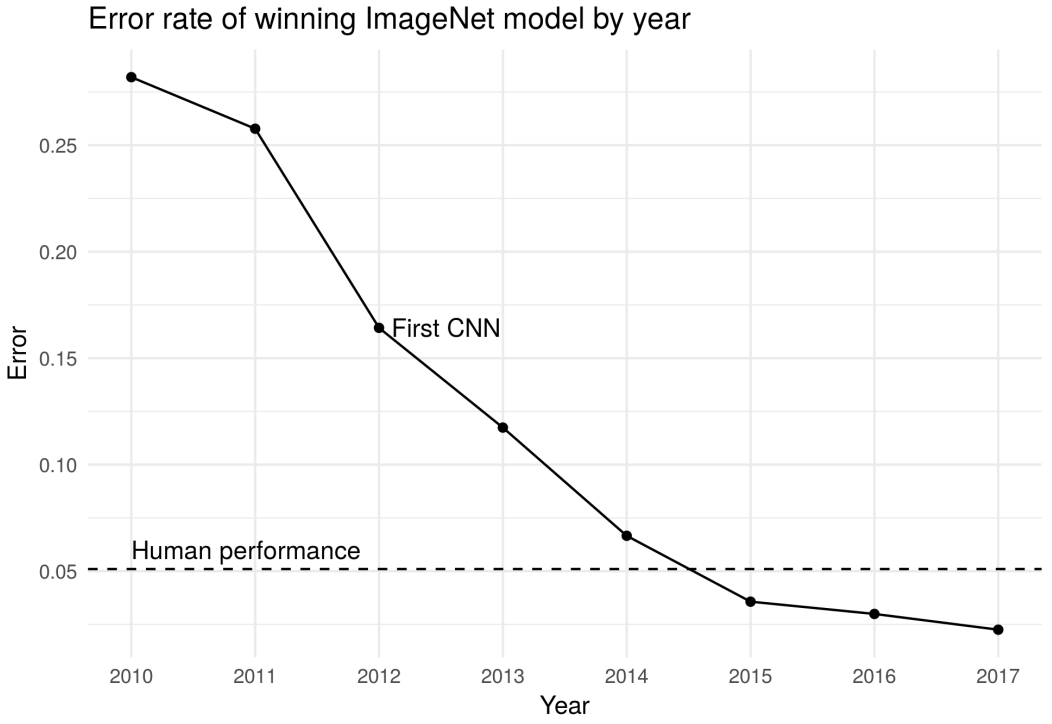

Figure 1.1: Classification error rates of the winning ImageNet models by year.

sification was on problems where each image is annotated with only a single label. *Multi-Label Classification* (MLC) of images generalises this task to allow images to be simultaneously annotated with more than one label. The number of labels to associate with each image is unknown, and differs across images. By definition, multi-label image classification is much more challenging than conventional (single-label) classification. Moreover, since the majority of real-world images contain more than one object, multi-label image classification is a more generally occurring problem in practice than single-label classification. In research, it was only fairly recently that more attention was paid to multi-label image classification (for example in (Wei *et al.*, 2014)). Therefore the field is regarded far from the level of maturity of its single-label counterpart. Since CNNs have proven to be so successful in the case of single-label image classification, it seems reasonable to ask whether their use may not also be extended to successfully annotate images with multiple labels.

Extending CNNs to handle images with multiple labels is not a trivial task. Thus far only a handful of proposals regarding how to tackle multi-label image classification have been published (for example in (Gong *et al.*, 2013, Wei *et al.* (2014), Wang *et al.* (2016), Zhu *et al.* (2017*a*), Chen *et al.* (2017))). These contributions were however mostly made in isolation of one another – hence no comprehensive comparison of these methods (whether theoretical or empirical) can be found. Also, the majority of proposals stem from a deep

learning background and could possibly gain from insights and methods already discovered in the MLC field.

In summary, the relevance of multi-label image classification in terms of its wide range of useful practical applications, combined with the power of deep learning methods for image classification tasks, forms the main motivation for the chosen research topic. Deep learning for multi-label image classification constitutes a research area that seems still to be relatively under-explored. As such, it may well be that a comprehensive understanding and review of the subject could lead to novel contributions.

1.2 Objectives

The core objective of this study is to learn how to effectively apply CNNs in a multi-label image classification context. In its pursuit, a thorough understanding of the following research areas seems essential:

- **The image classification problem** - Since this is the application domain of the envisioned research, a good understanding of the problem forms a solid foundation for further study in the domain. Through a discussion of current approaches to the problem, challenges and possible shortcomings in multi-label image classification may be identified.
- **Deep neural networks** - Being the class of algorithms which the study focuses on, an understanding of DNNs is essential to further explore ways in which to extend this class of models. Attention should be paid to the typical structures of DNNs in the context of image classification. Another important aspect is the way in which variations in DNN architectures influence the effectiveness of algorithms in scenarios of interest. Moreover, learning about the limitations of current networks may aid in directing the study.
- **The multi-label classification framework** - This is the *supervised learning* paradigm within which the research takes place. Although many contributions have been made to the MLC field, most MLC procedures only recently became more popular. Knowledge of the underlying concepts (especially those that may be considered novel relative to single-label classification), and insight into state-of-the art algorithms and challenges in the field, should guide the research. Although the focus will mainly be guided by the application domain, a review of previous work on multi-label image classification (other than DNNs) may also prove insightful.

The above objectives regarding the literature review are complementary to the main objective of the study, which may be partitioned as follows:

- **Extensions of DNNs to the MLC problem domain** - Despite sparse literature contributions in this regard, there exists a sufficient body of proposals in order to warrant a review on the subject. The aim will be to obtain a thorough understanding of each proposal in terms of both the way in which it solves the MLC problem, and in terms of its effectiveness in doing so. Conceptual differences among the approaches should form an important focus. If proposals for DNNs in MLC with other application domains are found, the possibility of transferring them to image classification should also be considered.

- **Improvements to existing approaches** - Learning from the strengths and weaknesses of each existing approach should provide a good basis for further research in terms of either extensions of existing approaches, or in terms of entirely novel proposals. The possibility of transferring ideas in pure MLC to deep learning to improve upon existing approaches should also be explored.

- **Empirical evaluation and comparison of approaches** - The literature review and theoretical comparisons should be complemented by empirical evaluations. Existing and proposed classification procedures may be evaluated and compared by means of benchmark image datasets. The practical implementation of algorithms may highlight challenges that may not necessarily be found in the literature.

Through the above objectives, the thesis should guide a reader to a thorough understanding of recent work on CNNs for multi-label image classification. Given a certain multi-label image classification problem, one should be able to recommend some approaches above others, know their advantages and limitations, and be able to effectively implement them.

1.3 Background and Important Concepts

In this thesis, the three key topics of this study, *viz.* image classification, convolutional neural networks and multi-label classification, will be introduced in their own chapters (image classification here, and the other in the following two chapters). Subsequently, a discussion of the combination of the above topics, in order to conceptualise multi-label image classification by means of CNNs, will follow.

Since we are interested in applying CNNs for MLC in the image classification domain, some background on image classification is first required. Image classification is an example of a *superivised learning* task, hence in the remainder of this chapter, we start with a brief introduction to supervised learning followed by the general problem of image classification. Discussions of CNNs and of

MLC follow in Chapters 2 and 3 respectively. A more comprehensive outline of the thesis may be found in Section 1.4.

1.3.1 Statistical Learning

Machine or statistical learning algorithms (used interchangably) are used to perform certain task that are too difficult or inefficient to solve with fixed rule-based programs. These algorithms are able to learn how to perform a task from data. For an algorithm to learn from data means that it can improve its ability in performing an assigned *task*, with respect to some *performance measure*, by processing *data*. This section gives a brief look at some of the important types of tasks, data and performance measures in the field of statistical learning.

A learning task describes the way an algorithm should process an observation. An observation is a collection of features that have been measured from some object or event that we want the system to process, for example an image. We will represent an observation by a vector $\boldsymbol{x} \in \mathbb{R}^p$ where each element x_j of the vector is an observed value of the j-th feature, $j = 1, \ldots, p$. For example, the features of an image are usually the color intensity values of the pixels in the image.

Many kinds of tasks can be solved with statistical learning. One of the most common learning tasks is that of *classification*, where it is expected of an algorithm to determine which of K categories an input belongs to. To solve the classification task, the learning algorithm is usually asked to produce a function $f : \mathbb{R}^p \to \{1, \ldots, K\}$, which we can also refer to as the model. When $y = f(\boldsymbol{x})$, the model assigns an input described by the vector $\boldsymbol{x}$ to a category identified by the numeric code y, called the *output* or *response*. In other variants of the classification task, f may output a probability distribution over the possible classes.

Regression is the other main learning task and requires the algorithm to predict a continuous value given some input. This task requires a function $f : \mathbb{R}^p \to \mathbb{R}$, where the difference to classification is the format of its output.

Learning algorithms can learn to perform such tasks by observing a relevant set of data points, *i.e.* a dataset. A dataset containing N observations of p features is commonly described as a design matrix $X : N \times p$, where each row of the matrix represents a different observation and each column corresponds to a different feature of the observations, *i.e.*

$$X = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1p} \\ x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \ldots & x_{Np} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_1^\intercal & \boldsymbol{x}_2^\intercal & \ldots & \boldsymbol{x}_N^\intercal \end{bmatrix}.$$

$\boldsymbol{x}_i^\intercal$ represents the p-dimensional vector that forms the i-th row of X.

Often the dataset includes annotations for each observation in the form of a label (classification) or a target value (regression). The N annotations are represented by the vector $\boldsymbol{y}$, where element y_i is associated with the i-th row of X. Therefore the response vector may be denoted by

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}.$$

Note that in the case of multiple labels or targets, a matrix representation $Y : N \times K$ is required.

Statistical learning algorithms can be divided into two main categories, *supervised* and *unsupervised* algorithms, determined by the presence or absence of annotations in the dataset to be analysed. Unsupervised learning algorithms learn from data consisting only of features, X, and are used to find useful properties and structure in the dataset (see Hastie *et al.*, 2009, Ch. 14). On the other hand, superivised learning algorithms learn from datasets which consist of both features and annotations, (X, Y), with the aim to model the relationship between them. Therefore, both classification and regression are considered to be supervised learning tasks.

In order to evaluate the ability of a learning algorithm to perform its assigned task, we have to design a quantitative performance measure. For example, in a classification task we are usually interested in the accuracy of the algorithm, *i.e.* the percentage of times that the algorithm makes the correct classification. We are mostly interested in how well the learning algorithm performs on data that it has not seen before, since this demonstrates how well it will perform in real-world situations. Thus we evaluate the algorithm on a *test set* of data points, independent of the *training set* of data points used during the learning process.

For a more concrete example of supervised learning, and keeping in mind that the linear model is one of the main building blocks of neural networks, consider the learning task underlying *linear regression*. The objective here is to construct a system which takes a vector $\boldsymbol{x} \in \mathbb{R}^p$ as input and predicts the value of a scalar $y \in \mathbb{R}$ in response. In the case of linear regression, we assume the output be a linear function of the input. Let $\hat{y}$ be the predicted response. We define the output to be

$$\hat{y} = \hat{\boldsymbol{w}}^T \boldsymbol{x},$$

where $\hat{\boldsymbol{w}} = [w_0, w_1, \ldots, w_p]$ is a vector of parameters and $\boldsymbol{x} = [1, x_1, x_2, \ldots, x_p]$. Note that an intercept is included in the model (also known as a *bias* in machine learning, not to be confused with bias in the statistical sence). The parameters are values that control the behaviour of the system. We can think of them as a

set of *weights* that determine how each feature affects the prediction. Hence the learning task can be defined as predicting y from $\boldsymbol{x}$ through $\hat{y} = \hat{\boldsymbol{w}}^T\boldsymbol{x}$.

Next, we need to define a performance measure to evaluate the linear predictions. For a set of observations, an evaluation metric tells us how (dis)similar the predicted output is to the actual response values. A very common measure of performance in regression is the *mean squared error* (MSE), given by

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

which is evaluated over a set of N observations. The process of learning from the data (or fitting a model to the data) can be reduced to the following optimisation problem: find the set of weights, $\hat{\boldsymbol{w}}$, which produces a $\hat{\boldsymbol{y}}$ that minimises the MSE. This problem has a closed form solution and can quite trivially be found by means of *ordinary least squares* (OLS) (see Hastie *et al.*, 2009, p. 12). However, we have mentioned that we are more interested in the algorithm's performance evaluated on a test set. Unfortunately the least squares solution does not guarantee the solution to be optimal in terms of the MSE on a test set, rendering statistical learning to be much more than a pure optimisation problem.

The ability of a model to perform well on previously unobserved inputs is referred to as its *generalisation* ability. Generalisation is the key challenge of statistical learning. One way of improving the generalisation ability of a linear regression model is to modify the optimisation criterion J, to include a *weight decay* (or *regularisation*) term. That is, we want to minimise

$$J(\boldsymbol{w}) = MSE_{\text{train}} + \lambda\boldsymbol{w}^T\boldsymbol{w},$$

where $J(\boldsymbol{w})$ now expresses preference for smaller weights. The parameter λ is non-negative and needs to be specified ahead of time. It controls the strength of the preference by determining how much influence the penalty term, $\boldsymbol{w}^T\boldsymbol{w}$, has on the optimisation criterion. If $\lambda = 0$, no preference is imposed, and the solution is equivalent to the OLS solution. Larger values of λ force the weights to decrease, and thus referred to as a so-called *shrinkage* method ((*cf.* for example Hastie *et al.*, 2009, pp. 61-79) and Goodfellow *et al.* (2016). This approach of modifying the learning algorithm to attempt to reduce its generalisation error is known as *regularisation* (Goodfellow *et al.*, 2016, pp. 118-120).

We can further generalise linear regression to the classification scenario. First, note the different types of classification schemes. Consider $\mathcal{G}$, the discrete set of values which may be assumed by G, where G is used to denote a categorical output variable (instead of Y). Let $|\mathcal{G}| = K$ denote the number of discrete categories in the set $\mathcal{G}$. The simplest form of classification is known as binary classification and refers to scenarios where the input is associated

with only one of two possible classes, *i.e.* $K = 2$. When $K > 2$, the task is known as multiclass classification. In multi-label classification an input may be associated with multiple classes (out of K available classes), where the number of classes that each observation belongs to, is unknown. A thorough discussion of MLC methods is given in Chapter 3. Here we start by introducing the two single label classification setups, *viz.* binary and multiclass classification.

In multiclass classification, given the input values $\boldsymbol{X}$, we would like to accurately predict the output, G, which we denote by $\hat{G}$. One approach would be to represent G by an indicator vector $\boldsymbol{Y}_G : K \times 1$, with elements all zero except in the G-th position, where it is assigned a 1, *i.e.* $Y_k = 1$ for $k = G$ and $Y_k = 0$ for $k \neq G$, $k = 1, 2, ..., K$. We may then treat each of the elements in $\boldsymbol{Y}_G$ as quantitative outputs, and predict values for them, denoted by $\hat{\boldsymbol{Y}} = [\hat{Y}_1, \ldots, \hat{Y}_K]$. The class with the highest predicted value will then be the final categorical prediction of the classifer, *i.e.* $\hat{G} = \arg\max_{k \in \{1,\ldots,K\}} \hat{Y}_k$.

Within the above framework we therefore seek a function of the inputs which is able to produce accurate predictions of the class scores, *i.e.*

$$\hat{Y}_k = \hat{f}_k(\boldsymbol{X}),$$

for $k = 1, \ldots, K$. Here $\hat{f}_k$ is an estimate of the true function, f_k, which is meant to capture the relationship between the inputs and output of class k. As with the linear regression case described above, we can use a linear model $\hat{f}_k(\boldsymbol{X}) = \hat{\boldsymbol{w}}_k^T \boldsymbol{X}$ to approximate the true function. The linear model for classification divides the input space into a collection of regions labelled according to the classification, where the division is done by linear *decision boundaries* (see Figure 1.2 for an illustration). The decision boundary between classes k and l is the set of points for which $\hat{f}_k(\boldsymbol{x}) = \hat{f}_l(\boldsymbol{x})$. These set of points form an affine set or hyperplane in the input space.

After the weights are estimated from the data, an observation represented by $\boldsymbol{x}$ (including the unit element) can be classified as follows:

- Compute $\hat{f}_k(\boldsymbol{x}) = \hat{\boldsymbol{w}}_k^T \boldsymbol{x}$ for all $k = 1, \ldots, K$.
- Identify the largest component and classify to the corresponding class, *i.e.* $\hat{G} = \arg\max_{k \in \{1,\ldots,K\}} \hat{f}_k(\boldsymbol{x})$.

One may view the predicted class scores as estimates of the conditional class probabilities (or posterior probabilities), *i.e.* $P(G = k|\boldsymbol{X} = \boldsymbol{x}) \approx \hat{f}_k(\boldsymbol{x})$. However, these values are not the best estimates of posterior probabilities. Although the values sum to 1, they do not lie within [0,1]. A way to overcome this problem is to estimate the posterior probabilities using the *logit transform* of $\hat{f}_k(\boldsymbol{x})$. That is,

$$P(G = k|\boldsymbol{X} = \boldsymbol{x}) \approx \frac{e^{\hat{f}_k(\boldsymbol{x})}}{\sum_{l=1} e^{\hat{f}_l(\boldsymbol{x})}}.$$

Through this transformation, the estimates of the posterior probabilities both sum to 1 and are squeezed into [0,1]. The above model is the well-known *logistic regression* model (Hastie *et al.*, 2009, p. 119). With this formulation there is no closed form solution for the weights. Instead, the weight estimates may be searched for by maximising the log-likelihood function. One way of doing this is by minimising the negative log-likelihood using gradient descent, which will be discussed in the following section.

Finally in this section, note that any supervised learning problem can also be viewed as a function approximation problem. Suppose we are trying to predict a variable Y given an input vector $\boldsymbol{X}$, where we assume the true relationship between them to be given by

$$Y = f(\boldsymbol{X}) + \epsilon,$$

where ϵ represents the part of Y that is not predictable from $\boldsymbol{X}$, because of, for example, incomplete features or noise present in the labels. Then in function approximation we are estimating f with an estimate $\hat{f}$. In parametric function approximation, for example in linear regression, estimation of $f(\boldsymbol{X}, \theta)$ is equivalent to estimating the optimal set of weights, $\hat{\theta}$. In the remainder of the thesis, we refer to $\hat{f}$ as the *model*, *classifier* or *learner*.

1.3.2 Optimisation

As mentioned before, fitting a linear regression model can be reduced to finding the optimal weights to minimise the MSE function (with or without weight decay). In fact, typically model training procedures can be described as the search for its internal parameters that minimises or maximises some *objective function*. Therefore statistical learning and optimisation are closely related. Optimisation refers to the task of either minimising or maximising some function $J(x)$ by altering x. The function we want to optimise is called the objective function. When we are minimising the objective function, we may also refer to the objective function as the *cost* or *loss function*. These terms will be used interchangeably throughout the remainder of the thesis.

As mentioned in the previous section, parameter estimation (or optimisation) of a linear (or logistic regression) model is usually done using OLS or maximum likelihood estimation (MLE). In this section, however, we discuss an alternative parameter estimation method which is also relevant for the optimisation of neural networks.

Consider the MSE loss function:

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

where $f_k(\cdot)$ in this case is the linear model used to predict the k-th class posterior probability. Although the MSE loss is mostly used in a regression setup and not really well suited for classification, we make use of it here for illustration purposes.

To find the weights, $\boldsymbol{w}$, that minimise L, we can follow a process of iterative refinement. That is, starting with a random initialisation of $\boldsymbol{w}$, one iteratively updates the values such that L decreases. The updating steps are repeated until the loss converges. In order to minimise L with respect to $\boldsymbol{w}$, we calculate the gradient of the loss function at the point $L(\boldsymbol{x}; \boldsymbol{w})$. The gradient (or slope) of the loss function indicates the direction in which the function has the steepest rate of increase. Therefore, once we have determined this direction, we can update the weights by a step in the opposite direction - thereby reaching a smaller value of L.

The gradient of L_i is computed by obtaining the partial derivative of L_i with respect to $\boldsymbol{w}_k$, *i.e.*:

$$\frac{\partial L_i}{\partial \boldsymbol{w_k}} = -2(y_{ik} - \boldsymbol{w}_k^T \boldsymbol{x}_i)\boldsymbol{x}_i$$

Given the above derivatives, an update at the $(r+1)$-th iteration has the form

$$\boldsymbol{w}_k^{(r+1)} = \boldsymbol{w}_k^{(r)} - \gamma \sum_{i=1}^{n} \frac{\partial L_i}{\partial \boldsymbol{w}_k^{(r)}},$$

where γ is called the *learning rate* and determines the size of the step taken toward the optimal direction. One typically would like to set the learning rate small enough so that one does not overshoot the minimum, but large enough to make significant progress towards the minimum. This value can be determined via a line search but is not always ideal since this may render the training time of DNNs too long. Another option is to reduce the learning rate after every fixed number of iterations. More detail regarding the implication of the learning rate will be given in Chapter 2.

The procedure of repeatedly evaluating the gradient of the objective function and then performing a parameter update, is called *gradient descent* [Cauchy, 1847]. Gradient descent forms the basis of the optimisation procedure for neural networks.

Note that a weight update is made by evaluating the gradient over a set of observations, $\{\boldsymbol{x}_i, i = 1, \ldots, n\}$. One of the advantages of gradient descent is that at any iteration, the gradient need not be computed over the complete training dataset, *i.e.* $n \leq N$. When updates are iteratively determined by using randomly sampled subsets of the data, the process is called *mini-batch gradient descent*. This is extremely helpful in large-scale applications, since it does not require the computation of the full loss function over the entire dataset. This leads to faster convergence, because of more frequent parameter updates, and allows processing of datasets that are too large to fit into a computer's memory. The choice regarding batch size depends on the available computation power. Typically a batch consists of 64, 128 or 256 data points, since in practice many vectorised operation implementations work faster when their inputs are sized in powers of 2. The gradient obtained using mini-batches is only an approximation of the gradient of the full loss but it seems to be sufficient in practice (Li *et al.*, 2014). Note at this point that the collection of iterations needed to make one sweep through the training data set is called an *epoch*.

The extreme case of mini-batch gradient descent is when the batch size is selected to be 1. This is called *Stochastic Gradient Descent* (SGD). Recently SGD has been used much less, since it is more efficient to calculate the gradient in larger batches compared to only using one example. However, note that it remains common to use the term SGD when actually referring to mini-batch gradient descent. Gradient descent in general has often been regarded as slow or unreliable but it works well for optimising DNNs. SGD will most probably not find even a local minimum of the objective function. It typically however finds a very low value of the cost function quickly enough to be useful.

1.3.3 Optimisation Example

To illustrate the SGD algorithm, consider the linear model in a classification context. Suppose we are given a training data set with two-dimensional inputs and only two possible classes. Let the data be generated in the same way as described in (Hastie *et al.*, 2009, pp. 16-17).

> First we generated 10 means m_k from a bivariate Gaussian distribution $N((1, 0)^T, I)$ and labeled this class BLUE. Similarly, 10 more were drawn from $N((0, 1)^T, I)$ and labeled class ORANGE. Then for each class we generated 100 observations as follows: for each observation, we picked an m_k at random with probability 1/10, and then generated a $N(m_k, \frac{1}{5}I)$, thus leading to a mixture of Gaussian clusters for each class.

We want to fit a linear regression model to the data such that we can classify an observation to the class with the highest predicted score. In the binary case it is only necessary to model one class probability and then assign

an observation to that class if the score exceeds some threshold (usually 0.5), otherwise it is assigned to the other class. Therefore the decision boundary is given by $\{\boldsymbol{x} : \boldsymbol{x}^T\hat{\boldsymbol{w}} = 0.5\}$.

The example is illustrated in Figure 1.2. The colour shaded regions represent the parts of the input space classified to the respective classes, as determined by the decision boundary based upon OLS parameter estimates. Gradient descent was applied to the determine the optimal weights using a learning rate of 0.001. Since the total number of training observations are small, it is not necessary to use SGD. In Figure 1.2, the dashed lines represent the decision boundary defined by the gradient descent parameter estimates at different iterations. We observe that initially the estimated decision boundary is far from the OLS solution, but as the update iterations proceed, the decision boundary is rotated and translated until finally matching the OLS line. It took 29 iterations for the procedure to reach convergence. Note that here we chose when to stop based on the training loss, however in real modelling scenario we would determine when to stop by looking at the loss over some validation dataset.

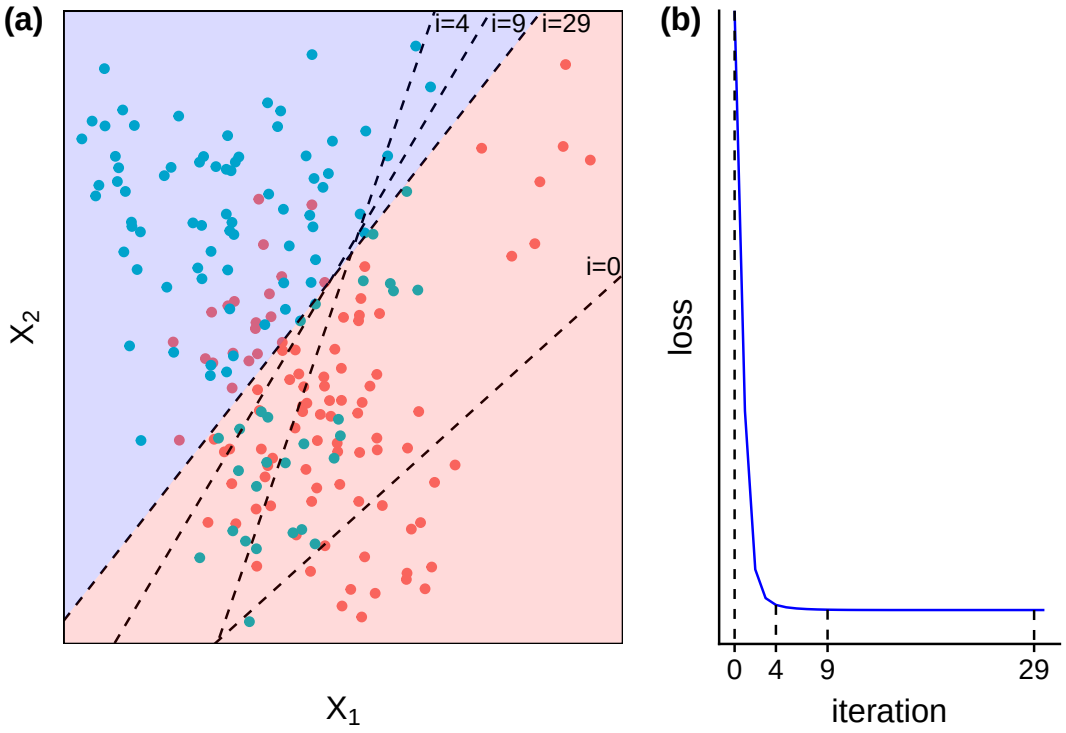

Figure 1.2: Plots of the gradient descent example. (a) The data points in input space. The shades in the background represent the class division in input space, with the decision boundary determined by linear least squares estimation. The dashed lines represent the decision boundaries learned at different iterations. (b) The loss calculated at each iteration.

Along the same lines as our discussion thus far, statistical learning using

DNNs (or *deep learning*) requires a model, loss function, optimisation procedure and of course, a data set. We have discussed all of these ingredients in simple and general terms, and are now ready to narrow our focus to deep learning methods. The following section on image classification introduces the application domain in this thesis.

1.3.4 Image Classification

We have introduced the statistical learning task of classification. As can be derived by its name, image classification refers to this task, but narrowed down to the case where the inputs are images, *i.e.* the task is to assign labels to images. As with classification in general, until recently, the main focus was classification of images associated with a single label. The more general case, and the focus of this thesis, where images can belong to more than one class simultaneously (for example images containing more than one object), is still relatively new. In this section we will mostly consider the single-label classification case, but most of the concepts are transferable to multi-label image classification. The novel concepts and challenges of multi-label image classification will be introduced in Chapter 3 and Chapter 4.

Before the era of deep learning, the main point of consideration for image classification was how to numerically represent images in such a way that an image classifier can distinguish between its classes, *i.e.* what features to extract from the images to be used as input. Before answering this question, we need a better understanging of the nature of images as inputs and how they can be represented in digital format.

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 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 1.3 illustrates this representation. 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 $32 \times 32 \times 3$.

A straightforward approach to transforming an array of pixel intensities to a compatible representation for conventional classifiers, is to *flatten* each array into a vector of size $width \times height \times channels$.

For example a hypothetical grayscale image of size $3 \times 3 \times 1$ is transformed as

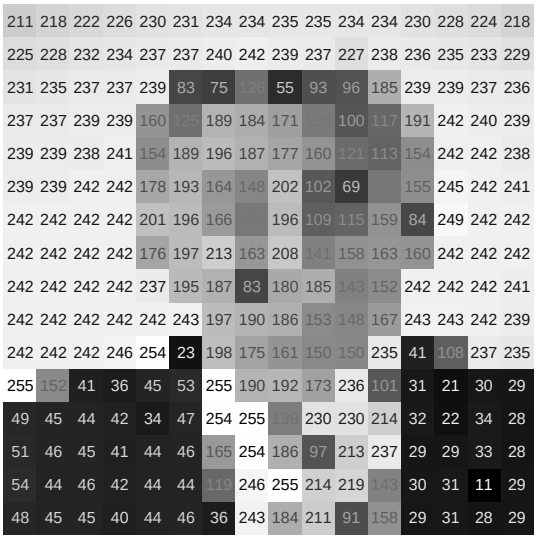

Figure 1.3: A low resolution grayscale profile of a man's head and shoulders, with pixel values overlayed onto the image.

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \end{bmatrix}.$$

Needless to say, the above representation is not very effective. Firstly, this transformation discards most of the shape and structural information of an image. In addition, flattened vectors are very high-dimensional ($p = 256 \times 256 \times 3 = 196608$ for a colour image of size 256×256). This abdicates the classifier to the *curse of dimensionality* (Hastie *et al.*, 2009, pp. 22-27), a well-known problem in statistical learning.

On top of that, in a typical image classification scenario, the images in its raw representation can have high intra-class variance and/or very low inter-class variance - rendering it extremely difficult for even state-of-the-art classifiers, such as Random Forests, Support Vector Machines or Boosting to discriminate between classes. Intra-class variation occurs within the same class of images, and typically stems from images of the same objects being taken from different viewpoints, using different scales, and with varying illumination and backgrounds, amongst others (see Figure 1.4). On the other hand, inter-class variation occurs between classes. Suppose the task is to distinguish between the image of an ostrich and the image of an emu. Here the inter-class variation may be very low, due to images of the two species appearing very similar.

Toward alleviating the above problem, one may attempt to extract features from images with lower intra-class variation and higher inter-class variation. Before we had the computational power to optimise DNNs, this was the standard procedure in image classification - designing feature representations of images that are much easier for conventional classifiers to learn from than

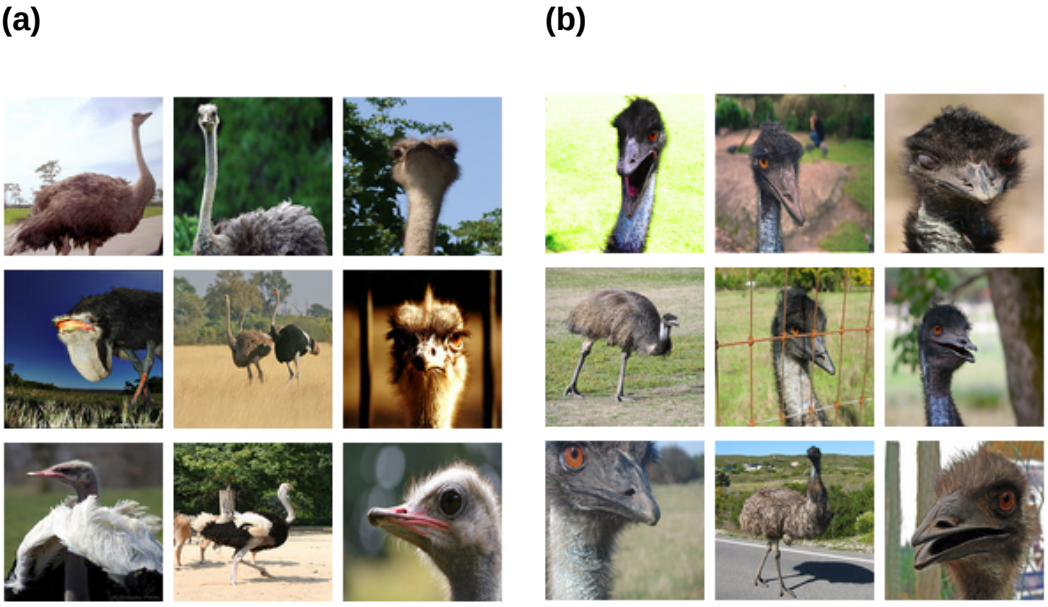

Figure 1.4: ImageNet examples of ostriches (a) and emus (b), illustrating high the intra-class variation and low inter-class variation. Notice the variation among images from the same class, and the similarity among images belonging to different classes.

raw pixel data. Constructing such feature representations can however become quite complex. Prior to deep learning, the most popular representation for classification was the Bag-of-Visual (BoV) words (Csurka *et al.*, 2004). Very briefly, the BoV approach entails the extraction of a set of local descriptors of images (see *Scale Invariant Feature Transform* (SIFT) (Lowe, 2004) and *Histogram of Oriented Gradients* (HOG) (Dalal and Triggs, 2005)), encoding them into a high-dimensional vector (see *Fisher Vectors* (Sánchez *et al.*, 2013)), and then pooling them into an image level signature (see *Spatial Pyramid Pooling* (Lazebnik *et al.*, 2006)).

As mentioned before, this approach to image classification is not trivial, with many complex facets to the modelling process. It usually requires considerable engineering skill and/or domain expertise. In addition, these feature representations are only suited for a narrow subset of image classification tasks and do not generalise well. This is where deep learning methods prove to be superior, since it has the ability to learn these feature representations from the data. This is discussed in detail in the next chapter. Hopefully now the reader has a better understanding of the image classification problem in general, which will most likely lead to a better appreciation of the power and effectiveness of deep learning methods.

1.4 Outline

From this point on the reader should have a basic understanding of the topic of this thesis. The reasons why this topic was chosen has been described in Section 1.1 and the goals of this thesis under Section 1.2. After briefly covering the fundamentals of statistical learning, the problem of image classification was introduced with the help of basic examples, along with a brief conceptual description of the multi-label image classification problem. Therefore, the reader should also be comfortable with the problem we want the DNNs to solve and the common challenges to such a problem. Although, DNNs have not yet been introduced, it should have been sufficient to know that it is class of algorithms. However, brief glimpses of the main components of DNNs have been given under the image classification problem description.

The rest of the thesis will follow the following outline. Chapter 2 introduces deep neural networks for image classifcation. Convolutional neural networks are specifically good at image classification and therefore most of the chapter will deal with them. The main focus is on the different structures and layers of the networks and how these influence performance (review this sentence - probably need to elaborate).

Chapter 3 starts with the fundamentals of multi-label classification. It includes a formal definition and emphasis on the important concepts, such as multi-label evaluation metrics and the modelling of label dependencies (maybe add class imbalance here). The chapter also includes a review of the state-of-the-art algorithms for MLC. A discussion is given on the challenges and ongoing research of the field. One of the objectives of this chapter is to find approaches that could help extend DNNs for multi-label image classification.

Chapter 4 reviews deep learning approaches for multi-label image classification. It looks at practically all of the proposals in the latest literature, how they attempt to model multiple labels and what are their shortcomings. The extensions are mostly adaptions of loss functions to be optimised or structural changes of the networks. An extensive comparision of the approaches is an important part of the chapter. Should I make proposals in this chapter?

Chapter 5 evaluates the highlighted approaches in previous chapter on popular benchmark image datasets. Note that more on the benchmark datasets are given in Appendix A. The goal is to give a comprehensive empirical comparison of the best representative approaches in terms of many multi-label evaluation metrics (with standard errors). It includes discussions on the challenges of implementing these approaches and how the results correlate with the literature. The experiments done in this chapter are made as reproducible as possible with additional information on the code and software given in Appendix B.

The thesis is concluded in Chapter 6 with a summary of the work done in this project, general discussion of the literature and results and what directions can be followed for future research. It includes a section on the limitations of

this study.

Chapter 2

Deep Learning and Image Classification

2.1 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, Section 2.2 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 2.5 discusses the important problem of reducing overfitting of DNNs. It reviews the most important strategies for improving the generalisation ability of DNNs. Section 2.6 on transfer learning. Maybe also something on attention and then a conclusion.

2.2 Deep Neural Networks

The main components of any supervised learning task are the data, objective function, model and optimisation procedure. In the previous chapter we have already discussed the type of data relevant for this work (images) and we have given trivial examples of objective functions, models and optimisation. This chapter zooms in on the class of models known as deep neural networks, specifically for images as input data and the relevant objective functions and optimisation procedures for this scenario.

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 strucute of this type of model which is most naturally visualised as a network of inter-connected nodes.

2.2.1 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 2.1.

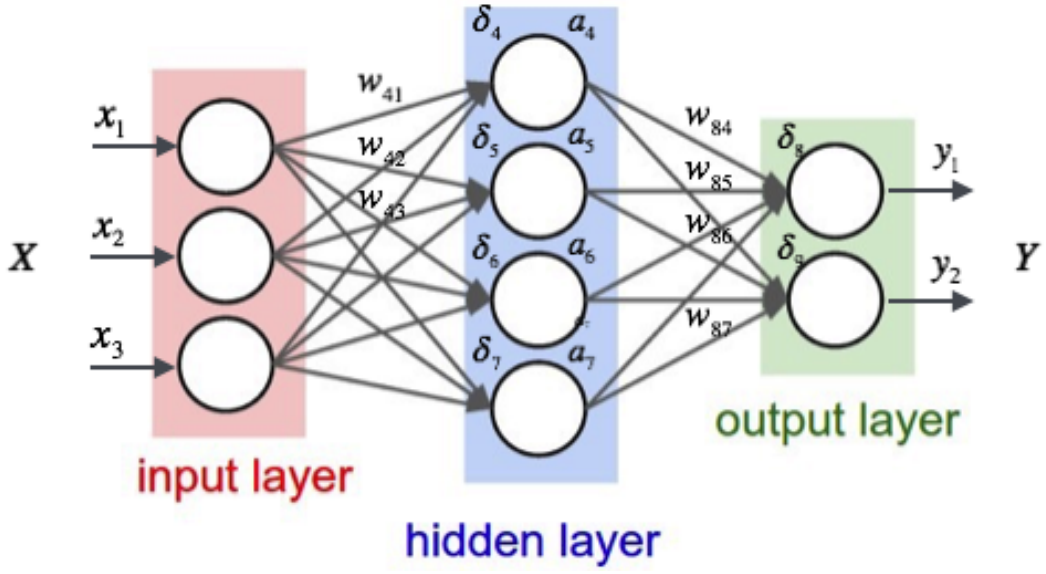

Figure 2.1: Graph structure of a vanilla neural network.

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*, $a(\cdot)$,

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

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 2.2.2.

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.

2.2.2 Activation Functions

In the previouse 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 rein-

forcement 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.

2.3 Training a Neural Network

2.3.1 Backpropogation

In Section 1.3.2 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.

thus far this section is too close to hastie book

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.

2.3.2 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) (not 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.

2.3.3 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 2.5.

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.

> sal n kort afdeling oor neurale netwerke vs die brein 'n goeie idee wees?

2.4 Convolutional Neural Networks

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.

2.5 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.

2.5.1 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

2.5.2 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

2.5.3 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 $\beta = \mu$.

2.6 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 3

Multi-Label Classification

3.1 Introduction

Multi-label Classification (MLC) belongs to the supervised learning paradigm and can be viewed as a generalisation of the conventional single-label classification problem. Suppose the data set to be analysed consists of a set of observations each representing a real-world object such as an image or a text document. In the single-label context each object is restricted to belong to a single, mutually exclusive class, *i.e.* each observation is associated with a single label. One can quite effortlessly come up with tasks that will not fit into this framework: an image annotation problem where each image contains more than one semantic object, a text classification task where each document has multiple topics or an acoustic classification task where the recordings contain the sounds of multiple bird species. Therefore the need for a multi-label classifier that can assign a set of labels to an observation.

The rapid growth of the MLC (see Figure 3.1) is probably owed to the vast and expanding range of MLC application domains, the biggest being text and multimedia categorisation especially those generated and/or stored on the web. Other application domains common to MLC are: biology, chemical data analysis, social network mining and E-learning amongst others. A thorough list of applications and their citations can be found in (Gibaja and Ventura, 2014). This thesis is on applying MLC in the image classification domain. However in this chapter we are interested in the general approach of MLC, regardless of the nature of the inputs.

This chapter serves as an overview of the MLC task. Since this framework is not that well known and contains some novelties compared to single label classification, it deserves some exploration. Here we are mainly interested in the formal defintion of MLC (Section 3.2), how MLC models are evaluated (Section 3.4) and some of the representative approaches to solve this problem (Section 3.5).

- mention challenges?

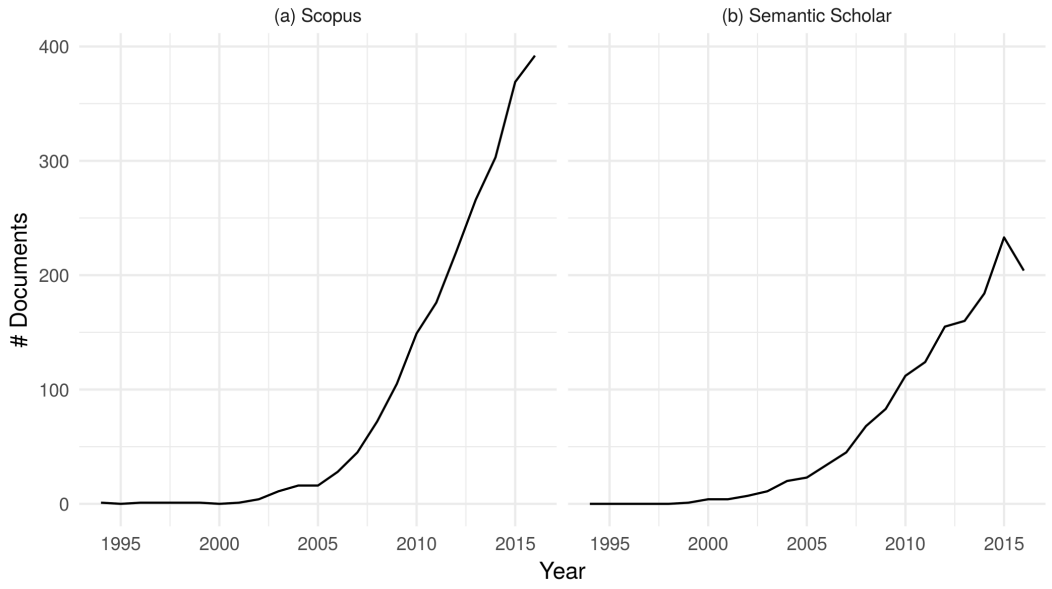

Figure 3.1: Line graphs illustrating the rise in multi-label learning publications per year for two databases. The database searches were done on 24-03-2017. The searches were not identical since they were limited to the search features of the databases. (a) The search on Scopus (cite) was for all documents (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.

3.2 Formal Defintion

Let $\mathcal{L} = \{l_1, l_2, \ldots, l_K\}$ denote the complete set of possible labels that can be assigned to an observation. Whereas a single-label classifier aims to find which single label l_k, $k = 1, 2, \ldots, K$, belongs to a given observation, a multi-label classifier is capable of assigning a set of labels $L \subseteq \mathcal{L}$ to the observation.

Recall from Section 1.3.1 that the label or output matrix can be given by (for multiple targets)

$$Y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1K} \\ y_{21} & y_{22} & \cdots & y_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N1} & y_{N2} & \cdots & y_{NK} \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_1^\intercal & \boldsymbol{y}_2^\intercal & \ldots & \boldsymbol{y}_K^\intercal \end{bmatrix} = \begin{bmatrix} \boldsymbol{Y}_{(1)} & \boldsymbol{Y}_{(2)} & \ldots & \boldsymbol{Y}_{(K)} \end{bmatrix},$$

where K is the size of the label set $\mathcal{L}$. Y only contains zeros and ones, *i.e.* $y_{ik} = 1$ if label l_k, $k = 1, \ldots, K$, is present for observation i and $y_{ik} = 0$

if it is absent. Thus $\boldsymbol{Y}_{(k)}$ is a N-dimensional binary vector indicating which observations are associated with label l_k. A multi-label dataset will be defined as $D = \begin{bmatrix} X & Y \end{bmatrix}$, which contains the N input-output pairs, $\{(\boldsymbol{x}_i, \boldsymbol{y}_i) \,|i = 1, \ldots, N\}$. Note that, $\boldsymbol{y}_i = (y_1, y_2, \ldots, y_K)$, $y_k \in \{0, 1\}$, used here is the label vector, however, it is also common to use the labelset notation, *i.e.* $L_i \subseteq \mathcal{L}$, where $\mathcal{L}$ is the complete label set and L_i is the set of relevant labels for observation i.

Often, the terms multi-label classification and multi-label learning are used interchangeably. Strictly speaking, multi-label learning is the umbrella term including both the tasks of multi-label classification and multi-label ranking. Both these tasks learn from multi-labelled data (multi-label learning) but their objectives differ slightly. Multi-label classification is concerned with approaches that output a subset of relevant labels (binary output) for each input, whereas multi-label ranking requires the output to be a list of all labels in the labelset ordered by their predicted relevance (real-valued output) to the input. There are minor differences between the two and therefore using one term to refer to both is usually acceptable. Many classifiers base their final (categorical) prediction on the thresholding of the real-valued output of the algorithm and thus can also be used for ranking. Similarly, ranking algorithms can also be used for classification if a thresholding function is applied to the real-valued output. We will use multi-label classification to refer to both classification and ranking, unless otherwise specified.

The task of ML classification is to find a function h that accurately maps the observations contained in X to the label matrix Y, i.e., $h : X \to Y$, so that given a new observation, h can determine which labels belong to it.

On the other hand, the goal of ML ranking is to find a function $f : X \to G$, where G is a similar matrix to Y, but with the g_{ij} a real value representing the relative confidence score that label j is relevant to observation i. f is found by optimising a ranking metric, also discussed shortly. From the confidence scores of observation i, $f(\boldsymbol{x}_i)$, a ranking $\boldsymbol{r}_i$ can be obtained, giving the rank of labels in descending order of $f(\boldsymbol{x}_i)$.

- threshold calibration?
- calibrate real-valued output against thresholding function output in order to determine labels of unseen instances.
- constant vs induced from training data + ad hoc specific to certain learning algorithms
- for Maximising F1-Score: https://arxiv.org/pdf/1402.1892.pdf
- mention the calibration factor of (Zhang and Zhou, 2014). Finding z_i from r_i

h will be referred to as the ML classifier and f as the ML ranker. When ML learner will be a collective term covering both h and f. Before different

ML learners can be discussed, an understanding of how the output of these algorithms are evaluated is necessary, since fitting f of h envolves optimising an evaluation metric. (always?)

3.3 Label Correlation and other Challenges

The key challenge in MLC is to exploit dependecies amongst labels, *e.g.* using the information on the presence/absence of label l_i to predict label l_j, $i, j \in \{1, 2, \ldots, K\}$, $i \neq j$. This becomes especially difficult for a multi-label classifier when dealing with large labelset. It is not uncommon for multi-labelled datasets to have hundreds of thousands of labels. Proof of this can be found at The Extreme Classification Repository[1] or the dataset of the recent YouTube Video Classification Challenge (Abu-El-Haija *et al.*, 2016). Algorithms that can accurately and efficiently model label dependence on these datasets are scarce (Sorower, 2010). This is a focus area of recent MLC research, called extreme multi-label learning (Xu *et al.*, 2016). A more formal definition of label dependence will be given later on. An in-depth discussion on the unique challenges (thorough list by (Gibaja and Ventura, 2014)) that arise from dealing with label dependence and some of the possible strategies to follow will also be covered.

In (Zhang and Zhou, 2014) the existing strategies for multi-label classification are divided into categories based on the order of label correlations being considered by the algorithms. So-called first-order approaches are those that do not take label correlations into account. Second-order approaches consider the pairwise relationships between labels and high-order approaches allows for all interactions between labels and/or combinations of labels. First-order strategies simply ignore label correlations, but they are usally simpler. The latter two strategies are far more complex but also limited in some cases. Second-order strategies will not generalise well when higher-order dependencies exist amongst the labels and the the high-order strategies may 'overfit' if only subgroups of the labels are correlated (Zhang and Zhou, 2014).

From the Bayesian point of view, the problem of multi-label learning can be reduced to modeling the conditional joint distribution of $P(\boldsymbol{y}|\boldsymbol{x})$. This can be done in various ways. First-order approaches solve the problem by decomposing it into a number of independent task through modelling $P(y_k|\boldsymbol{x})$, $k = 1, \ldots, K$. Second-order approaches solve the problem by considering interactions between a pair of labels through modelling $P((y_k, y_{k'})|\boldsymbol{x})$, $k \neq k'$. High-order approaches solve the problem by adressing correlations between a subset of labels through modelling $P((y_{k_1}, y_{k_2}, \ldots, y_{k_{K'}})|\boldsymbol{x})$, $K' \leq K$. Our goal is to find a simple and efficient way to improve the performance of multi-label learning by exploiting the label dependencies (Zhang and Zhou, 2014). Propose LEAD approach.

[1] https://manikvarma.github.io/downloads/XC/XMLRepository.html

- [Tsoumakasf] use the ϕ coefficient to estimate label correlations.
- (?)
- mention the holy grail comment
- comment on what 'exploitation' means. Since many authors claim that exploiting label dependence structures is the only way to effectively handle multiple labels, I would assume this means that we can make use of label correlations to spare time and increase accuracy.
- we need to think about how observations are labelled, when will it be useful to take label dependence into account and how.
- Such a solution, however, ne- glects the fact that information of one label may be help- ful for the learning of another related label; especially when some labels have insufficient training examples, the label correlations may provide helpful extra information (Huang *et al.*, 2012)
- minimisation of surrogate loss functions and consistency
- Consistency [Zhou2011]:

They were the first to do a theoretical study on the consistency of multi-label learning algorithms, focusing on the ranking loss and the hamming loss. A learning algorithm is said to be consistent if its expected risk converges to the Bayes risk as the size of the training data increases. They found that any convex surrogate loss is inconsistent with the ranking loss and therefore proposed a partial ranking loss (which is consistent with some surrogate loss functions) as an alternative. They also show how some recent multi-label algorithms are inconsistent in terms of the hamming loss and provides a discussion on the consistency of approaches which transforms the multi-label problem into a set of binary classification tasks.

- more theoretical work at (Gasse *et al.*, 2015). Mentions: Finding theoretically correct algorithms for other non label-wise decomposable loss functions is still a great challenge.
- more theory: Optimizing the F-Measure in Multi-Label Classification: Plug-in Rule Approach versus Structured Loss Minimization

Other solutions: exploit correlation of labels from both types conditional and unconditional dependencies, features selection methods that are designed especially to handle multi label datasets, and having new stratification methods that are suitable to the nature of multi label datasets (copied from (Alazaidah and Ahmad, 2016))

- Symmetry:
- (Huang *et al.*, 2012) claims that most of the time the label dependencies are asymmetric and suggest the MAHR algorithm. Also most of the existing methods exploit label correlations globally, which is not necessarily a good assumption if these correlations only exist for some instances (Huang *et al.*, 2012). They suggest a ML-LOC algorithm (which seems to do very well).
- Locality
- is local the same as conditional? and global unconditional?
- (Zhu *et al.*, 2017*b*)

Existing approaches to exploiting label correlations either assume the the label correlations are global and shared by all instances, or that the label correlations are local and shared only by a subset of the data. It may be that some label correlations are globally applicable and some share only in a local group of observations.

- give example
- mention GLOCAL (Zhu *et al.*, 2017*b*)
- (Huang *et al.*, 2012)

Existing approaches typically exploit label correlations globally by assuming that the label correlations are shared by all observations. In the real-world, however, different observations may share different label correlations and few correlations are globally applicable.

- propose ML-LOC approach
- mentions that by assuming global correlations may be hurtful to the performance (Huang *et al.*, 2012) in empirical discussion

3.3.0.1 Class Imbalance

- https://www.reddit.com/r/MachineLearning/comments/6iq5i8/d_what_are_your_favorite_ways_for_dealing_with/
- (Charte *et al.*, 2015)
- towards class imbalance aware multi label learning
- way of stratifying batches: https://arxiv.org/pdf/1705.00607.pdf

3.4 Evaluation of Multi-Label Classifiers

There is a plethora of metrics to evaluate the performance of multi-label classifiers. They can be divided into two groups: example-based and label-based measures. Example-based measures compare the actual labels versus the predicted labels for each **observation** and then computes the average across all the observations in the dataset. Label-based measures evaluates the predictions on a per **label** basis and then averages across all labels. Zhang and Zhou (2014) provides the full taxonomy.

The most common measures in MLIC are the micro and macro precision, recall and F_1-measure, as well as the mean average precision (mAP). These metrics are more robust to class imbalance. It is useful to report the performance of a multi-label classifier using multiple and contrasting measures. An improvement in one mertric does not guarrantee an improvement in another. For example, the optimal label calibration for the micro F_1-score will not be the same for the macro F_1-score.

(**?**) first to categorise into label-based and example-based.

The evaluation of the performance of ML algorithms is another distinct problem to this setting. Compared to the single-label case, many more evaluation metrics exist, with subtle or obvious differences in their measurement. According to (Madjarov *et al.*, 2012) it is essential to evaluate a ML algorithm on multiple and contrasting measures because of the additional degrees of freedom introduced by the ML setting. In addition, care should be taken when reporting multiple measures and with their interpretation. Since some of the measures are contrasting it is dangerous to report multiple metrics and conclude that on average one learner is better than the other. This was highlighted in (Dembcz *et al.*, 2012), where the authors suggested that when evaluating the performance of a ML learner, it should be made clear which metric(s) it is aiming to optimise, otherwise the results can be misleading. It is impossible (?) for a learner to have superior performance over others in terms of all the multi-label evaluation metrics simultaneously.

The evaluation measures of predictive performance of multi-label learnerss can be divided into two groups: example-based and label-based measures. Example-based measures compares the actual versus the predicted labels for each observation and then computes the average across all the observations in the dataset. Where label-based measures computes the predictive performance on each label separately and then averages across all labels (Madjarov *et al.*, 2012). For both groups the measures can further be partitioned into metrics from a classification persepective and measures from a ranking perspective, *i.e.* metrics for h and metrics for f respectively. The most commonly used metrics in each of the groups will be introduced here.

- Figure 3.2 is just an example. The image quality is lacking.

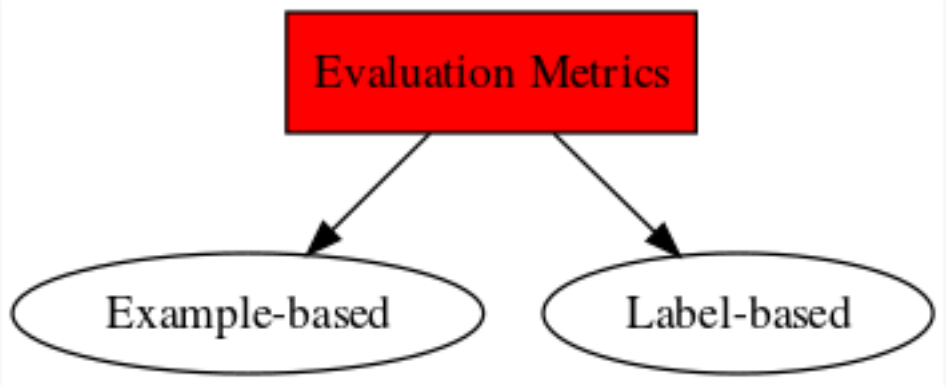

Figure 3.2: Categorisation of the taxonomy of MLL evaluation metrics

3.4.1 Example-based Metrics

For the following definitions, let y_i be the set of true labels for observation $\boldsymbol{x}_i$ and z_i the set of predicted labels for the same observation, obtained from the predicted indicator vector of $\hat{h}(\boldsymbol{x}_i)$. The Hamming loss is then defined as

$$\text{hloss}(h) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{K}|z_i \triangle y_i|,$$

where $\triangle$ stands for the symmetric difference and $|.|$, the size of the set. For example, $|\{1,2,3\}\triangle\{3,4\}| = |\{1,2,4\}| = 3$. Thus the Hamming loss counts the number of labels not in the intersection of the predicted subset of labels and the true subset of labels, as a fraction of the total size of the labelset, averaged across each observation in the dataset. When h returns perfect predictions for each observation in the dataset, $\text{hloss}(h) = 0$, and if h predicts for each observation that it belongs to all the labels except for its the true labels, $\text{hloss}(h) = 1$.

Accuracy is defined as

$$\text{accuracy}(h) = \frac{1}{n}\sum_{i=1}^{n}\frac{|z_i \cap y_i|}{|z_i \cup y_i|}.$$

Thus for each observation the number of correctly predicted labels is calculated as a proportion of the sum of the correctly and incorrectly predicted labels. These quantities are then averaged over each observation in the dataset. If the h perfectly predicts the relevant subset of labels for each observations, $\text{accuracy}(h) = 1$. If h does not manage to predict a single correct label for any observation, $\text{accuracy}(h) = 0$.

The precision and recall are respectively defined as

$$\text{precision}(h) = \frac{1}{n}\sum_{i=1}^{n}\frac{|z_i \cap y_i|}{|z_i|},$$

and

$$\text{recall}(h) = \frac{1}{n}\sum_{i=1}^{n}\frac{|z_i \cap y_i|}{|y_i|}.$$

Precision calculates the average proportion of correctly predicted labels in terms of the number of labels predicted, across all the observations in the dataset. Recall calculates a similar average, with the only difference that the proportion is calculated in terms of the number of true labels per observation. Both these metrics lie in the range $[0, 1]$ with larger values desirable.

The harmonic mean between the precision and the recall is called the F_1-score and is defined as

$$F_1 = \frac{1}{n}\sum_{i=1}^{n}\frac{2|z_i \cap y_i|}{|z_i| + |y_i|}.$$

The perfect classifier will result in a F_1-score of 1 and the worst possible score is zero.

The subset accuracy or classification accuracy is defined as

$$\text{subsetacc}(h) = \frac{1}{n}\sum_{i=1}^{n} I(z_i = y_i),$$

where $I(.)$ is the indicator function. This the subset accuracy is the proportion of observations that were perfectly predicted by h.

The above are all performance measures of ML classifiers. If the ML learner outputs real-valued confidence scores, these ranking metrics can be used to evaluate the learner's performance:

One-error:

Coverage:

Ranking Loss:

Average Precision:

3.4.2 Label-based Metrics

The idea with label-based measures is to compute a single-lable metric for each label based on the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) made by the classifier on a dataset and then obtaining an average of the values (Gibaja and Ventura, 2014). Note, TN_k, TP_k, FN_k and FP_k denote the quantities for label l_k, $k = 1, 2, \ldots, K$. Thus $TP_k + TN_k + FP_k + FN_k = n$. Let B be any binary classification metric, i.e. $B \in \{\text{accuracy}, \text{precision}, \text{recall}, F_1\}$. B can be written in terms of TN_k, TP_k, FN_k and FP_k, for example

$$\text{accuracy}(TN_k, TP_k, FN_k, FP_k) = \frac{TP_k + TN_k}{TP_k + TN_k + FP_k + FN_k}.$$

B is then calculated for each label and then an average is calulated. The averaging can be done either by the micro or the macro approach. The micro approach considers predictions of all observations together and then calculates the measure across all labels, i.e.

$$B_{micro} = B\left(\sum_{k=1}^{K} TP_k, \sum_{k=1}^{K} TN_k, \sum_{k=1}^{K} FP_k, \sum_{k=1}^{K} FN_k\right).$$

Whereas the macro approach computes one metric for each label and then the values are averaged over all the labels, i.e.

$$B_{macro} = \frac{1}{K}\sum_{k=1}^{K} B(TP_k, TN_k, FP_k, FN_k).$$

Note, also that $\text{accuracy}_{micro}(h) = \text{accuracy}_{macro}(h)$ and that $\text{accuracy}_{micro}(h) + \text{hloss}(h) = 1$, since Hamming loss is the average binary classification error.

Again, all of the above mentioned metrics are from a classification perspective. An example of a label-based metric from a ranking persepective is the macro- and micro-averaged AUC:

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). It might be that a algorithm does very well in terms of the Hamming loss, but performs poorly according to the subset accuracy, or vice versa, as shown in (Dembcz *et al.*, 2012). In (**?**) they claim that the Hamming loss reported together with the micro-average F-measure gives a good indication of the performance of a multi-label classifier.

These multi-label metrics are usually non-convex and discontinuous (Zhang and Zhou, 2014). Therefore multi-label classifiers resort to considering surrogate metrics which are easier to optimise.

Other than predictive performance, are there other aspects on which multi-label classifiers can be evaluated, such as efficiency and consistency. Multi-label algorithms should be efficient in the sense that it takes 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 [reference]. The other desirable attribute of multi-label classifiers are that they are consistent. This means that the expected loss of the classifier converges to the Bayes loss when the number of observations in the training set tends to infinity. Actually only a very few number of multi-label classifiers satisfy this property [Zhou2011], (Koyejo *et al.*, 2015).

3.4.3 Partitioning Datasets

3.4.4 Complexity and Statistical Tests

3.5 Methods

Approaches to MLC can be divided into two groups: Problem Transformation (PT) and Algorithm Adaption (AA). PT algorithms are any method that transforms the MLC problem into one or more binary or multiclass classification problems. The simplest example of such an algorithm is the binary relevance (BR) method. BR transforms the MLC task into K binary classification tasks where K is the number of unique labels in the dataset. By training a classifier for each label separately, the BR approach cannot exploit label correlations.

One solution to this may be to sequentially train a classifier to predict a label and then in the next iteration to use that label's predicted values as additional input to the classifiers trained to predict the rest of the labels. Now some interaction between labels are considered in the modelling process. This approach is called Classifier Chains (CC). The challenge here is to determine in which order to predict the labels. For example, if label i "depends" on label j, we would first want to train a model to predict label j and then use those predicted values as additional input for the classifier trained to predict label i.

For both BR and CC we need to train at least[2] K classifiers. The greater K becomes the less feasible it is to use DNNs as the base model. The final PT example we will give is the label powerset (LP) algorithm. The LP algorithm transforms the MLIC task to a multiclass classification task by treating each unique combination of labels as a new class. This results in a possible 2^K number of classes and the disadvantage here is that some combinations will be extremely rare, making it hard for a classifier to learn. The LP method is capable of exploiting label correlations. Note that the use of ensembling is also common to improve on these methods. Sometimes they are assigned to their own group of multi-label classifiers, Ensemble Methods. The popular algorithms here are the ensemble of CC's (ECC) and the Random k labelsets (Rakel).

The other group of multi-label classifiers are the AA methods. These are any method for which a single label classifier was internally adapted to suit MLC. For example the multi-label k-nearest neighbours (ML-kNN) (Zhang and Zhou, 2007) approach is just an adaption of the the conventional kNN algorithm so that it can handle multiple labels per observation.

[2]There is no limit to how long the chain of a CC algorithm should be. After predicting all the labels, one can start from the beginning again, for which we would then have predicted scores for all labels which can be treated as input.

3.5.1 Problem Transformation Approaches

There are numerous multi-label learning algorithms. It is difficult to keep up with the all the latest proposed methods. These algorithm can be categorised in a number of ways, *e.g.* the review (Zhang and Zhou, 2014) and the tutorials (Gibaja and Ventura, 2015) and (Carvalho, André C P L F de, 2009), all have different ways of grouping the algorithms. The categorisation for this thesis is chosen to satisfy the criteria of being common, simple and intuitive. Nevertheless, the characteristics of the algorithms leading to the other grouping variants will still be given in the remarks of the algorithms.

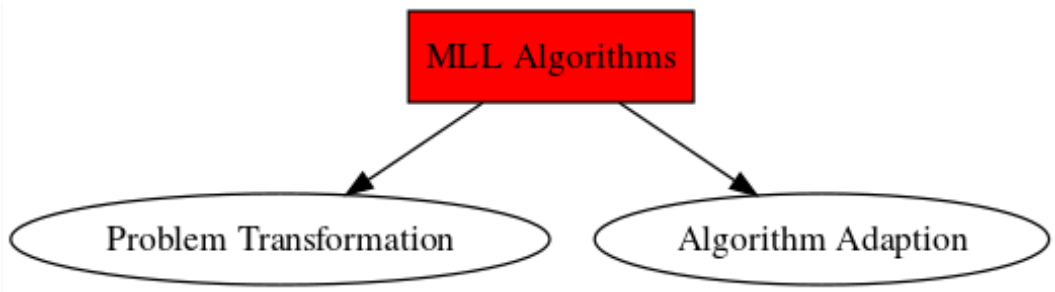

Figure 3.3: Categorisation of multi-label learning taxonomy (this is just an example)

- still need to edit Figure 3.3
- want to keep it simple and representative but also give table with full list of methods
- many proposals
- scrutinise 8 representative algorithms for feasibility concerns
- representativeness criteria: broad spectrum; primitive impact; favourable impact
- introduce PT vs AA
- diagram of categorisation
- very thorough one in (Gibaja and Ventura, 2015)
- mention ensemble category

Problem transformation 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, problem transformation methods are called algorithm independent, i.e. once the data is transformed, any single-label classifier can be used (**?**).

The two main problem transformation algorithms are the binray relevance and label powerset transformations. Both methods suffer from several limitations but they form the basis of arguably any problem transformation method. The state-of-the-art problem transformations algorithms are most of the times extensions of either the standard binary relevance or label powerset algorithms (Alazaidah and Ahmad, 2016). Therefore the understanding of these two basic methods are crucial in dealing with the more complex, modern problem transformation methods.

3.5.2 Binary Relevance

- remarks: first-order; parallel; straightforward; building block of state-of-the-art; ignores potential label correlations; may suffer from class-imbalance; computational complexity

The most common transformation method is binary relevance (BR). BR transforms the mutli-label into K single-label problems by modelling the presence of the labels separately. Typically K single-label binary data sets, $D_k = (X, \boldsymbol{Y}_k)$ for $k = 1, ..., K$, would be constructed from the multi-label data set, $D = (X, Y)$. To each D_k any single-label classifier can be applied. In the end, predictions $\hat{\boldsymbol{Y}}_1, ..., \hat{\boldsymbol{Y}}_K$ are obtained separately which can then be combined to allocate all the predicted relevant variables to each instance. Note, that it may occur that all of the single-label learners produces zeroes, which would imply that the instance belongs to an empty set. To avoid this (Zhang and Zhou, 2014) suggests following the T-criterion rule. The rule states, briefly, that in such a case the labels associated with the greatest output should be assigned to the instance. Clearly, this will only work if the base learners used gives continuous outputs and it will only make sense if all the base learners are of the same type. I suppose these rules are ad-hoc and I can think of alternatives.

The biggest drawback for this approach is that it models each label separately and ignores the possible correlations between labels. Thus BR assumes that there are no correlations between the labels. However, these correlations can be very helpful in predicting the labels present. This is a first-order strategy. Also it can be time consuming since data sets with hundreds of labels is not rare. This would mean more than a hundred models should be fit and tuned separately. But this complexity scales linearly with increasing K, which is actually not so bad when comparing to other multi-label algorithms. Grouping the labels in a hierachical tree fashion may become useful when K is very large [Cherman2011] (see also Incorporating label dependency into the binary relevance framework for multi-label classification by the same authors).

Another argument against BR from (**?**): The argument is that, due to this information loss, BR's predicted label sets are likely to contain either too many or too few labels, or labels that would never co-occur in practice.

Advantage of BR by (**?**): Its assumption of label independence makes it suited to contexts where new examples may not necessarily be relevant to any known labels or where label relationships may change over the test data; even the label set L may be altered dynamically - making BR ideal for active learning and data stream scenarios.

Nevertheless, BR remains a competitive ML algorithm in terms of efficiency and efficacy, especially when minimising a macro-average loss function is the goal (Luaces *et al.*). The most important advantage of BR is that it is able to optimise several loss functions (Luaces *et al.*) also see small proof. They also show empirically that BR tends to outperform ECC when there are many labels, high label dependency and high cardinality, i.e. when the multi-label data becomes more complicated.

Compared to label powerset (LP) which will be discussed later, BR is able to predict arbitrary combinations of labels (Tsoumakas *et al.*, 2009) not restricted only to those in the training set.

[Cherman2011] also proposes a variation of BR called BR+. Its aim is to keep the simplicity of BR but also to consider the possible label correlations. It does so by also creating K binary data sets but this time each of these data sets treat all the label columns not to be predicted by the current single-label classifier as features to the classifier. Thus each sinlge-label classifier will have $p + K - 1$ inputs. So now when predicting label l, all of the original features in X and the remaining variables $\boldsymbol{Y}_k$, $k \neq l$, are used as inputs for classifier l. (second order strategy?)

The problem arises when predicting unseen instances for which the labels are unknown. Thus the input needed for each binary classifier is not available. One workaround is to obtain an initial prediction of the labels using an ordinary BR approach and then using these predictions as inputs to the BR+ algorithm. The BR+ algortihm will most likely produce different predictions to the initial predicitons or BR which can then also be used in a next round of BR+. These steps can be continued until convergence but this seems like the classifier chains approach. (to be investigated).

(Tsoumakas *et al.*, 2009) mentions the 2BR strategy that seems very similar/identical to BR+. They describe the 2BR method as follows: first train a binary classifier on each of the K binary data sets and then use their predictions (and or probabilities) as so called meta-features for a second round of BR. They mention that it might be better to train the base and meta learners on separate parts of the training data to avoid biased predictions. They suggest using a cross-validation approach for both learners to also avoid size constraints of the training data. They describe this approach as a stacked generalisation, also mentioned in (**?**), (Godbole and Sarawagi, 2004), (Pachet and Roy, 2009) calls it classifier fusion.

The adding of all the base learner predicitions as meta-feature to the meta-learners is not necessarily desirable. Some label pairs might have no correlation and adding predictions for those labels as inputs to the meta-learner will add noise to the model and waste computation time. (Tsoumakas *et al.*, 2009) suggests a solution called corerlation-based pruning. They calculate the pairwise correlations between labels, ϕ, and only add base learner prediction of label i as a meta-feature to meta-learner j if ϕ_{ij} is greater than some threshold. In this way only label-pairs that are highly correlated will be used in the final prediction of each other.

- BR performs well for Hamming loss, but fails for subset 0/1 loss.
- It is not clear, in general, whether the meta-classifier b should be trained on the BR predictions h(x) alone or use the original features x as additional inputs. Another question concerns the type of information provided by the BR predictions. One can use binary predictions, but also values of scoring functions or probabilities, if such outputs are delivered by the classifier (Dembcz *et al.*, 2012).

3.5.3 Label Powerset

The other widely known problem transformation approach is the label powerset (LP) algorithm. Each combination of the labels is seen as a distinct class and then a standard multiclass classification learner can be applied. More formally, the transformation $h : L \rightarrow P(L)$ is applied (**?**). Thus label correlations are taken into account but LP has other limitations. The number of possible classes increase exponentially with the increase in K and some of the classes/combinations are under-represented (if represented at all) in the training set. This leads to the difficult problem of learning from unbalanced classes and also restricts the algortihm to only predict combinations of labels present in the training set. Labels (or labelsets) that only occur a limited number of times are called tail labels. These are generally the ones difficult to model and a classifier can easily neglect their importance (Xu *et al.*, 2016).

One way to reduce the number of resulting classes after a label powerset transformation is to create meta-labels (not to be confused with meta in the stacking sense) (**?**). Meta-labels represent partitions of the label set, but I still do not fully understand the concept. Seems like after the transformation we still end up with a multi-label problem. Investigate further.

Another option is to throw away the combinations that appear infrequently in the training set. This obviously limits the possible output of the mutli-label algorithm even more. Sounds like PPT (**?**).

LP takes conditional dependence into account but usually fails for losses like Hamming (Dembcz *et al.*, 2012). Can improve with RAKEL, but it is still not well understood from a theoretical point of view.

3.5.4 Classifier Chains

- importance of ordering
- remarks: high-order; considers label correlations in a random manner; not parallel; computational complexity

Another extension of BR, similar to 2BR and BR+, is the classifier chains (CC) approach introduced by (**?**). It also consists of transforming the mutli-label data set D to K single-label data sets but the transformations are done sequentially in the sense that the label previously treated as a response will be added as a feature for predicting the next label. This will give data sets similar to $D_1 = (X, \boldsymbol{Y}_1), D_2 = (X, \boldsymbol{Y}_1, \boldsymbol{Y}_2), ...D_K = (X, \boldsymbol{Y}_1, \boldsymbol{Y}_2, ..., \boldsymbol{Y}_K)$, where the last column of each is the response that needs to be predicted. To each of these single-label data sets a classifier can be trained and then their predictions are combined in the same fashion as BR. CC keeps the simplicity of BR but has that additional capacity to model label dependencies by passing label information between classifiers. This should raise the question of what order of labels should the chain consist of and should it stop after one cycle?

Paper still need to look at for CC (Sucar *et al.*, 2013).

3.5.5 Algorithm Adaption Approaches

- NNs?

These are methods tackling the multi-label learning task by adapting, extending and/or customising an existing supervised learning algorithm (Madjarov *et al.*, 2012).

The main weakness of algorithm adaption 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) (algorithm independent).

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

- basic idea
- procedure
- psuedo-code
- remarks: first-order; merits of lazy learning and Bayesian reasoning; mitigate class-imbalance; extensions/variations; computational complexity

3.5.7 Multi-Label Decision Tree (ML-DT)

- basic idea
- procedure
- psuedo-code

- remarks: first-oders; efficient; improve with pruning and or ensembling; computational complexity

3.5.8 Ranking Support Vector Machine (Rank-SVM)

- basic idea
- procedure
- psuedo-code
- remarks: second-order; variants; computational complexity

3.5.9 Collective Multi-Label Classifier (CML)

- basic idea
- procedure
- psuedo-code
- remarks: second-order; conditional random field model; DAG; computational complexity

3.6 Ensemble Approaches

- Ensembles are well known for their effect of increasing overall accuracy and overcoming over-fitting, as well as allowing parallelism. The main idea behind ensembles is to exploit the fact that different classifiers may do well in different aspects of the learning task so combining them could improve overall performance. Ensembles have been extensively used in literature [13] with stacking [14], bagging [15] and boosting [16] being the main methods employed. In the context of multi-label problems, [17] proposes a fusion method where the probabilistic outputs of heterogeneous classifiers are averaged and the labels above a threshold are chosen. Copied from [Papanikolaou] (can maybe use to explain why these methods perform better and not because of label dependence)
- evidence of stacking working [Tsoumakase]. Read conclusions chapter. Ensembling effective. Linear models good for text classification. Thresholding important.

3.6.1 Ensemble of Classifier Chains

In a response to this (referring to CC), the ensembles of classifier chains (ECC) was suggested by (**?**). Here the term ensemble refers to an ensemble of multi-label classifiers instead of an ensemble of binary classifiers already mentioned before. ECC trains m classifier chains, each with a random chain ordering and a random subset of instances. These parameters of ECC contributes to the uniqueness of each classifier chain which helps with variance reduction when

their predictions are combined. These predictions are summed by label so that each label receives a number of votes. A threshold is used to select the most popular labels which form the final predicted multi-label set (**?**) (copied from). More details still to cover in article.

CC and ECC has an advantage over the ensemble methods of BR, that it is not necessary for an initial step of training to obtain predictions of labels that can later be used as features, it does this simultaneously.

3.6.2 Random k-Labelsets

As mentioned before, the LP method has the advantage of taking label correlations into account but typically suffers from a huge class imbalance problem. (**?**) suggested the Random k-labelsets (RAKEL) algorithm to overcome the drawbacks of LP while still being able to model label dependencies. RAKEL is simply an ensemble of LP classifiers, but the LP classifiers are trained on different subsets of the labelset. The author defined a k-labelset as a set $Y \subseteq L$ with $k = |Y|$, where L is the complete labelset and $|Y|$ the size of the set, Y. Let L^k denote the set of all distinct k-labelsets on L. The size of L^k can thus be given by $|L^k| = \binom{|L|}{k}$.

First, the RAKEL algorithm iteratively constructs m LP classifiers. At each iteration, $j = 1, 2, \ldots, m$, it randomly selects a k-labelset, Y_j, from L^k without replacement, and then learns the classifier $h_j : X \to P(Y_j)$ (review notation). For classifying an instance, x, each model, h_j, provides binary decisions, $h_j(x, \lambda_l)$ for each label λ_l in k-labelset Y_j. The average of these binary decisions are then computed and a final prediction for a label is given if its corresponding average is bigger than some threshold t. Note, the average for label λ_l is not calculated by the sum of $h_j(x, \lambda_l)$ divided by m, but by instead dividing by the number of times λ_l was in Y_j for $j = 1, \ldots, m$.

The values m, k and t, are all parameters to be specified by the user. Clearly, k can only lie between 1 and $|L|$, where if $k = 1$, the algorithm is equivalent to the BR approach, and if $k = |Y|$, the algorithm is equivalent to the LP approach. In the original paper, the author showed empirically that by using small labelsets and an adequate number of iterations, RAKEL will manage to model label correlations effectively. An intuitive value for t would be 0.5, however, in the same paper, it is shown that RAKEL performs well over a wide range of values for t.

A concern might be the number of classes, 2^k that each LP classifiers must deal with. In practice, each LP classifier deals with a much smaller subset of label combinations, since it can only model combinations that exist in the training set. Also, RAKEL is preferred to LP when there are a large number of labels. In this case, RAKEL would only need to model a subset of 2^k possible label combinations compared to LP that needs to model a much larger subset of $2^{|Y|}$ possible label combinations.

In (**?**) it is shown that RAKEL outperforms LP and BR on 3 benchmark datasets with numerous configurations. The author concluded that the randomness of the RAKEL algorithm might not be the best ensemble selection approach since it may lead to the inclusion of models that affect the ensemble's performance in a negative way. Continue with papers that improve on this idea.

There are other ways of choosing subsets of the labelset, references in (**?**).

Note, with all these ensemble extensions, we can still try different ways of ensembling/stacking, especially with RAkEL. Not only taking the average but also by assigning weights to each model or by fitting a model to the predictions. Think (Lo *et al.*, 2013) is an example of this with generalised k-labelsets ensemble.

- LP takes the label dependence into account, but the conditional one: it is well-tailored for the subset 0/1 loss, but fails for the Hamming loss.
- LP may gain from the expansion of the feature or hypothesis space.
- One can easily tailor LP for solving the Hamming loss minimization problem, by marginalization of the joint probability distribution that is a by-product of this classifier.

3.6.3 Thresholding Strategies

The CNN outputs a set of class score which minimises the average of the binary cross-entropy over each labels. Therefore a mapping is needed to transform the class scores to binary outputs, indicating label presence. This is an important facet of MLC, often overlooked, but can make a huge difference in performance for certain metrics. This is similar to the problem in single label classification where the classification threshold can be adjusted to optmise either precision or recall instead of accuracy, which is especially important for imbalanced data.

In MLC threshold calibration is also a common technique to go from the class score to binary outputs. If the class scores mimics class probabilities, a threshold of 0.5 is a common and intuitive choice, *i.e.* all labels with scores higher than 0.5 are labeled with a 1 and the rest as zero. However, this may not be the optimial threshold for certain metrics. For example, a lower threshold (lower than 0.5) will most likely result in a better recall score. Determining this optimal threshold for certain metrics can become quite complicated.

A relatively simple method is to test multiple thresholds and evaluate the selection's performance on a left out validation set. Naturally this method also extends to a cross-validation approach. This becomes more complicated when label dependent thresholds are used, *i.e.* a different threshold for each label. Jointly determining these multiple thresholds through the validation approach is hard since there are many possible combinations to be tested in which case users normally resort to optimising each label threshold separately. This becomes less accurate for example based metrics.

[http://www.cs.waikato.ac.nz/~eibe/pubs/chains.pdf] suggests an alternative approach to determining thresholds, which is to choose a single threshold such that the label cardinality of the test set is as close as possible to that of the training set. Obviously this is only possible when a complete test set is available at test time. There is no need for heavy validation testing with this approach. This supposedly works well for optimising accuracy and the F-measure, given the assumption that the class distribution of the test set is similar to that of the training set. Of course other multi-label data characteristics can be used instead of cardinality, depending on the problem.

Another approach is to view the threshold selection as a learning problem [http://machinelearning.wustl.edu/mlpapers/paper_files/nips02-AA45.pdf]. For example using a linear model taking the class scores as input and outputs a threshold minimising the number of misclassifications. Thus the threshold depends on the class scores and is not fixed over all points.

This is similar to [http://digibuo.uniovi.es/dspace/bitstream/10651/6203/1/multilabel-pr.pdf] where the authors referred to this method as probabilistic thresholds (PT). They found this approach takes very little computation but can cause drastic improvements to metrics such as the F_{1}-score or accuracy. This approach, however, does not improve metrics such as hamming loss. In the paper they compared it to *one threshold* and *meta threshold* from [http://s3.amazonaws.com/academia.edu.documents/39820887/Obtaining_Bipartitions_from_Score_Vector20151109-30004-mdv7br.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1499607607&Signature=JkJHB%2BqK2QyYGE9xzDUKnuCAsaM%3D&response-content-disposition=inline%3B%20filename%3DObtaining_Bipartitions_from_Score_Vector.pdf] to show that PT is on average the best for accuracy and F_{1}-score. Used 10-fold cv.

- see also [https://cs.nju.edu.cn/_upload/tpl/01/0b/267/template267/zhouzh.files/publication/tkde06a.pdf]

The threshold calibration strategies described thus far are mostly general purpose approaches that could be applied as a post-processsing step to any MLC algorithm that outputs class scores.

An alternative to threshold calibration is to decide on the number, say m, of labels to be present for each instance. Then the labels with the m highest class scores will be assigned a 1 and the rest zero. Most of the strategies described above for selecting the best threshold can also be applied to selecting the best m. (also described in the bipartition paper.) Nice paper about it here [http://www2009.eprints.org/22/1/p211.pdf], think it is the same as the Meta threshold mentioned above.

- see adhoc methods such as calibrated label ranking: [https://pdfs.semanticscholar.org/5918/04251e15cfb571bc90c2fab2344f462e1617.pdf] and

3.7 More on Label Dependence

With this chapter I want to investigate the need for approaches in multi-label classification which model the dependence structure between labels. For this we need a sound theoretical definition and analysis of label dependence and then we might want to investigate it empirically with synthetic datasets (or real world). The main papers inspiring this chapter are (Dembcz *et al.*, 2012) and (Read and Hollmén, 2015), and some content will be taken from (**?**), (Madjarov *et al.*, 2012) (for empirical evidence maybe), (**?**), (Dembczy, 2010), (Dembczynski *et al.*, 2012). My main hypothesis is that modelling the input-output pairs individually should have just as good, if not better performance compared to approaches trying to model label dependence, since all the available information of the labels should be contained in X and by the assumption that label y_i can be determined with the help of the knowledge of label y_j, it should also be possible to find y_i from X since y_j is found from X. This argument probably only holds for approaches trying to "correct" binary relevance (BR) with regards to its lack of modelling label dependence, such as classifier chains (CC), stacking like MBR/2BR/BR+, etc. Reformulate hypothesis later.

It is essentially a given in multi-label classification literature that in order to obtain competitive results, a learner should be able to model the dependence structure between labels in some way. Whenever a new MLC algorithm is proposed, it will be compared to independent label learning (BR) and if it has superior empirical performance, it is usually ascribed to its ability of modelling label dependence in some ad-hoc way (examples?). The authors of (Dembczy, 2010), (Dembczynski *et al.*) and (Dembczyński *et al.*, 2010) were the first to point out this lack of understanding of the term *label dependence* in the literature (later on a comprehensive and extended discussion of the topics covered in the aforementioned papers was given in (Dembcz *et al.*, 2012)). They argued that *label dependence* is only understood and used by most in the literature in a purely intuitive manner, and that in order to build a better understanding of multi-label classifiers, theoretical backing is essential.

Modelling each label independently, *i.e.* using the binary relevance (BR) approach, is one of the simplest and most intuitive approaches to tackling the multi-label problem. But it has been criticized and overlooked by the majority because it does not take into account the possible dependence between labels. However, BR has many advantages. (Dembcz *et al.*, 2012) shows that BR is the risk minimizer of the Hamming Loss and (**?**) pointed out that it is very rare for 'improved' methods to achieve significantly better results than BR in terms of this measure (also visible in (Madjarov *et al.*, 2012) (make sure)). In addition, BR is highly resistant to overfitting label combinations, since it does not expect samples to be associated with previously-observed combinations of labels [Read2011a]. It can naturally handle data streaming or other dynamic scenarios where the addition and removal of labels are quite common. BR's biggest strength is its low computational complexity compared

to other multi-label classification methods. It scales linearly with increasing number of labels and it is easily parallelizable - desirable properties, especially working with large label sets.

Recently, (Read and Hollmén, 2015) has gone so far as to claim that BR can perform just as well as methods supposedly modelling label dependence, and if it does not, it is usually because of the inadequacy of the base learners used. In other words, if the base learner can extract the right features, BR will be as good as any other multi-label classifier, without the need to model label dependence. Some theoretical justifications were given but the empirical evidence was not convincing. This is what motivated the writing of this chapter - to answer the question, "is it essential for a multi-label classifier to take label correlations into account in order to be optimal?". To investigate this one needs a thorough, theoretical understanding of *label dependence*, how to possibly exploit it and how to evaluate it. This is what this chapter aims to do. Most of the work is based on the papers (Dembcz *et al.*, 2012) and (Read and Hollmén, 2015). We will also attempt to back up the theory with empirical results.

3.8 Conclusion

- more empirical evidence is needed; with with wide range of data sets, algos and measures; compare with statistical tests and consider computation time (training and test)
- part on statistical tests in (Gibaja and Ventura, 2015)
- trees for efficiency, ensembles for predictive performance, transformation methods for flexibility
- label correlation understanding is holy grail of ML (**?**)
- complement of this paper would be a broad empirical study

Chapter 4

Convolutional Neural Networks for Multi-Label Image Classification

4.1 Introduction

This post is about using deep neural networks (DNNs) to perform multi-label image classification (MLIC). MLIC is a generalisation of the single-label image classification task. It allows for an input image to be annotated with more than one label (*i.e.* classes are not mutually exclusive) which is often required in real-world applications. DNNs have showed its superiority over other methods when it comes to single-label image classification and other Computer Vision tasks. Here we investigate how it can be used to solve the lesser researched problem of MLIC.

To motivate the importance of MLIC, we will give examples of some of its useful applications. Thereafter we will discuss the main challenges of MLIC. The majority of this post covers existing deep learning approaches to MLIC in chronological order of publishing and how they attempt to overcome the above mentioned challenges. Note that we restrict our study space to methods that can be described as end-to-end or unified learning approaches. We will conclude with suggestions for future research.

4.2 Applications

MLIC is a more general and practical problem compared to single-label classification since most real-world images contain multiple objects of different categories. Some interesting (for me at least) datasets for MLIC are listed below:

- **Planet: Understanding the Amazon from Space**: The dataset consist of ± 100,000 satellite images of the Amazon rainforest labeled with various land cover/land use classes along with atmospheric conditions. Each satellite image is associated with at least one label (see Figure @ref(fig:apps) (a)) and therefore fits into the MLIC framework. The data was released as part of a Kaggle competition. The competition hosts wanted a model that can accurately monitor the Amazon from satellite images in order to help them prevent deforestation.

- **ChestX-ray8**: The ± 110,000 images in this dataset are chest X-rays labeled with various thoracic pathologies where each image can have multiple labels (see Figure @ref(fig:apps) (b)). The data was released by the NIH (National Institute of Health) to promote the development of computer aided diagnosis and detection.

- **WIDER-Attribute**: This is a human attribute prediction benchmark dataset. From the dataset we can get ± 60,000 images of individuals labeled with their corresponding attributes (see Figure @ref(fig:apps) (c) for an example).

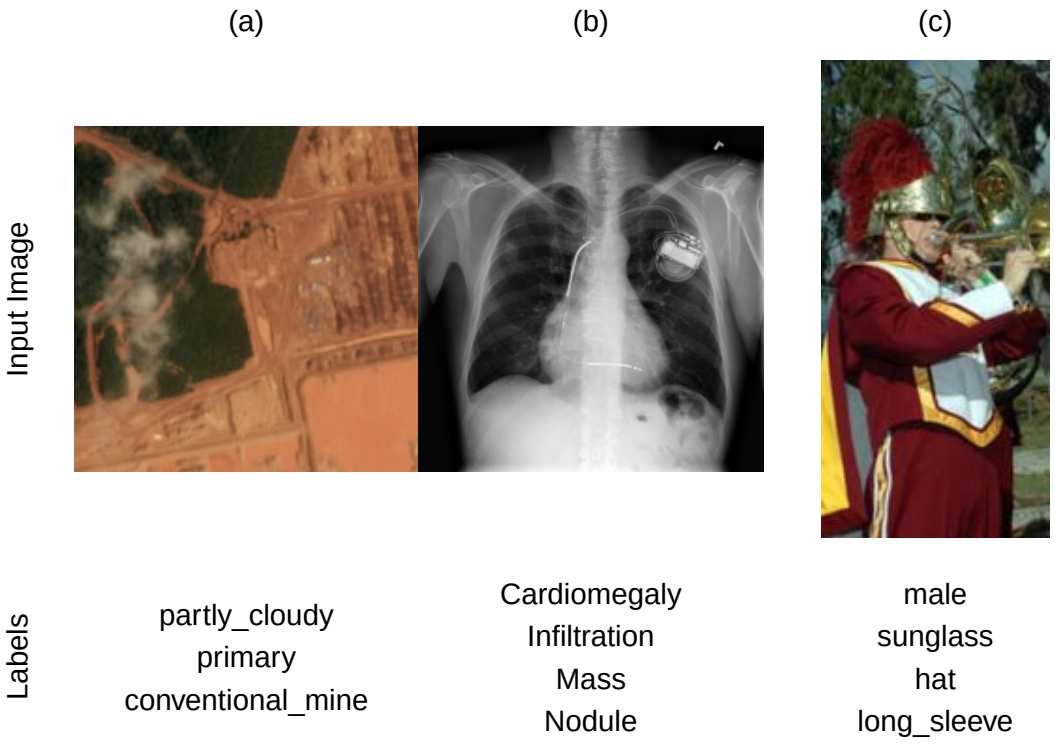

Figure 4.1: Examples of input-output pairs in a MLIC problem.

4.3 Main Challenges

Here, we show how MLIC is different to single-label classification[1] and how these differences make the MLIC task more challenging. We will first look at the differences in terms of the input images, then how the multi-label output presents some new challenges, followed by the added complexity of evaluating MLIC models.

4.3.1 How are the input images different to conventional image classification?

Objects in a typical multi-label image come in various sizes and positions, wheras in a typical single-label image, the object is usually in the center and covering a large part of the image. See for example in Figure @ref(fig:apps) (c) where the image region related to 'sunglass' is very small compared to the region related to 'male'. The objects in a multi-label image are often also only partially visible, for example in Figure @ref(fig:apps) (a) where the 'partly_cloudy' region covers a big part of the 'primary (forest)' region. Notice the presence of spatial relations between label regions in a multi-label image. Again in Figure @ref(fig:apps) (c) we can observe that the image regions related to 'sunglass' and 'hat' will usually be close to each other, in contrast to 'hat' and say 'shoes', which will rarely occur in the same proximity.

This diverse and complex contents of multi-label images make it difficult to learn effective feature representations and classifiers. This raises a question on the optimality of transfer learning from single-label image datasets to do MLIC, since the nature of the two types of images differ greatly. Using a network pre-trained on a object detection dataset might be a better starting point.

One approach to make better use of single-label feature representations is to transform the MLIC problem into mutliple multiclass classification problems over various regions of the images, *i.e.* identify patches in an image which is likely to be related to only one label and perform a single-label image classification on each of those patches. This is a common approach in object detection. An example of such an approach in MILC can be found in Wei *et al.* (2014) (see Figure @ref(fig:HCP)). However, we will not consider this method, since the so-called 'hypotheses extraction' step can be tedious and requires a separate algorithm. It therefore does not meet our requirements of an end-to-end network solution.

[1] I chose to compare MLIC to single-label classification since it is one of the more commonly known and basic tasks in computer vision. MLIC also has close relationships with (weakly-supervise) object detection and image captioning.

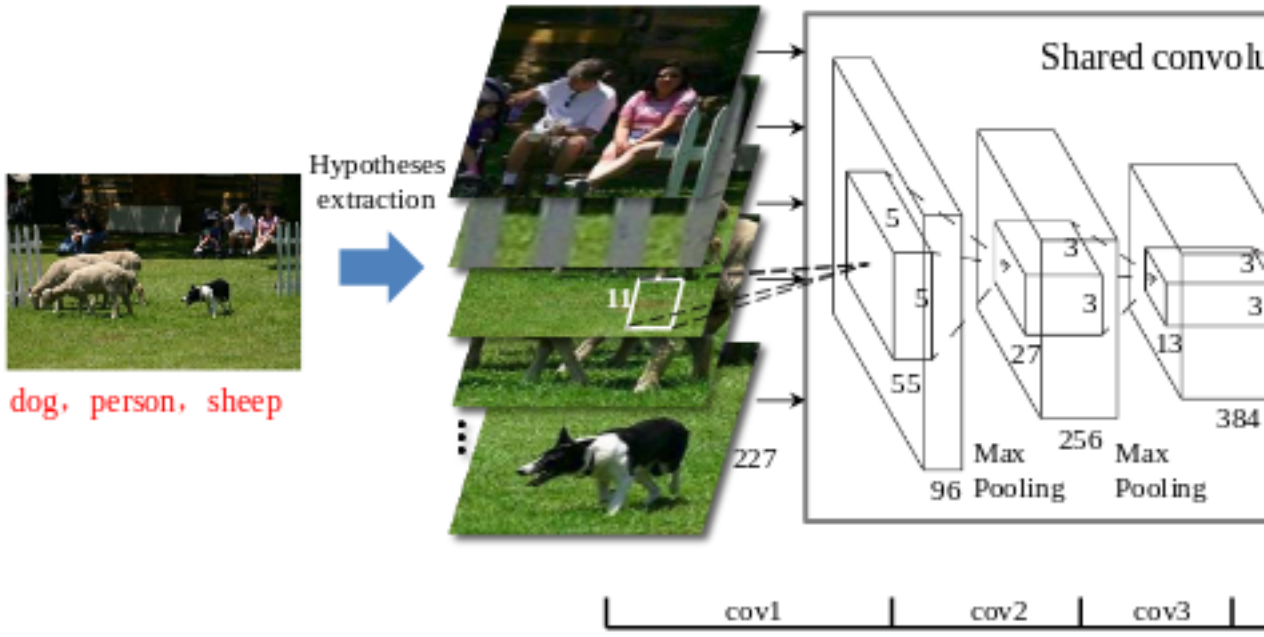

Figure 4.2: An illustration of the infrastructure proposed in @Wei2014.

4.3.2 How does the mutli-labeledness influence learning?

The output space of a MLIC problem can get really big. The number of possible label combinations increases exponentialy with the increase in the number of unique labels in the dataset. This is said to be the key challenge in learning from multi-labeled data - handling such large label spaces. Not only does it require a large amount of computational power (which is not necessarily such a big problem anymore, thanks to DL + GPUs) but it also introduces other problems that needs to be considered in the modeling process, such as extreme class imbalance and label sparsity.

In our examples, the number of labels is relatively small. However, in the YouTube multi-label video classification challenge there are 4716 unique labels. On average, each video in the training set is assoicated with only 3.7 labels and only 14 of the labels occur in more than 1% of the training videos.

To more effectively learn in such large output spaces, algorithms can try to exploit the label dependency structures in the data. As an illustration of such structures, see Figure @ref(fig:corr) for the normalised label co-occurrence matrices of the human attribute prediction dataset. See how ‘male’ almost never occur with ‘longHair’ and ‘skirt’, and most of the time when ‘formal’ is relevant to an image, so to is ‘longSleeve’. These are the types of “correlation”

in the labels we would want a model to exploit. In a perfect world, we would expect a model to identify a skirt regardless of whether or not a male is already identified. However, the data is never a 'perfect world' and knowing whether or not the person in an image is a male might help the model to determine if he/she is wearing a skirt, especially if there are only a few examples of skirts in the training data and/or the skirt is only partially visible.

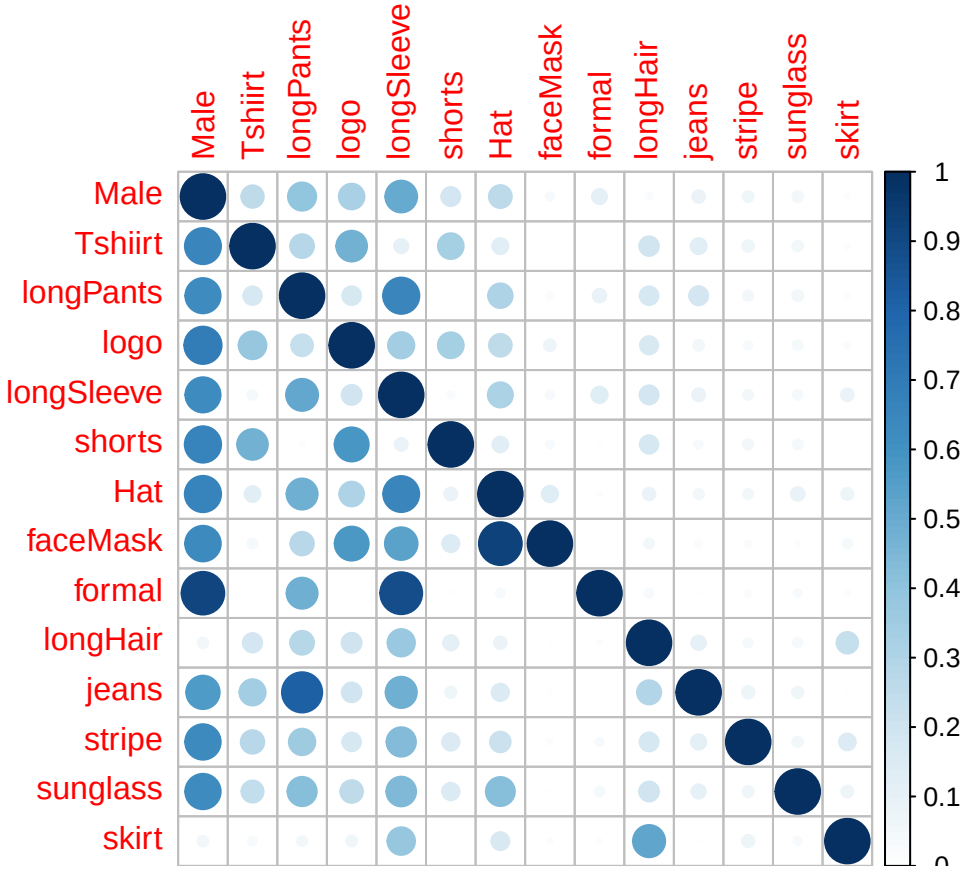

Figure 4.3: The normalised label co-occurrence matrix for WIDER-attribute dataset. The colour intensity in cell (i,j) indicates the number of times label i and j co-occurred divided by the number of times label i occurred.

Exploiting these correlations is not a trivial task and has been the main focus of multi-label classification (MLC[2]) research. The correlations are usualy asymmetric and depend on the input. We explore the different approaches later in the post.

One more consideration for MLIC is that the number of labels associated with an image is unknown. For example in the Chest X-ray data the number of labels associated with an image ranges from 0 to 9. The reason this might be a challenge is as follows. The output of most multi-label classifiers (including

[2]I know this might be confused with MLIC when not read carefully.

DNNs) are in the form of numeric scores for each label, which can usually be interpreted as the likelihood of a label's presence. The problem arises when we need to map these numeric scores to a binary code to ultimately indicate which labels are relevant or not (sometimes referred to as label calibration). In binary classification we threshold the numeric scores to map it to a binary code, for example if $f(x) > 0.5$ let $y = 1$ or $y = 0$ otherwise. In multiclass classification the label with highest class score is assigned a 1 and the rest 0. This is more complicated in MLC since the optimal threshold may vary across labels and if we rather want to choose the k labels with the highest class score, we first need a way to determine k.

There are some interesting proposals in the literature which we will have a look at soon. Note that the solution to the above mentioned challenges are highly dependent on the chosen evaluation metric for the problem. The evaluation metric or loss function of an MLIC problem is again more complex than for the single-label case - we discuss it next.

4.3.3 How do we evaluate MLIC models?

We can't directly optimse a DNN for these multi-label metrics using stochastic gradient descent (SGD), therefore we will need an appropriate (approximately) differentialble surrogate loss function. The standard choice is to use the binary cross-entropy averaged over all labels, but some other suggestions have been made which will be discussed in the next section.

4.4 Overview of Existing Approaches

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.

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

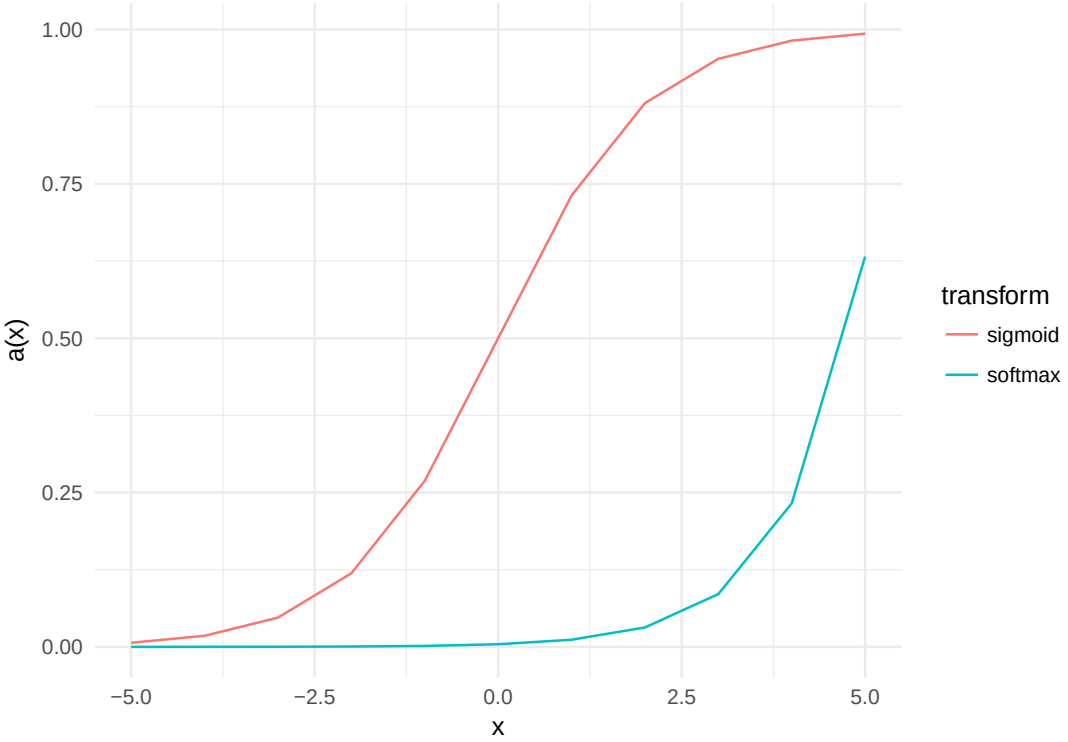

Figure 4.4: Difference between the sigmoid and softmax transformation

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.

4.5 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.

4.5.1 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 4.5. 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 4.5 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.

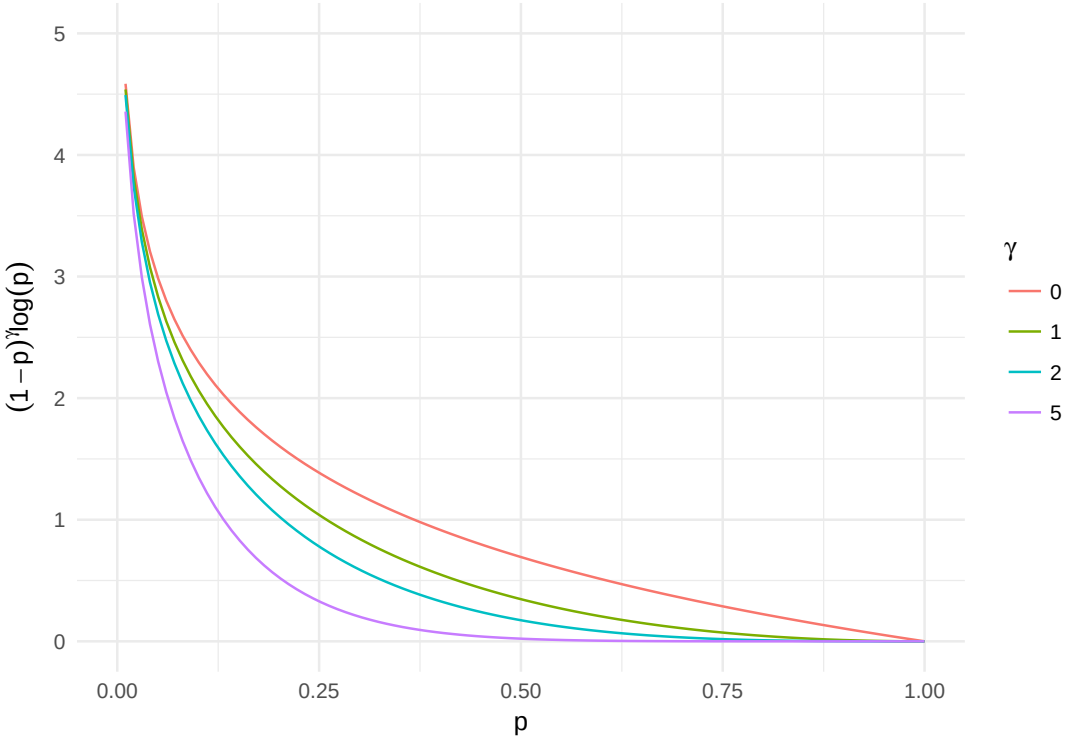

Figure 4.5: The loss contributions made by the focal loss.

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.

4.5.2 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). 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 4.6 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:

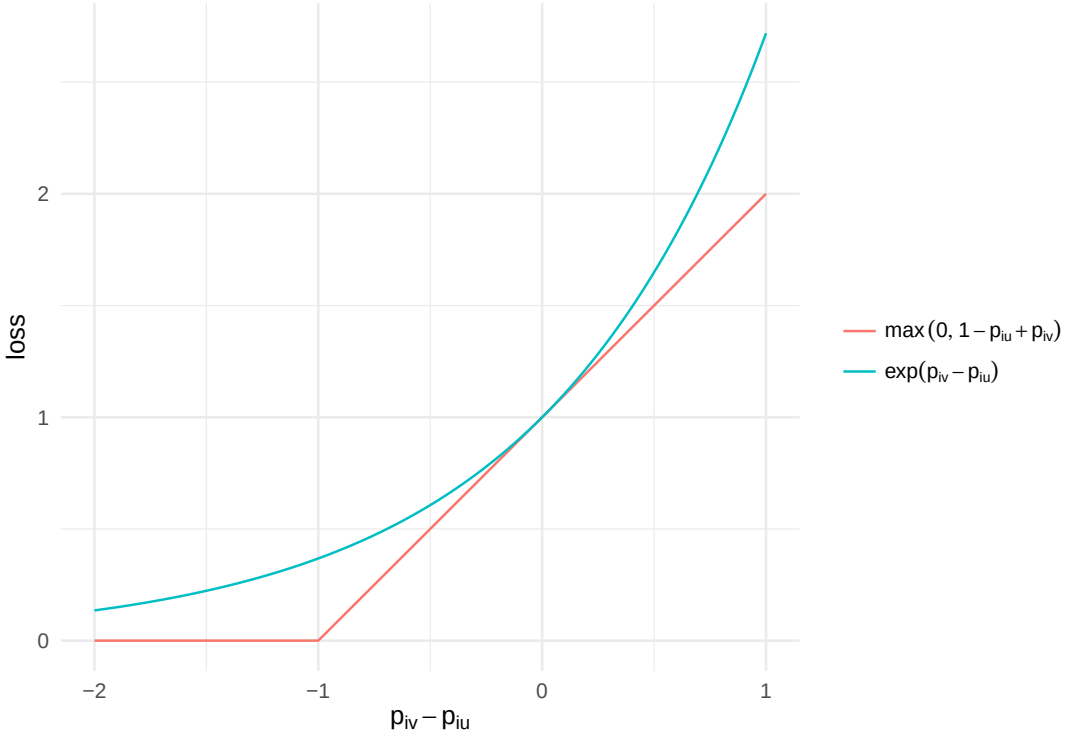

Figure 4.6: The hinge function compared to the log-sum-exp function for different margins.

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

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.

4.5.3 Deep Convolutional Ranking for Multilabel Image Annotation

To the best of my knowledge, this is the first work on using Convolutional Neural Networks (CNNs) for the problem of MLIC. Their main focus was to find a loss function more suitable for MLIC. They used a standard VGG-like CNN architecture and tested the KL-divergence, pairwise-ranking loss and the weighted approximate ranking (WARP) loss. They tested it on the NUS-WIDE dataset and found the WARP loss to give the best results. They also compared it to conventional image feature extraction used with classical classifiers to show that the representation learning approach of CNNs outperforms these methods.

The WARP loss is just a weighted version of the pairwise-ranking loss, weighted by a function of the estimated rank of the labels, hence the name. The pairwise ranking loss is given as

$$J_{rank} = \sum_{i=1}^{n} \sum_{j=1}^{c+} \sum_{k=1}^{c-} \max(0, 1 - f_j(\boldsymbol{x}_i) + f_k(\boldsymbol{x}_i))$$

where j is the index for the positive labels of observation i (of which there are $c+$) and k the index of the negative labels (of which there are $c-$). A non-zero loss is incurred if $f_j(\boldsymbol{x}_i) + f_k(\boldsymbol{x}_i) < 1$, *i.e* when the predicted score of a positive label is not by an offset of 1 bigger than the predicted score of a negative label. This we will call a violation. Therefore we want for any pair (positive label, negative label) the predicted score of the positive label to be much higher than the predicited score for the negative label. Note, if the output of the model, $f(\boldsymbol{x}_i)$ is between 0-1, a non-zero loss will always occur.

- is this actually the hinge loss?

The authors found that this loss function optimises the area under the ROC curve. However, they were more interested in optimising top-k accuracy. Therefore they incorporated a weighting function, $L(.)$, into the pairwise-ranking loss to weight the loss contribution by the estimate rank of the positive label. This resulted in the WARP loss:

$$J_{WARP} = \sum_{i=1}^{n} \sum_{j=1}^{c+} \sum_{k=1}^{c-} L(r_j) \max(0, 1 - f_j(\boldsymbol{x}_i) + f_k(\boldsymbol{x}_i)),$$

where r_j is the estimated rank of the j-th class for image i. They defined $L(.)$ as:

$$L(r) = \sum_{j=1}^{r} \alpha_j,$$

with $\alpha_1 \geq \alpha_2 \geq \cdots \geq 0$. In the paper the authors used $\alpha_j = \frac{1}{j}$. $L(.)$ assigns a higher weight to the loss when a positive label has higher estimated rank (lower r_j). The rank of a positive label is estimated through a sampling trick. For each positive label, a negative label is randomly selected until a violation occurs. The estimated rank of a positive label j is given by

$$r_j = \lfloor \frac{K-1}{s} \rfloor$$

where s is the number of trials it took for a violation to occur. $\lfloor . \rfloor$ denotes the *floor* function. Thus, the longer it takes to randomly find a negative label such that $f_j(\boldsymbol{x}_i) + f_k(\boldsymbol{x}_i) < 1$, the smaller r_j will be.

Note, the experiments were only done on a single dataset and that their metrics were based on a fixed selection of predicted labels, for example $k = 5$. Unfortunately, the WARP loss was not compared to the binary cross-entropy loss in this paper.

4.5.4 CNN-RNN: A Unified Framework for Multi-label Image Classification

The CNN-RNN architecture has proven its worth in sequential prediction tasks like machine translation and image captioning. In this work the CNN-RNN network is applied to MLIC by treating the problem as an ordered prediction problem. Their proposed CNN-RNN network is able to effectively learn the semantic redundancy and the co-occurrence dependency in an end-to-end manner. Although in the prediction phase a beam search is required and therefore the complete approach is not strictly what we are looking for.

The multi-label RNN learns a low dimensional joint image-label embedding to model the semantic relevance between images and labels. The image embedding is done by a CNN (VGG pretrained on ImageNet), while each label has its own learnable embedding vector. In this space a long short term memory (LSTM) recurrent layer is used to model the high-order label co-occurrence dependency. The LSTM layer is able to maintain contextual information about the previously predicted labels in it internal memory states. At each time step the probability of a label is computed based on teh image embedding and the output of the recurrent neurons. This approach is comparable to that of CCs. During prediction, the multi-label prediction with the highest probability can be approximately found with beam search algorithm. See Figure @ref(fig:cnn-rnn1) for an illustration of this framework.

In their experiments they tested the model on the the three most common MLIC benchmark datasets, NUS-WIDE, MSCOCO and PASCAL VOC 2007. However, again only in terms of fixed top-k metrics. Overall their model showed the best results, including better than the WARP approach. They also showed how the binary cross-entropy loss gives better results than WARP. Surprisingly,

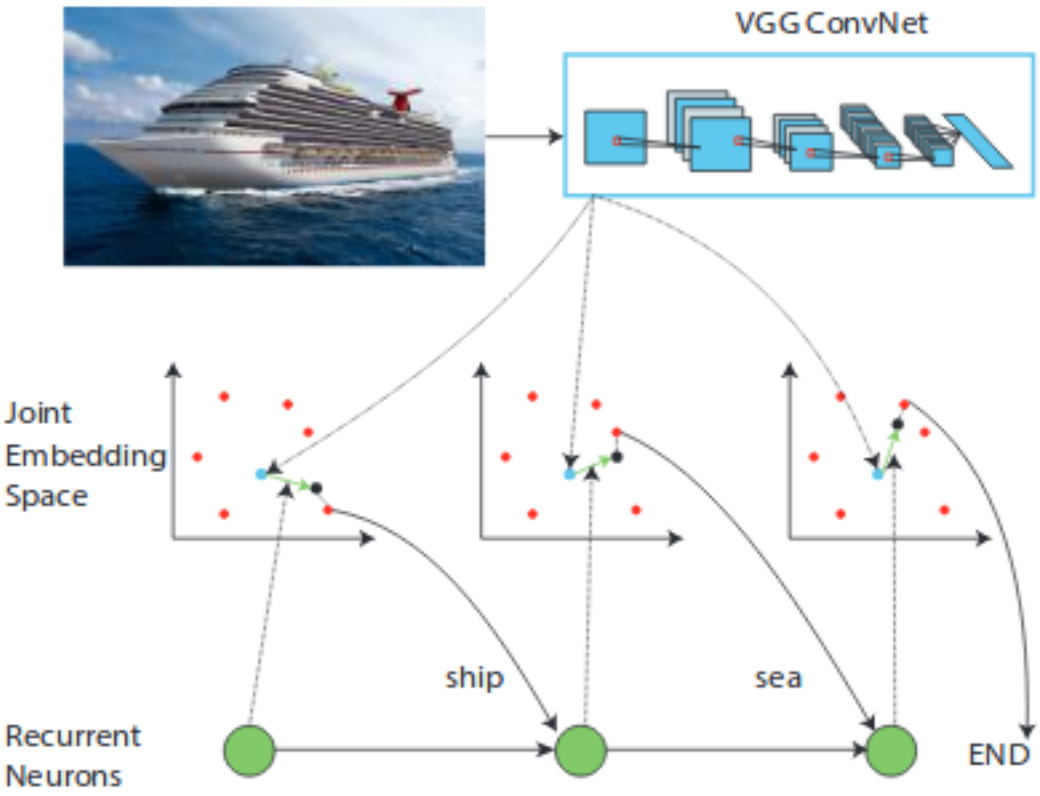

Figure 4.7: An illustration of the CNN-RNN framework.

they did not fine-tune the CNN part of the network, supposedly for simplicitly reasons, which would definitely improve their results.

An added bonus of this framework is that the RNN allows the model to adapt its focus on the image features for each label prediction, meaning for each label, the model can look at different parts of the image. In theory this should help to recognise smaller objects, however the authors reported that small objects were still hard to classify.

A major drawback of this approach is that the order of the prediction path during training should be pre-specified. There is no best way to determine this order. The authors suggests making the assumption that the labels appearing the most should be the easiest to learn and therefore predicting them first.

4.5.5 Annotation Order Matters:Recurrent Image Annotator for Arbitrary Length Image Tagging

- another RNN based approach.
- experiment with different label orders and find rare-first to work the best.

- evaluation using fixed annotation length is not realistic and arbitrary length annotation is required.
- propose Recurrent Image Annotator (RIA) (uses CNN+RNN)
- in the annotation phase, RIA receives an image as input and the outputs a label one-by-one (inspired by image captioning).
- The advantages of using RNN do not only include its nature to generate varied length outputs, but also its ability to refer to previous inputs when predicting the current time step output. Such ability allows RNN to exploit the correlations of both image-to-tag and tag-to-tag.
- no natural order, so we have to choose of learn an order.
- see Figure @ref(fig:ria) for an illustration.

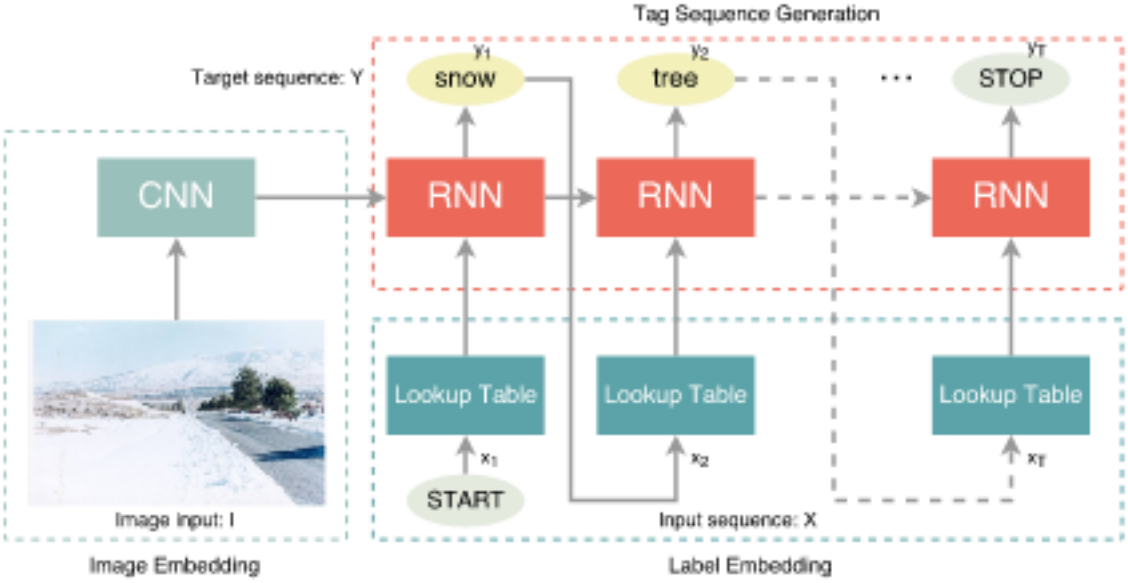

Figure 4.8: An illustration of the RIA framework.

- use a linear projection layer to embed the image into a lower dimensional space, suitable for the RNN,
- same with the label vectors
- all very similar to the original CNN-RNN (can be seen as representative)
- predict until stop signal, predicted
- cross-entropy loss
- experimented with four label orders: dictionary (alphabetical) order, random order, rare-first order and frequent-first order

- tested on lesser known datasets
- used VGG pretrained
- Adam opt.
- rare-first did the best

4.5.6 Learning Spatial Regularization with Image-level Supervisions for Multi-label Image Classification

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 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.

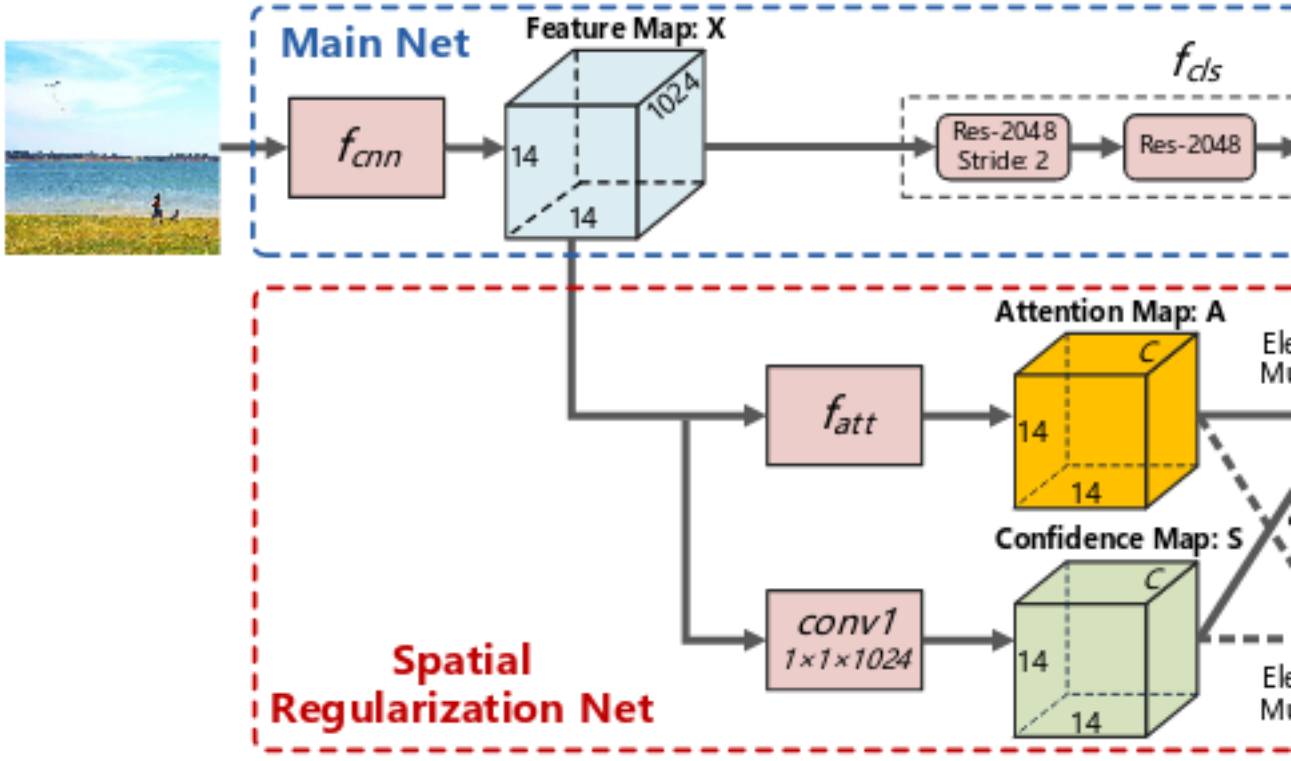

Figure 4.9: An illustration of the SRN framework.

4.5.7 Improving Pairwise Ranking for Multi-label Image Classification

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. The proposed loss function is called the Log-sum-exp pairwise (LSEP) function,

$$J_{LSEP} = \log\left(1 + \sum_{v \notin Y_i} \sum_{v \in Y_i} \exp(f_v(\boldsymbol{x}_i - f_u(\boldsymbol{x}_i))\right)$$

This is a smooth approximation of the J_{rank} loss defined above (note, I prefer this notation so I should change the above loss function's notations). J_{rank} is not smooth everywhere and is thus difficult to optimse. They also suggest to use a negative sampling trick if the number of label pairs is large. Specifically, sampling at most t pairs from the cartesian product denoted as $\phi(Y_i; t) \subseteq Y_i \otimes \mathcal{Y} - Y_i$. This gives us,

$$J_{LSEP} = \log\left(1 + \sum_{\phi(Y_i;t)} \exp(f_v(\boldsymbol{x}_i - f_u(\boldsymbol{x}_i))\right)$$

Note, that this does not have the weighting function of the WARP loss. It is possible to incorporate it here as well, however the authors found it unecessary. They believe weighting is done implicitly by LSEP.

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.

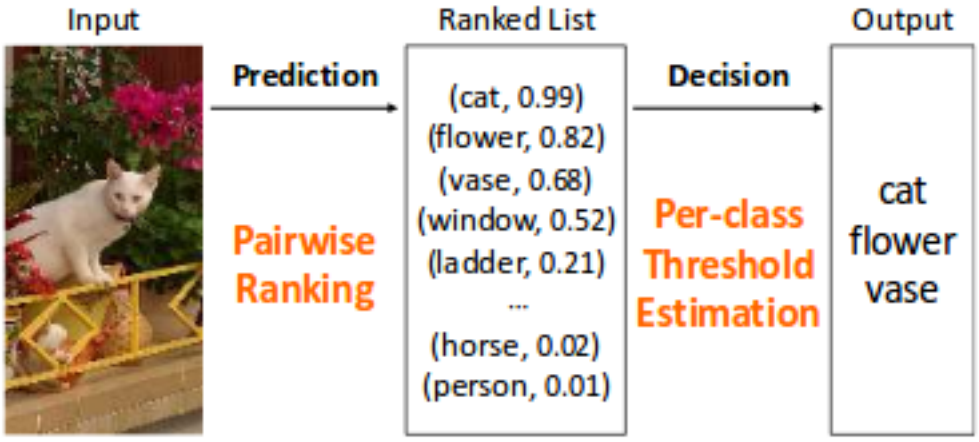

Figure 4.10: An illustration of the label decision module concept.

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.

4.5.8 ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases

This works introduce the Chest X-ray dataset mentioned earlier. Along with the dataset, they published a few benchmark results on the data. They took the transfer learning approach of using a pretrained CNN and swapping out the top layers for a specific purpose. Here they experimented with 4 differenct CNN architecutres pretrained on ImageNet: AlexNet, GoogleLeNet, VGG and ResNet 50 (which gave the best results). For their top layers they chose a transition layer, Global pooling layer and prediction layer (see Figure @ref(fig:chestnet8)).

The transition layer is a convolutional layer (?) transforming the activations from the different base CNNs to an uniform dimension. This is crucial since they don't train the base CNN layers on the dataset (which is something worth experimenting).

The global pooling layer is a novel pooling method to serve as an in-between of the max and average pooling layers. They called it the log-sum-exp pooling layer and defined it as

$$x_p = x^* + \frac{1}{r} \cdot \log \left[\frac{1}{S} \sum_{(i,j) \in \boldsymbol{S}} \exp(r \cdot (x_{ij} - x*)) \right],$$

where $x^* = \max \{|x_{ij}|, (i,j) \in \boldsymbol{S}\}$, X_{ij} is the activation value at (i,j), (i,j) is one location in the pooling region $\boldsymbol{S}$, and $S = s \times s$ is the total number of

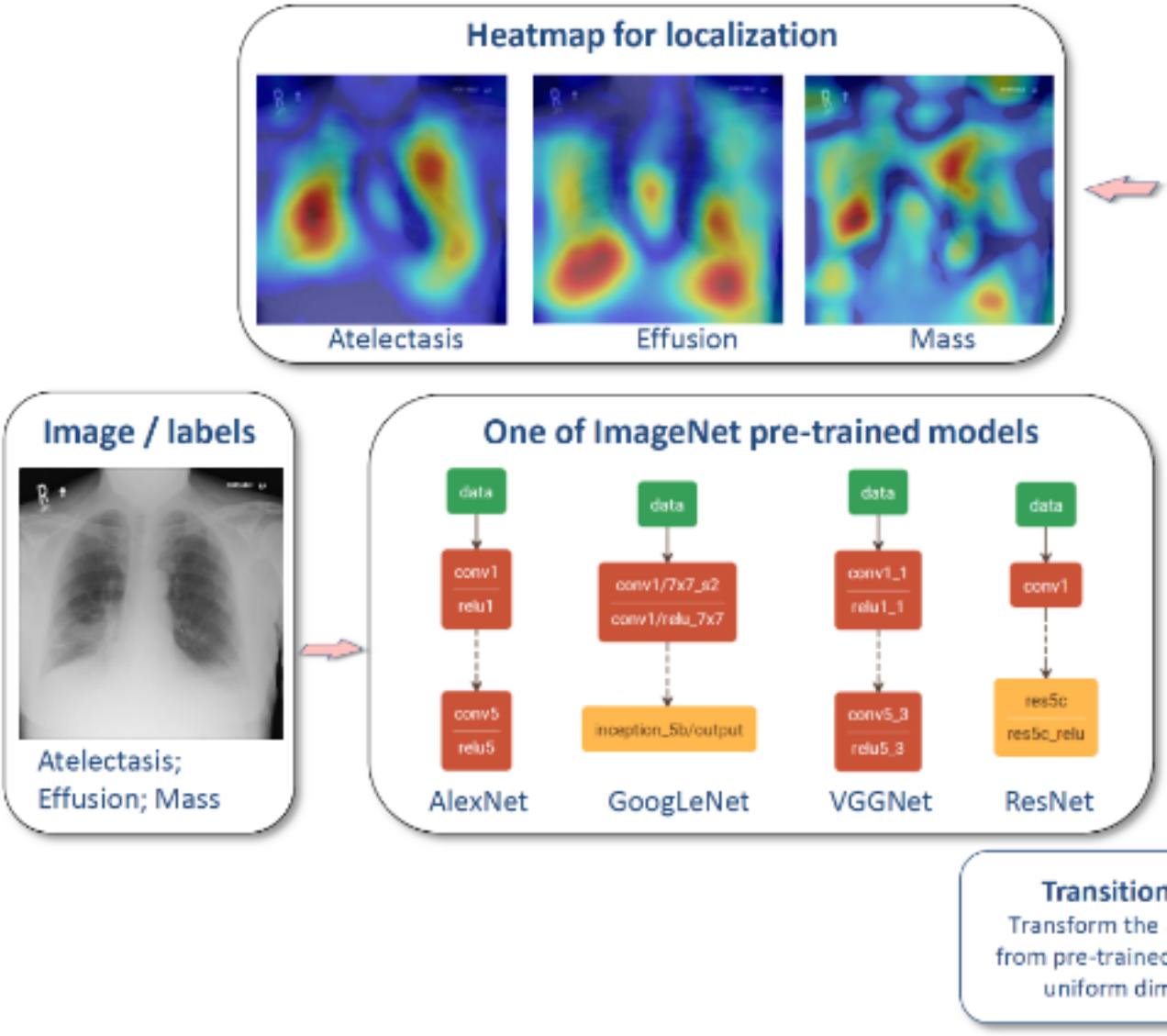

Figure 4.11: Illustration of the framework proposed in this work.

locations in $\boldsymbol{S}$. When $r \to \infty$ the operation is equivalent to max pooling and when $r \to 0$ it becomes equivalent to the average pooling operation. The use of this pooling gave negligible improvements in terms of accuracy and added complexity because r is a hyperparameter that needs to be tuned. However, the authors were also interested in extracting the pathologie's localisations in the images, and this pooling method seemed to improve the quality of these localisation maps. See Figure @ref(fig:chestnet8) for an illustration of the pooling operations and why it may help with localisation.

Although not specified, I suspect the prediction layer is a standard fully connected layer with a sigmoid activation. They experimented with the hinge, euclidean, cross-entropy and weighted cross-entropy (W-CE) loss functions. They found W-CE to give the best results. W-CE is defined as

$$J_{WCE} = \beta_P \sum_{y_c=1} -\ln(f(\boldsymbol{x}_c)) + \beta_N \sum_{y_c=0} -\ln(1 - f(\boldsymbol{x}_c))$$

where $\beta_P = \frac{|P|+|N|}{|P|}$ and $\beta_N = \frac{|P|+|N|}{|N|}$, with $|N|$ the number of positive classes and $|N|$ the number of negative classes. Therefore the positive labels will be given more weight if there are few of them. They argue that this formulation is necessary since otherwise the network struggles to find positive instances because of the sparsity and imbalance in the labels.

By performing a global pooling after the transition layer, the weights learned in the prediction layer can function as the weights of spatial maps from the transition layer. Therefore, we can produce weighted spatial activation maps for each disease class by multiplying the activation from transition layer and the weights of prediction layer.

4.5.9 Order-Free RNN with Visual Attention for Multi-Label Classification

4.5.9.1 Summary

- predetermined label order for LSTM is not ideal
- inconsistency between training and predictiion phases
- present a novel deep learning framework of visually attended RNN (visual attention + LSTM)
- this model learns the correct label order from the image
- improved attention module to accommodate labels associated with small image regions.
- Therefore, how to introduce the flexibility in learning optimal label order while jointly exploiting the associated visual information would be the focuses of our proposed work.
- 3 modules: mapping, attention, prediction - see Figure @ref(fig:impCNNRNN)
- mapping extracts visual features from image with pretrained CNN

- the attenion module learns a set of feature maps describing the corresponding layer of image semantic information.
- prediction module is a LSTM followed by a final prediction layer.
- during the LSTM prediction process, after each step the attention module is updated by the hidden state guiding the network to visually attend the next area of interest in the image.
- FC layer after CNN as initial prediction
- use prediction of previous time step to order labels by confidence
- use candidate pooling to avoid predicting duplicates
- also use beam search to find best prediction path

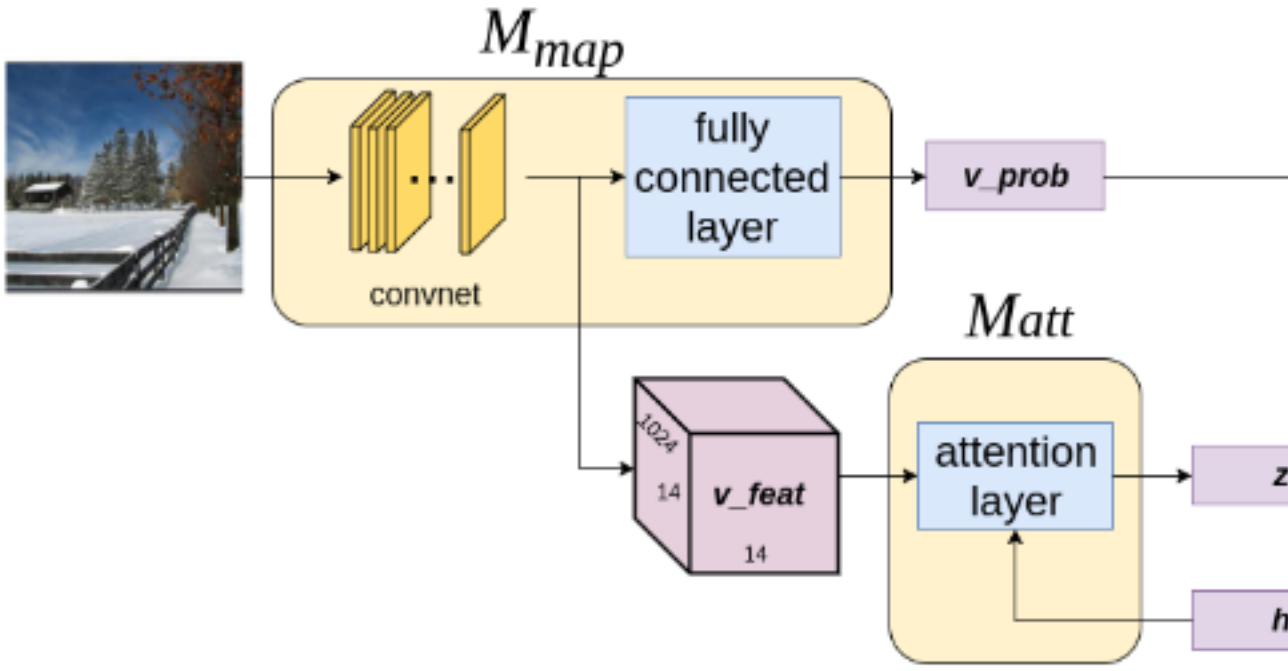

Figure 4.12: Illustration of the framework proposed for an improved CNN-RNN.

- tested on MSCOCO and NUS-WIDE
- does not do better than SRN
- compares other label orders
- can visualise attention

4.6 Learning to diagnose from scratch by exploiting dependencies among labels

4.6.0.1 Summary

- predicting the absence of a label is just as important as predicting its presence
- takes conditional dependence into account
- use of out-of-box RNN decoders are naive
- show no need for pretraining since images data are large
- significant dependencies among labels in this dataset
- show that ordering does not seem to impose as a significant constraint when models are sufficiently trained
- used densenet as CNN encoder since better for smaller amounts of data
- use a LSTM and treat the mlc as a sequence prediction of fixed length
- no attention mechanism and no need to know when to stop
- sigmoid at each step
- runs for the K iterations
- use greedy search

4.6.1 Multi-label Image Recognition by Recurrently Discovering Attentional Regions

4.6.1.1 Summary

- capable of automatically discovering semantic-aware regions over image scales and simultaneously capturing their long-range contextual dependencies.
- incorporate a recurrent memorized-attention module in the neural network to simultaneously locate the attentional regions and predict the labels on various located regions. Our proposed method does not resort to the extraction of object proposals and is thus very efficient and can be trained in an end-to-end mode.
- see @ref(fig:rnnatt)

- uses a VGG CNN
- the spatial transformer (ST) locates an attentional region for the LSTM, and the LSTM predicts the scores regarding this region for multi-label classification and simultaneously updates the parameters of ST. Finally, the scores from several attentional regions are fused to achieve the final label distribution.
- details on ST
- category wise max pooling

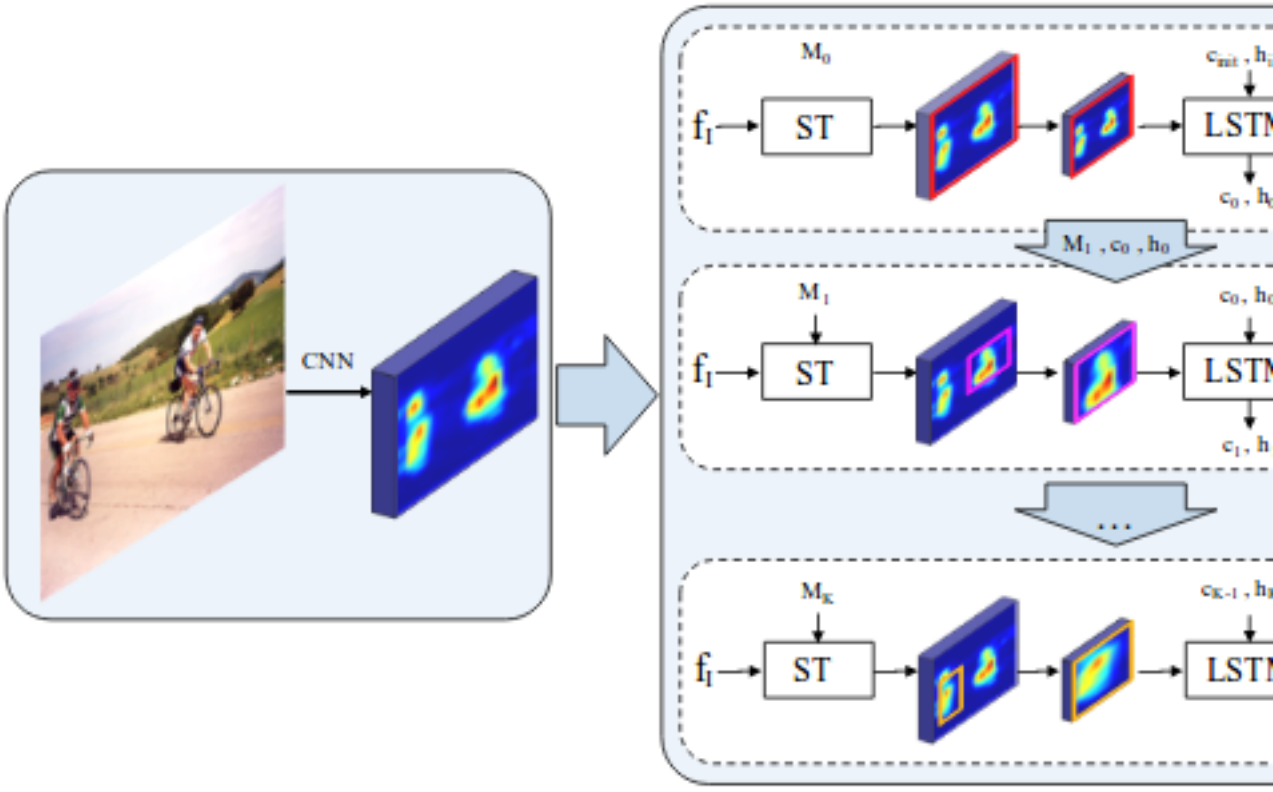

Figure 4.13: Illustration of the framework proposed for this RNN with improved attention.

- for some reason they use the euclidean loss
- still need to look at the rest of the complicated details.

4.6.2 CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning

CheXNet is the latest attempt to improve the performance on the Chest X-ray dataset. Their main focus was on detecting Pneumonia, however, they also extend the model to do MLIC. In Figure @ref(fig:chexnetres) we see how much better CheXNet does compared to the other methods.

No novel MLC technique was used in CheXNet. The boost in performance came from using a more modern CNN architecture, known as DenseNet. This time the network was trained from scratch (no transfer-learning). The classification layer was a standard fully-connected layer with an elementwise sigmoid activation, supervised with the binary cross-entropy sum over all labels.

This is a good example of how the base CNN influences the performance

Pathology	Wang et al. (2017)	Yao et al. (2017)	C
Atelectasis	0.716	0.772	
Cardiomegaly	0.807	0.904	
Effusion	0.784	0.859	
Infiltration	0.609	0.695	
Mass	0.706	0.792	
Nodule	0.671	0.717	
Pneumonia	0.633	0.713	
Pneumothorax	0.806	0.841	
Consolidation	0.708	0.788	
Edema	0.835	0.882	
Emphysema	0.815	0.829	
Fibrosis	0.769	0.767	
Pleural Thickening	0.708	0.765	
Hernia	0.767	0.914	

Figure 4.14: This table reports the AUC per pathology for CheXNet compared to the previous baselines.

of a network and why it is unfair to compare MLIC techniques when the base networks are different.

4.6.3 Proposals from video classification

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.

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.

4.7 Latest Developments in CNN

- dilated convolutions
- se-resnet

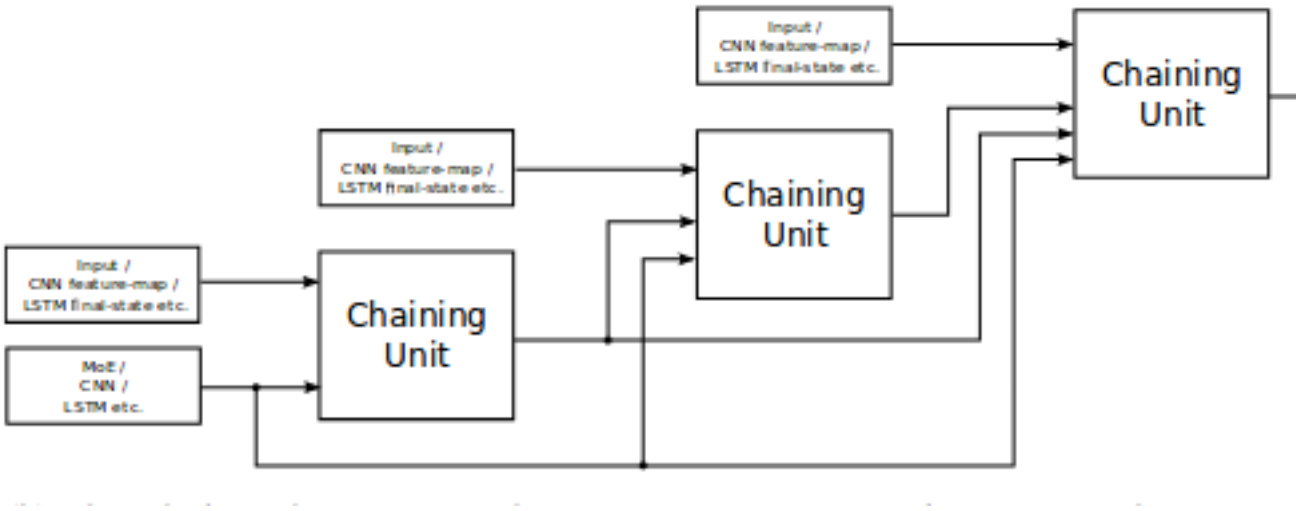

Figure 4.15: An illustration of the chaining layers.

4.8 Suggestions from object decection

Focal loss Lin *et al.* (2017) as a way to do hard example mining and to overcome class imbalance. Work very well in object detection. Worth a try for a MLIC problem.

4.8.1 Multi-scale features

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

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.

4.9 Sugesstions from image captioning

- glove/ word2vec embeddings
- not sure if already done
- Meta Analysis:
  - unsure if this is necessary

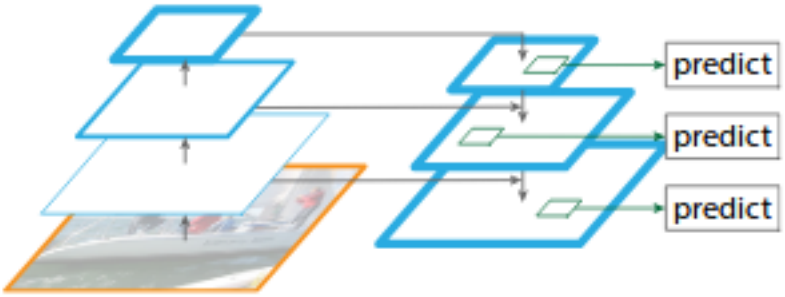

Figure 4.16: An illustration of the FPN concept in @Lin2016.

- maybe can come earlier to show why a overview/ comparision is valuable
- want to compare things like what datasets were tested on, way of evaluation, base networks, results, etc.

4.10 Conclusion

- what we discussed
- Future directions and open questions?

Chapter 5

Results

5.1 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 A), 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?

5.2 Loss Functions for Multi-Label Image Classification

5.2.1 Method

5.2.2 Results

5.2.3 Discussion

5.3 Multi-Level Predictions to Detect Small Objects

5.3.1 Method

5.3.2 Results

5.3.3 Discussion

5.4 Exploiting Label Correlations

5.4.1 Method

5.4.2 Results

5.4.3 Discussion

5.5 Label Calibration

5.5.1 Method

5.5.2 Results

5.5.3 Discussion

5.6 The Final Model

5.6.1 Method

5.6.2 Results

5.6.3 Discussion

5.7 Notes

- maybe also shrink images for faster computations.

- add time taken for learning as a metric.

Chapter 6

Conclusion

6.1 Contributions

Novel contributions made to the literature by this thesis:

- First review of multi-label DNNs - there are a few papers on ML-CNN. They are usually compared to only a limited number of other approaches, on limited number of datasets and limited number of metrics. To the best of our knowledge, at the time of completing this thesis, no other work provides such an extensive review of the literature.
- Some of the proposed algorithms empirical evaluations are lacking completeness. Usually they lack combinations of either or all of the following:
  - evaluated on multiple benchmark datasets
  - evaluated in terms of a representative set of multi-label evaluation metrics
  - estimated errors provided with standard errors
  - compared with a wide range of state-of-the-art algortihms
- improvements on existing approaches?
- summary
- contributions
- reccomendations
- limitations
- future work

Appendices

Appendix A

Benchmark Datasets for Multi-Label Image Classification

A.1 Introduction

The progress of areas in machine/statistical learning is highly dependent on the availability of quality and diverse benchmark data sets. This enables researchers to compare their methods in a wide variety of environments.

Some of the most popular and recent ML benchmark data sets for image classification will be introduced here along with their unique properties.

- very important for stratification: https://arxiv.org/pdf/1704.08756.pdf and more ML metrics

A.2 Satellite Images

A.2.1 Image Format

The data for this task comes from a set of images (also referred to as chips). Each chip is a small excerpt from a larger image of a specific scene in the Amazon taken by satellites. The chip size in pixels is 256×256, representing roughly 90 hectares of land, and is taken from a larger scene of 6600×2200 pixels. All of the satellite images were taken between January 1, 2016 and February 1, 2017. The format of these images differ from the standard image format. Each image contains four spectral bands: red (R), green (G), blue (B) and near infrared (NIR), where the standard format images usually only contain R, G and B. The additional NIR colour channel is common in remote sensing[1] applications and supposedly allows for clear distinction between water and vegetation in satellite images, for example.

[1]The use of satellite- or aircraft-based sensor technologies to detect and classify objects on Earth [https://en.wikipedia.org/wiki/Remote_sensing].

Another difference between these images and the usual format is that these have pixel intensities in 16-bit digital number format as opposed to the usual 8-bit of standard RGB images. This allows the colours in the images to have a much higher range since 16-bit pixel intensities have 65536 (2^{16}) levels, compared to 256 levels of 8-bit images. This becomes useful, for example, to distinguish between very dark or very bright areas in an image. If the pixel values of a chip gets flattened out into a vector, it will be of size 262144 ($256 \times 256 \times 4$). However, CNNs take the images in their array form as input.

A.2.2 Collection and Labelling of the Images

The image collection was created by first specifying a "wish list" of scenes containing the phenomena the creators wanted to be included, in addition to a rough estimate of the number of such scenes that are necessary for a sufficient representation in the final collection. This set of scenes was then searched for manually on Planet Explorer[2]. From these scenes the 4-band chips were created. A schematic of this process can be seen in Figure A.1. The chips were labelled manually by crowd sourcing. The utmost care was taken to get a large and well-labelled dataset, but that does not mean the labels all correspond to the ground-truth, *i.e.* the data will contain some inherent error. The creators believe that the data has a reasonable high signal to noise ratio.

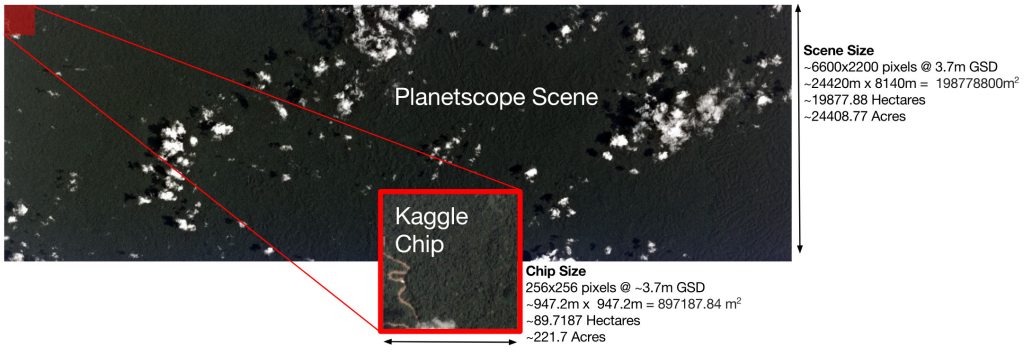

Figure A.1: Schematic of the image collection process.

Note, the training and test splits were determined by the Kaggle competition creators. The training chips are labeled but at the time of writing this, the test chips are not yet made available to competitors. Predicted labels for the test chips can be submitted to Kaggle to evaluate in terms of the F_2-score, a metric which will be discussed in Chapter ??. This setup prevents competitors from

[2]A web based interactive map of Earth consisting of satellite images, similar to Google Earth - www.planet.com/explorer

using the test chips for training a classifier. There are 40479 training chips and 61191 test chips.

A.2.3 Class Labels

The class labels for the images can be divided into three groups: atmospheric conditions, common land cover/use phenomena and rare land cover/use phenomena. In total there are 17 posssible labels. Each chip will have one atmospheric label and zero or more common and rare labels. Chips that are labeled as cloudy should have no other labels.

The atmospheric condition labels are: *clear*, *haze*, *partly cloudy* and *cloudy*. They are relevant to a chip when:

- **clear**: there are no evidence of clouds.
- **haze**: clouds are visible but they are not so opaque as to obscure the ground.
- **partly cloudy**: scenes show opaque cloud cover over any portion of the image but the land cover/use phenomena are still visible.
- **cloudy**: 90% of the image is obscured with opaque cloud cover.

Examples of chips with atmospheric labels can be found in Figure A.2. Each chip should only have one atmospheric label and therefore this classifying task simplifies to a multiclass problem. This allows for the option to break up the labeling task of all the labels into two tasks: a multiclass classification problem for the atmospheric labels and a multi-label classification problem for the land cover/use labels. This approach might save some computational time and give extra information to the multi-label learners for classifying the land cover/use labels. We will experiment with these approaches in Chapter ??.

The common land cover/use labels are: *primary*, *agriculture*, *water*, *habitation*, *road*, *cultivation* and *bare ground*. They are relevant to a chip when:

- **primary**: it is primarily consisting of rain forest (virgin forest), *i.e.* dense tree cover.
- **agriculture**: it contains any land cleared of trees that is being used for agriculture or range land.
- **water**: it contains any one of the following: rivers, reservoirs, or oxbow lakes.
- **habitation**: it contains human homes or buildings.
- **road**: it contains any type of road.
- **cultivation**: it shows signs of smaller-scale/informally cleared land for farming.
- **bare ground**: it contains naturally (not the caused by humans) occurring tree-free areas.

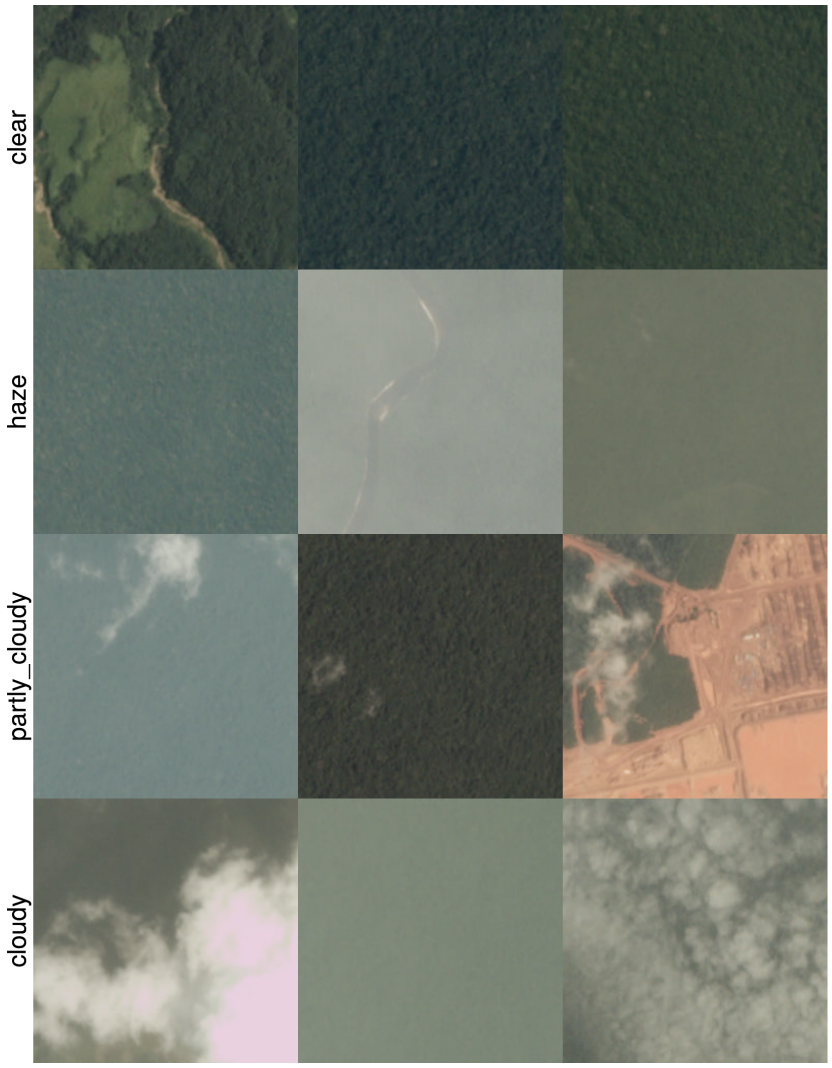

Figure A.2: Examples of chips with atmospheric labels. These (along with all the other chips plotted throughout the thesis) are the JPEG conversions of the original 4-band, 16-bit images.

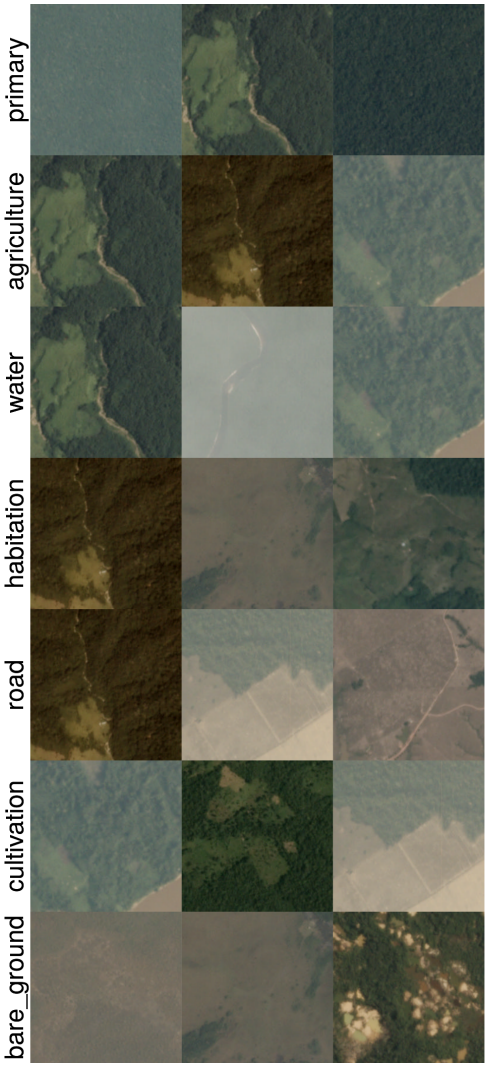

Figure A.3: Examples of chips with common land cover/use labels.

Examples of chips with common land cover/use labels are found in Figure A.3. According to the competition page on Kaggle, small, single-dwelling habitations are often difficult to spot but usually appear as clumps of a few pixels that are bright white. Roads sometimes look very similar to rivers and therefore these two labels might be noisy. The NIR band might give a classifier additional information to help distinguish between the two. Cultivation is a subset of agriculture and is normally found near smaller villages, along major rivers or at the outskirts of agricultural areas. It typically covers very small areas.

The less common land cover/use labels are: *slash and burn*, *selective logging*, *blooming*, *conventional mine*, *artisinal mine* and *blow down*. Chips are tagged with these labels when:

- **slash and burn**: there are signs of the farming method that involves the cutting and burning of the forest to create a field. These look like cultivation patches with black or dark brown areas.
- **selective logging**: winding dirt roads are present adjacent to bare brown patches in otherwise primary rain forest. Selective logging is the practice of selectively removing high values tree species from the rainforest.
- **blooming**: there are signs of trees flowering. Blooming is a natural phenomena where particular species of flowering trees bloom, fruit and flower at the same time. These trees are quite big and the phenomena can be seen in the chips. They usually appear as white dots.
- **conventional mine**: it contains signs of large-scale legal mining operations.
- **artisinal mine**: it contains signs of small-scale (sometimes illegal) mining operations.
- **blow down**: there are signs of trees uprooted or broken by wind. High speed winds (~160km/h) in the Amazon are generated when the cold dry air from the Andes settles on top of the warm moist air in the rainforest and then sinks down with incredible force, toppling larger rainforest trees. These open areas are visible from space.

Examples of chips with these less common land cover/use labels are given in Figure A.4. These labels are more challenging to identify in the chips and since they also appear less frequently, it might be difficult for the classifier to learn these labels. The imbalance in the class distribution is apparent in Figure A.5.

A.3 Chest X-Rays

A.4 Compared to Other Multi-Label Image Datasets

A.4.1 Multi-Label Indicators

As with all supervised learning problems, no one ML algorithm performs optimally on all problems. It is common practice in classical single output supervised learning to first consider, for example, the number of features (p) and the number of observations (n) in a data set before deciding on which model(s) to fit to the data. The same naturally holds for a ML problem but with added complexity. The multiple outputs of the data introduces many more factors to consider before continuing to the modelling phase. Some ML data sets have only a few labels per observation, while others have plenty. In some ML data sets the number of label combinations is small, whereas in others it can be very large. Some labels appear more frequently than others. Moreover, the labels can be correlated or not. These characteristics can have a serious

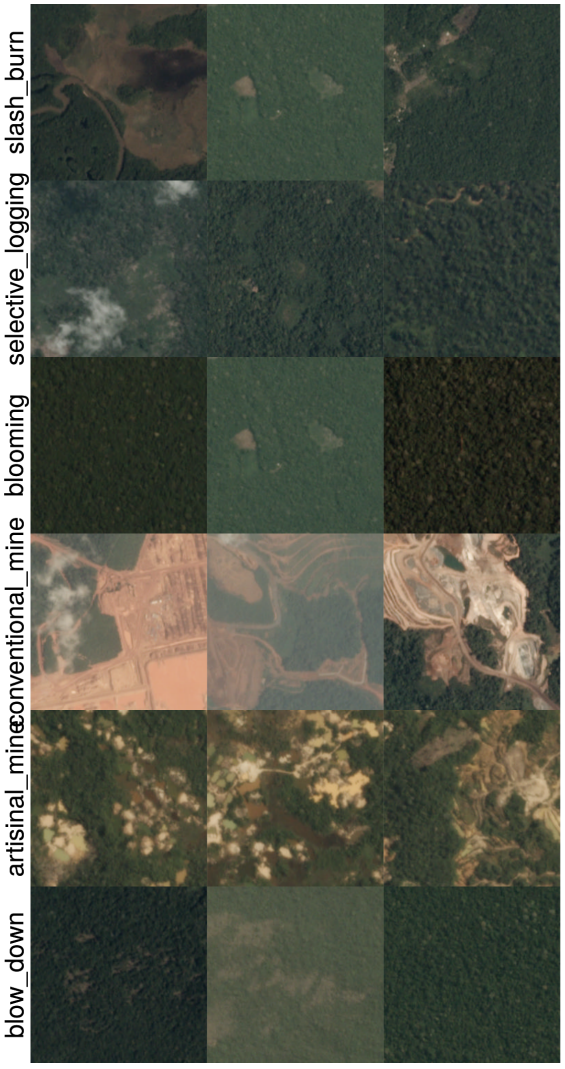

Figure A.4: Examples of chips with less common land cover/use labels.

impact on the performance of a ML classifier. This is the reason why several specific indicators have been designed to assess ML data set properties.

The two standard measures for the multi-labeledness of a data set are *label cardinality* and *label density*, introduced by (Tsoumakas and Katakis, 2007). The label cardinality of a ML data, D, set is the average number of labels per observation:

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

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

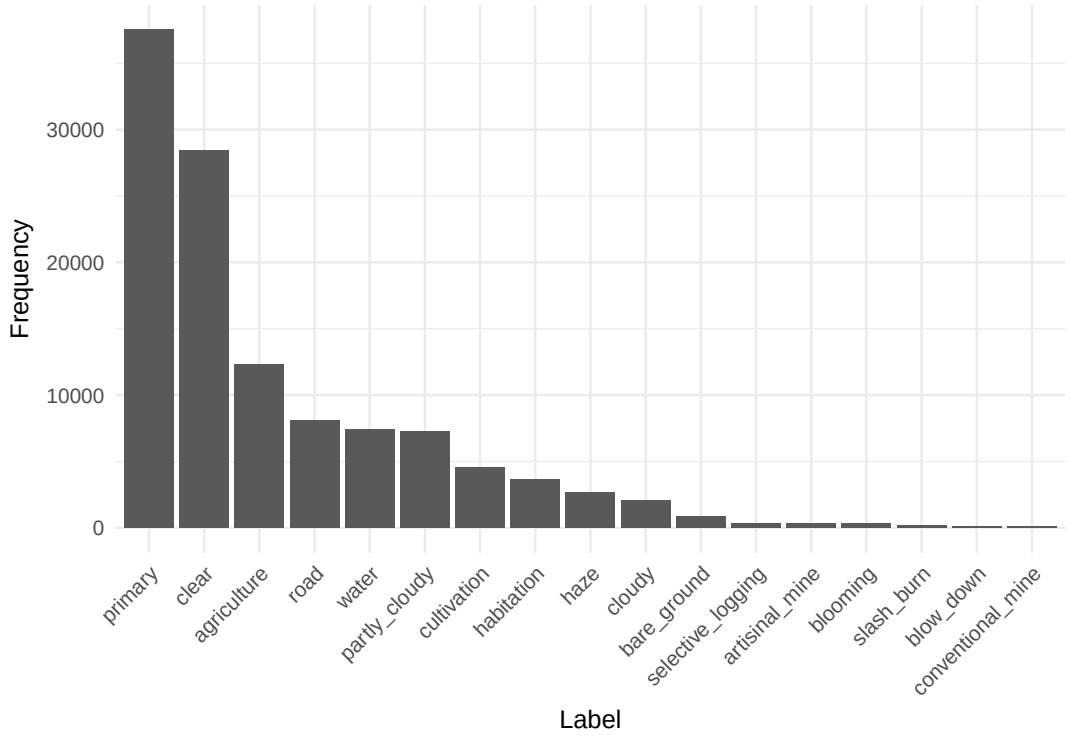

Figure A.5: Class distribution of the labels in the training set.

$$LDens(D) = \frac{1}{K} LCard(D) = \frac{1}{nK} \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik}.$$

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 properties and cause different behaviour to the ML classification 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 (**?**) suggested measuring the proportion of distinct label sets and the proportion of label sets with the maximum frequency. Consider the number of distinct label sets, also referred to as the label diversity (Zhang and Zhou, 2014), which can be defined as:

> there are multiple ways this is defined in the literature - still need to decide on which one I want to use

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

by (Zhang and Zhou, 2014). ((**?**) uses $\exists!$ instead of $\exists$ and Y as a vector $\boldsymbol{y}$. I want to consider a way of defining it in matrix notation. Maybe with

an indicator function. Some papers define it as DL instead of $LDiv$.) The proportion of distinct label sets in D is then

$$PLDiv\{/PUniq/PDL\}(D) = \frac{1}{n} LDiv(D).$$

The proportion of label sets with the maximum frequency is defined by (**?**) as:

$$PMax(D) = \max_{\boldsymbol{y}} \frac{\text{count}(\boldsymbol{y}, D)}{n},$$

where $\text{count}(\boldsymbol{y}, D)$ is the frequency that label combination $\boldsymbol{y}$ is found in data set D. This represents the proportion of observations associated with the most frequently occurring label sets. High values of $PLDiv$ and $PMax$ indicate an irregular and skewed labeling scheme, respectively, *i.e.* a relatively high number of observations are associated with infrequent label sets and a relatively high number of observations are associated with the most common label sets. (*think about this again*) When this is the case, and the labels are modelled separately, the classifiers will suffer from the class imbalance problem, a common problem in supervised classification tasks. More detail about this will be adressed shortly.

Very little research has been done on how all these ML indicators affect the performance of a ML classifier. (Chekina *et al.*, 2011) made a worthy attempt. Their goal was to find a way of determining which ML 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 ML algorithms on benchmark data sets with different properties. This trained meta-learner is then able to predict which ML algorithm is most likely to give the best results in terms of a specific evaluation metric, given the properties of the data set to be analysed. Although we will not use their meta-learner for this thesis, we will consider some of the additional findings in their research. They found that the following properties (among others) of a ML data set was important to their trained meta-model (which was based on classification trees) in predicting which ML algorithm is most appropriate: K; $LDiv(D)$; $LCard(D)$; the standard deviation, skewness and kurtosis of the number of labels per observation in D; number of unconditionally dependent label pairs (based on what?); average of χ^2 -scores of all dependent label pairs; number of classes with less than 2, 5 and 10 observations; ratio of classes with less than 2, 5, 10 and 50 observations; average, minimal and maximal entropy of labels (def of entropy?); average observations per class. This strengthens the argument that it is important to take ML indicators into account before the training process.

- (**?**) defines a complexity measure as $n \times p \times K$

- also new yt8m
- provide a table comparing our datasets with other image datasets in terms of ML properties

A.5 General Comments

- on appropriateness

Appendix B

Software and Code

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

B.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

Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B. and Vijayanarasimhan, S. (2016). YouTube-8M: A Large-Scale Video Classification Benchmark. *arXiv*. 1609.08675.
Available at: https://arxiv.org/pdf/1609.08675.pdfhttp://arxiv.org/abs/1609.08675

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

Carvalho, André C P L F de, A.A.F. (2009). A Tutoria on Multi-Label Classification Techniques. *Foundations of Computational Intelligence*, vol. 5, pp. 177–195.
Available at: http://www.icmc.usp.br/{~}andrehttp://www.cs.kent.ac.uk/{~}aafhttp://www.cs.kent.ac.uk/people/staff/aaf/pub{_}papers.dir/Found-Comp-Intel-bk-ch-2009-Carvalho.pdf

Charte, F., Rivera, A.J., del Jesus, M.J. and Herrera, F. (2015). Addressing imbalance in multilabel classification: Measures and random resampling algorithms. *Neurocomputing*, vol. 163, pp. 1–14. ISSN 09252312.
Available at: http://sci2s.ugr.es/sites/default/files/ficherosPublicaciones/1790{_}2015-Neuro-Charte-MultiLabel{_}Imbalanced.pdfhttp://linkinghub.elsevier.com/retrieve/pii/S0925231215004269

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

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.

Csurka, G., Dance, C.R., Fan, L., Willamowski, J. and Bray, C. (2004). Visual categorization with bags of keypoints. In: *In Workshop on Statistical Learning in Computer Vision, ECCV*, pp. 1–22.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pp. 886–893. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2372-2. Available at: http://dx.doi.org/10.1109/CVPR.2005.177

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

Dembczynski, K., Waegeman, W., Cheng, W. and Hüllermeier, E. (). On Label Dependence in Multi-Label Classification.

Dembczyński, K., Waegeman, W., Cheng, W. and Hüllermeier, E. (2010). Regret analysis for performance metrics in multi-label classification: The case of hamming and subset zero-one loss. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6321 LNAI, pp. 280–295. ISBN 364215879X. ISSN 03029743.
Available at: https://biblio.ugent.be/publication/1155381/file/1210780.pdf

Dembczynski, K., Waegeman, W. and Hüllermeier, E. (2012). An analysis of chaining in multi-label classification. In: *Frontiers in Artificial Intelligence and Applications*, vol. 242, pp. 294–299. ISBN 9781614990970. ISSN 09226389.
Available at: https://biblio.ugent.be/publication/3132158/file/3132170

Gasse, M., Aussem, A. and Elghazel, H. (2015). On the Optimality of Multi-Label Classification under Subset Zero-One Loss for Distributions Satisfying the Composition Property. *ICML*, vol. 37.

Gibaja, E. and Ventura, S. (2014). Multi-label learning: A review of the state of the art and ongoing research. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 6, pp. 411–444. ISSN 19424795.

Gibaja, E. and Ventura, S. (2015). A Tutorial on Multilabel Learning. *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, pp. 52:1—-52:38. ISSN 0360-0300.
Available at: https://www.researchgate.net/profile/Sebastian{_}Ventura/publication/270337594{_}A{_}Tutorial{_}on{_}Multi-Label{_}Learning/links/54bcd8460cf253b50e2d697b.pdfhttp://doi.acm.org/10.1145/2716262

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

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

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

Koyejo, O.O., Natarajan, N., Ravikumar, P.K. and Dhillon, I.S. (2015). Consistent Multilabel Classification. *Advances in Neural Information Processing Systems*, pp. 3303–3311. ISSN 10495258.
Available at: http://papers.nips.cc/paper/5883-consistent-multilabel-classification

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

Lazebnik, S., Schmid, C. and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pp. 2169–2178. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2597-0.
Available at: http://dx.doi.org/10.1109/CVPR.2006.68

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

Li, M., Zhang, T., Chen, Y. and Smola, A.J. (2014). Efficient mini-batch training for stochastic optimization. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pp. 661–670. ACM, New York, NY, USA. ISBN 978-1-4503-2956-9.
Available at: http://doi.acm.org/10.1145/2623330.2623612

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

Lo, H.-Y., Lin, S.-D. and Wang, H.-M. (2013). Generalized k-Labelsets Ensemble for Multi-Label and Cost-Sensitive Classification. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, , no. X.

Lowe, D.G. (2004 November). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110. ISSN 0920-5691.
Available at: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

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

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.

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

Read, J. and Hollmén, J. (2015). Multi-label Classification using Labels as Hidden Nodes. pp. 1–23. 1503.09022.
Available at: https://arxiv.org/pdf/1503.09022.pdfhttp://arxiv.org/abs/1503.09022

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

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

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

Sorower, M. (2010). A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, pp. 1–25.
Available at: http://people.oregonstate.edu/{~}sorowerm/pdf/Qual-Multilabel-Shahed-CompleteVersion.pdf

Sucar, L.E., Bielza, C., Morales, E.F., Hernandez-Leal, P., Zaragoza, J.H. and Larrañaga, P. (2013). Author's personal copy Multi-label classification with Bayesian network-based chain classifiers.

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

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.

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.M. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. *CoRR*, vol. abs/1705.02315.
Available at: http://arxiv.org/abs/1705.02315

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

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

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.

Zhu, F., Li, H., Ouyang, W., Yu, N. and Wang, X. (2017*a*). 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

Zhu, Y., Kwok, J.T. and Zhou, Z.-H. (2017*b*). Multi-Label Learning with Global and Local Label Correlation. 1704.01415.
Available at: https://arxiv.org/pdf/1704.01415.pdfhttp://arxiv.org/abs/1704.01415