Deep 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

Deep Neural Networks for Multi-Label Image Classification

J. A. Marais

Thesis: MCom (Mathematical Statistics)

December 2017

English abstract

Uittreksel

Diep Neurale Netwerke vir Multi-Etikel Beeldklassifikasie

("Deep 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

Declaration **i**

Abstract **ii**

Uittreksel **iii**

Acknowledgements **iv**

Contents **v**

List of Figures **ix**

List of Tables **xi**

Nomenclature **xii**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 2
- 1.3 Code and Reproducibility . . . 3
- 1.4 Background and Important Concepts . . . 3
  - 1.4.1 Image Classification . . . 3
  - 1.4.2 Score Function . . . 6
  - 1.4.3 Loss Function . . . 8
  - 1.4.4 Optimisation . . . 12
- 1.5 Outline . . . 20

2 Deep Neural Networks **21**
- 2.1 Introduction . . . 21
- 2.2 Biological Motivation and Connections . . . 22
- 2.3 Common Activation Functions . . . 22
  - 2.3.1 Sigmoid . . . 22
  - 2.3.2 Tanh . . . 22
  - 2.3.3 ReLu . . . 22
  - 2.3.4 Maxout . . . 22

2.4 Architectures 22
2.5 Setup 22
2.5.1 Data Preprocessing 22
2.5.2 Weight Initialisation 22
2.5.3 Batch Normalisation 23
2.5.4 Regularisation 23
2.6 Loss Functions 23
2.6.1 Classification 23
2.6.2 Attrubute Classification 23
2.6.3 Regression 23
2.6.4 Structered Prediction 23
2.7 Learning 23
2.7.1 Monitoring 23
2.7.2 Parameter Updates 24
2.7.3 Freezing Layers 24
2.8 Hyperparameter Optimisation 24
2.9 Evaluation 24
2.9.1 Ensembles 24
2.10 ConvNet Layers 24
2.10.1 Convolutional Layer 24
2.10.2 Pooling Layer 24
2.10.3 Normalisation Layer 25
2.10.4 Fully Connected Layer 25
2.11 ConvNet Architectures 25
2.11.1 Layer Patterns 25
2.11.2 Layer Sizing Patterns 25
2.11.3 Famous Architectures 25
2.12 Visualizing CNN's 25
2.12.1 Activations and First Layer Weights 26
2.12.2 Images with Maximum Activation 26
2.12.3 t-SNE Embedding 26
2.12.4 Occluding 26
2.13 Transfer Learning 26
2.13.1 Feature Extractor 26
2.13.2 Fine-Tuning 26
2.13.3 Pretrained Models 26

3 Multi-Label Classification 27
3.1 Introduction 27
3.2 The Task of Multi-Label Learning 30
3.2.1 Notation 30
3.2.2 Classification and Ranking 31
3.3 Evaluation Metrics 32
3.3.1 Example-based Metrics 33

3.3.2 Label-based Metrics 34
3.4 Label Dependence 36
3.4.1 Two types of label dependence 37
3.4.2 Link between label dependence and loss minimization . . 41
3.4.3 Theoretical Results 52
3.5 Problem Transformation Approaches 53
3.5.1 Binary Relevance 54
3.5.2 Label Powerset 57
3.5.3 Classifier Chains 57
3.6 Algorithm Adaption Approaches 58
3.6.1 Multi-Label k-Nearest Neighbour (ML-kNN) 58
3.6.2 Multi-Label Decision Tree (ML-DT) 58
3.6.3 Ranking Support Vector Machine (Rank-SVM) 58
3.6.4 Collective Multi-Label Classifier (CML) 58
3.7 Ensemble Approaches 59
3.7.1 Ensemble of Classifier Chains 59
3.7.2 Random *k*-Labelsets 59
3.7.3 Summary 61
3.8 Threshold Calibration 61
3.9 Class Imbalance 62

4 Multi-Label Deep Neural Networks 64
4.1 Introduction 64
4.2 Differences Between Single Label and Multiple Labels Images . . 65
4.3 Short History 65
4.4 Non-Deep Learning Approaches 66
4.5 Optimising Multi-Label Loss functions 66
4.5.1 Sparsemax ML loss 70
4.5.2 F-measure maximisation 71
4.5.3 Other 72
4.6 Proposal Based Approaches 72
4.7 RNN-CNN 72
4.8 Spatial Regularization Networks 76
4.9 Mixture of Experts 78
4.10 Context Gating 79
4.11 Chaining 79
4.12 Label Concept Learning 80
4.13 Label Processing Layer 80
4.14 Nearest Labelset 80
4.15 Label Embedding Approaches 80
4.16 Unsorted 81

5 Results (/Application) 82
5.1 Introduction 82

5.2 Optimising Multi-Label Loss Functions 83
5.3 Label Embedding Approaches 83
5.4 Novel Architectures 83
5.5 Discussion 83

6 Conclusion **84**

7 Things that need a place: **85**

Appendices **86**

A Benchmark Datasets for Multi-Label Image Classification **87**
A.1 Multi-Label Indicators 87
A.2 Amazon 90
A.2.1 Image Format 90
A.2.2 Collection and Labelling of the Images 91
A.2.3 Class Labels 92
A.3 NUS-WIDE 95
A.4 Pascal VOC 95
A.5 MS-COCO 95
A.6 WIDER-Atribute 95

B Software and Code **98**

Bibliography **99**

List of Figures

1.1 Greyscale intensities. http://ai.stanford.edu/\protect\unhbox\voidb@x\penalty\@M\{}syyeung/cvweb/tutorial1.html 5
1.2 Pixelwise difference 5
1.3 Caption 6
1.4 Figure to help with interp. Explain. 11
1.5 Good visual summary of data flow. 15
1.6 Simple circuit diagram to visualise backpropogation. 17
1.7 Sigmoid circuit. 18
1.8 example circuit for interpretation. 19

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. 29
3.2 Categorisation of the taxonomy of MLL evaluation metrics 33
3.3 Categorisation of multi-label learning taxonomy (this is just an example) 54

4.1 End-to-end architecture of the proposed SRN. 76
4.2 fst 78

A.1 Schematic of the image collection process. 91

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. 93
A.3 Examples of chips with common land cover/use labels. 94
A.4 Examples of chips with less common land cover/use labels. 96
A.5 Class distribution of the labels in the training set. 97

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

$\vec{\boldsymbol{v}}$	Physical vector, see equation ...

Subscripts

a	Adiabatic
a	Coordinate

Chapter 1

Introduction

1.1 Motivation

> the core of what I want to say here is that multi-label image classification is chosen because it has many useful applications. Deep neural networks is chosen because it is the most powerful single label image classification model. The combination is also new research area with potential of contributing to literature. Make sure this is what this section communicates.

Image Classification is the task of assigning one (or more) label(s) to an input image. It is one of the core problems in Computer Vision. This seemingly simple task has a large variety of practical applications. For example, detecting deforestation in the Amazon from satellite images [1] or detecting cancer from images of skin lesions taken by a mobile phone (Esteva *et al.*, 2017). Image classification is a thoroughly researched subject and already regarded by many as a 'solved' problem. This progress is mainly attributed to the yearly large-scale image classification competition, called *ImageNet*[^imagenet], providing researchers 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*, a subfield of Machine Learning. Deep Learning or *Deep Neural Networks* (DNNs) is a class of models inspired by the structure and function of the brain called artificial neural networks. The type of DNN proven best suited for image classification problems are *Convolutional Neural Networks* (CNNs). CNNs is the uncontested state-of-the-art model for practically all image classification problems, mainly because of its ability of 'learning' highly discriminative feature representations of its input images. In the past, conventional approaches was built on carefully designed hand-crafted features such sa SIFT and BOW. AlexNet (Krizhevsky *et al.*, 2012) was the

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

first CNN to win ImageNet (in 2012) and thereafter, each of the following annual ImageNet competitions was won by a CNN.

The main focus in the fields of image classification and deep learning, until recently, was on problems where each image is only annotated with a single label. *Multi-Label Classification* (MLC) of images generalises this task to allow images to be annotated with more than one label. The majority of real-world images contain more than one object, making multi-label image classification a more general and practical problem, and naturally, also more challenging. Only recently more attention was given to multi-label image classification (for example, (Wei *et al.*, 2014)) and therefore the field is nowhere near the maturity level of its single-label counterpart. It is fair to assume that since DNNs are so powerful with single-label image classification, it may also be extended to successfully model images with multiple labels.

Extending CNNs and other DNNs to handle images with multiple labels is not a trivial task. Recently, a handful of proposals were made on how to tackle this task. However, these suggestions were mostly given in isolation of one another and no comprehensive review and comparison of these methods exist in the literature. Most of the proposals also come from a deep learning background and could gain from insights and methods already discovered in the field of MLC. In summary, the relevance of multi-label image classification, in terms of its range of useful practical applications, and the power of deep learning methods for image classification tasks, is the main motivation behind this thesis. This area of research is still relatively under explored and a comprehensive understanding and review of on the subject could lead to novel contributions to the literature. This is the secondary motivation behind the study. Exactly what this thesis aims to achieve will be discussed next.

1.2 Objectives

This thesis study will explore ways of extending the conventional single-label DNNs to make it applicable to problems of multi-label image classification. The main aim is to gain a comprehensive understanding of these methods and how they compare and relate to each other. The methodology for achieving this is as follows:

- Briefly introduce the domain of image classification.
- Provide a dense introduction to deep neural networks for image classification.
- Provide a comprehensive review of the multi-label classification framework and its common approaches.
- Review the various proposals to apply deep neural networks to multi-label image classification.
- Provide a framework in which these methods can be compared.

- Evaluate on benchmark datasets?

  these are still preliminary

The insights gained from the primary objective of this study may lead to the possibility of:

- suggesting improvements on the exsiting proposals to multi-label deep neural networks,
- propose a novel method for multi-label deep neural networks and/or
- being able to recommend some approaches above others.

These are the seconday objectives of this study.

1.3 Code and Reproducibility

All of the code for this project, including the source docoments, is made available at https://github.com/jandremarais/Thesis. More instructions on how to implement the code is contained in the file named, `README.md`, in the GitHub repository.

not sure about this section

1.4 Background and Important Concepts

It should be clear by now that the three main concepts in this thesis is that of image classification, deep neural networks and multi-label classification. Each of these areas will be introduced individually before putting it all together. DNNs will be discussed in Chapter 2 and MLC in Chapter 3. Since we are interested in applying both of these classes of models in the image classification domain, a background and comprehension of the image classification problem is necessary.

First, a brief introduction to the general problem of image classification is given. It will be explained with the help of a trivial example. This discussion will also introduce the main components of DNNs which are important to understand before moving on to Chapter 2.

1.4.1 Image Classification

There are three main tasks in computer vision (CV), namely: image classification, object detection and image segmentation. Traditional image classification is the task of assigning one label from a fixed set of categories to an input image. More recently the task has been generalised to assigning multiple labels to an input image, *i.e.* multi-label classification (MLC). First, we will only look at the single label case.

Image classification is the core of computer vision tasks and probably the most explored since it has a large variety of practical applications. It can be shown that the other two CV tasks, detection and segmentation, can be reduced to classification. Classification will be the main task of this thesis.

> show visual difference between classification, segmentation and detection.

Instead of hard coding rules on how to classify images into an image classification model, it can learn to classify images by seeing many examples of images and its corresponding labels. In this way it learns the visual appearance of each class. This is sometimes referred to as a data-driven approach. A very intuitive approach to image classification (and supervised learning in general) is called the nearest neighbour approach.

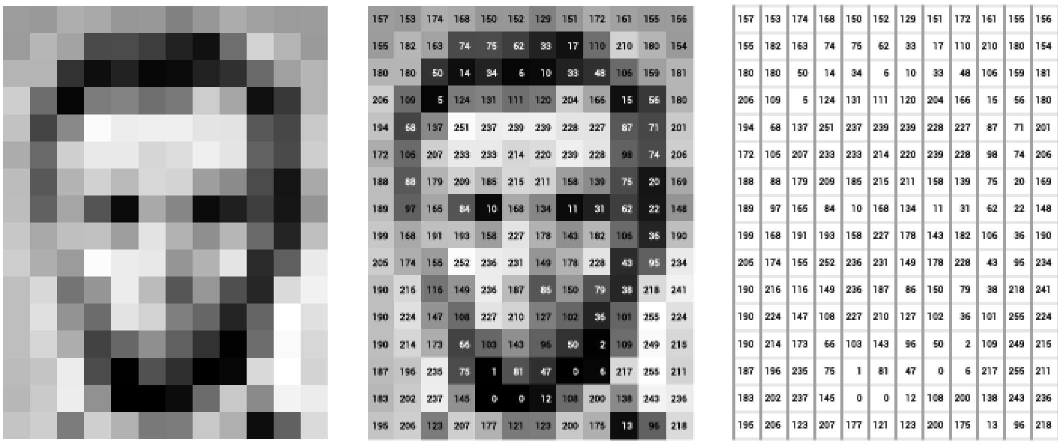
Although this approach is rarely used in practice, this description helps with the understanding of the image classification problem. The nearest neighbour classifier will take a test image, compare it to every single one of the training images, and predict its label to be the label of the closest training image. This leaves the question of how to measure the similarity between images.

- challenges for image classification: (still add this somewhere around here)
  - http://cs231n.github.io/classification/

An image is a grid of many small, square cells of 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’ (intensity) of each pixel. These integers are usually in $[0, 255]$, such that the greater the integer the brighter the pixel, *i.e.* a pixel with intensity 0 is totally black and a pixel with intensity 255 is totally white. Note that a color image consists of 3 spectral bands, red, green and blue (RGB), *i.e.* the color of one pixel is determined by 3 integers each representing the intensity of the color red, green and blue, respectively.

The (dis)similarity between two images can now be measured pixel by pixel. It is possible to represent the grayscale image mentioned above in a vector of length 32×32. Suppose a grayscale Image 1 is flattened out to be represented by the vector $\boldsymbol{I}_1 = \{I_{11}, I_{12}, \ldots, I_{1p}\}$ and similary, Image 2 by $\boldsymbol{I}_2$, where $p = 32 \times 32$. Then the dissimilarity between Image 1 and Image 2 can be calculated by the L_1-distance:

$$d_1(\boldsymbol{I}_1, \boldsymbol{I}_2) = \sum_{j=1}^{p} |I_{1j} - I_{2j}|.$$



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Figure 1.1: Greyscale intensities. http://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

$$\left| \underset{\text{test image}}{\begin{bmatrix} 56 & 32 & 10 & 18 \\ 90 & 23 & 128 & 133 \\ 24 & 26 & 178 & 200 \\ 2 & 0 & 255 & 220 \end{bmatrix}} - \underset{\text{training image}}{\begin{bmatrix} 10 & 20 & 24 & 17 \\ 8 & 10 & 89 & 100 \\ 12 & 16 & 178 & 170 \\ 4 & 32 & 233 & 112 \end{bmatrix}} \right| = \underset{\text{pixel-wise absolute value differences}}{\begin{bmatrix} 46 & 12 & 14 & 1 \\ 82 & 13 & 39 & 33 \\ 12 & 10 & 0 & 30 \\ 2 & 32 & 22 & 108 \end{bmatrix}} \rightarrow 456$$

Figure 1.2: Pixelwise difference

Now, suppose we want to predict the label of an test image a, then the nearest neighbour approach would assign the label of train image b^* to test image a if:

$$b^* = \arg\min_b d_1(\boldsymbol{I}_a, \boldsymbol{I}_b),$$

for $b = 1, 2, /dots, N$, where N is the number of training images. Of course there are other ways of measuring the dissimilarity between images. Another example would be to use the L_2-distance:

$$d_2(\boldsymbol{I}_1, \boldsymbol{I}_2) = \sqrt{\sum_{j=1}^{p} (I_{1j} - I_{2j})^2}.$$

The chosen metric depends on the use case.

The nearest neighbour approach can be generalised to use more than 1 nearest neighbour when predicting the label of a test image. This approach is called the k-Nearest Neighbours (k-NN). The only difference is that, you now search for the k (instead of just 1) images with the smallest dissimilarity with the test image and then combine the labels of these k images, either through averaging or majority voting, to predict the label of the test image. Choosing the right value of k is important and is usually done by cross-validation. See Hastie ref.

The advantage of using k-NN is that it is simple and requires no time to train. Unfortunately, when it comes to test time, the algorithm needs to calculcate the distance between the test image and all the other images in the training set, which is computationally very expensive. Also in [Haste ref], they show that k-NN suffers severely from the *curse of dimensionality* and that it is mostly only useful to classify lower dimensional objects. Images are very high-dimensional objects.

The dissimilarity measures discussed above are actually proven to be very poor in discriminating between images in an image classification problem. Images that are nearby in terms of the L_1 and L_2 distances are much more of a function of the general color distribution of the images, or the type of background rather than their semantic identity. Refer to the t-SNE figure.

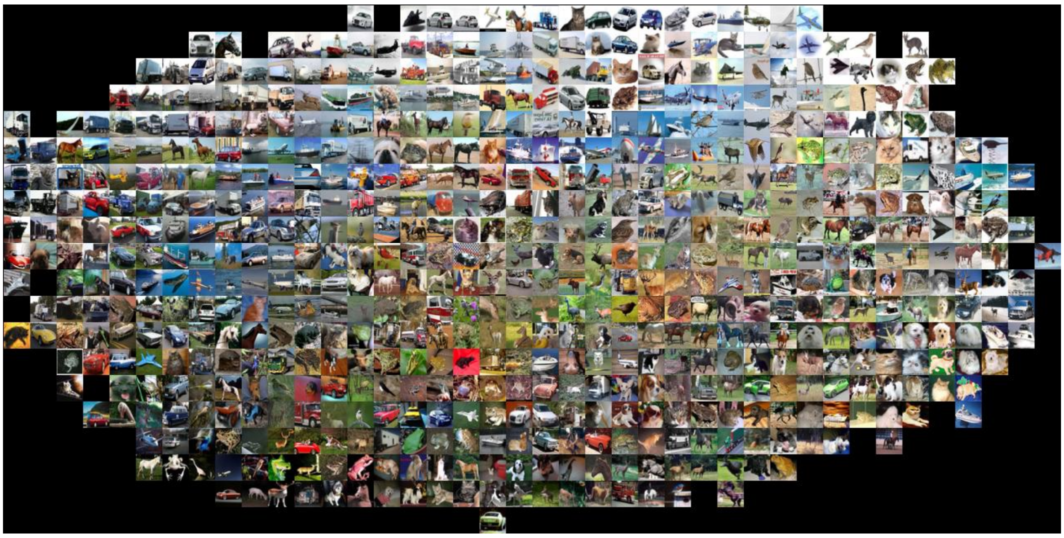


Figure 1.3: Caption

1.4.2 Score Function

The following simple approach to image classification naturally extends to neural networks and convolutional neural networks and is therefore very important to comprehend. This approach has to major components: a score function and a loss function. The score function maps raw data (*e.g.* an image) to a set of class scores, and a loss function quantifies the agreement between the predicted class scores and the actual ground truth labels associated with the raw data. This approach can then be described as an optimization problem in which the minimisation of the loss function with respect to the parameters of the score function is the main goal.

Some notation is needed to formally define this approach. Suppose we have N training images $\boldsymbol{x}_i \in \mathbb{R}^p$ each associated with a label $y_i \in \{1, 2, \ldots, K\}$, where $i = 1, 2, \ldots, N$ and K is the number of possible categories an image can belong to and p the number of pixels of each image. The score function is then defined as the function f that maps the raw image pixels to class scores:

$$f : \mathbb{R}^p \to \mathbb{R}^K.$$

The simplest possible score function is a linear mapping:

$$f(\boldsymbol{x}_i, W, b) = W\boldsymbol{x}_i + \boldsymbol{b}.$$

In the above equation, Image i is flattened out to be represented by a p-dimensional vector. The paramters of f are the matrix $W : K \times p$ and the vector $\boldsymbol{b}$, often called the weights and biases, respectively. These terms are comparable to the coefficient and constant terms in a statistical linear model and thus should not be confused with bias in the statistical sense.

We assume the pairs $(\boldsymbol{x}_i, y_i)$ to be fixed, but we do have control over the W and $\boldsymbol{b}$ terms. Our goal will be to set these in such a way so that the computed class scores for each image in the training set match the associated ground truth label as close as possible. What we have described thus far is very similar to the approach taken by convolutional neural networks, but instead the function, f, which maps the raw pixels to class scores, is much more complicated with plenty more parameters to tune.

Notice that this score function determines the score for each class as a weighted sum of the pixel values across all 3 of its spectral bands. We would imagine that a linear classifier trained to classify, say, ships would have a weight matrix that assigns heavier weights to blue pixels on the sides of an image, which loosely corresponds to water.

If we picture the images as points in a high-dimensional space, f is a hyperplane, W determines the angle of the hyperplane and $\boldsymbol{b}$ translates the hyperplane through the space. Another interpretation of this linear classifier is that each row of the weight matrix is a so-called template for the corresponding class. The linear classifier matches the input image with each of the class templates in W by calculating a dot product. A high class score would translate to a higher similarity between the input image and the class template. This interpretation is closely related to the nearest neighbour approach, but here only the test image's distance (here the negative of the inner product) to each of the K class templates are calculated instead of its distance to each of the N images in the training set.

Later on it becomes too cumbersome to keep track of two sets of parameters, W and $\boldsymbol{b}$, and therefore, for the rest of the thesis we will write the linear classifier as:

$$f(\boldsymbol{x}_i, W) = W\boldsymbol{x}_i,$$

where $\boldsymbol{b}$ is now contained in the last(/first?) column of W and the last element of $\boldsymbol{x}_i$ is now the constant, 1. This is the so-called bias trick.

Note that thus far we have used raw pixel values in the range of $[0, 255]$ as input. However, in practice, it is more common to subject the input images to some preprocessing before inputting them into the score function. The benefits of this will be made clear in the optimisation section. Common preprocessing techniques are the centering and scaling of the pixels so that their values lie in the range of $[-1, 1]$. To center the input image, is to calculate a *mean image* from the training images and subtract each of its pixel values to from the corresponding pixel values of each image in the training set. This is identical to zero mean centering for standard statistical learning tasks - each pixel is seen as in input feature. Scaling is done by dividing each pixel by a function of its variance accross the whole training set.

1.4.3 Loss Function

To evaluate the agreement between the score function and the ground truth labels, we need a loss function. A loss function, also known as the cost function or the objective, is high when the score function does a poor job of mapping the input images to the class scores, and low when it does so accurately. There are multiple ways of defining such a loss function.

1.4.3.1 Multiclass Support Vector Machine Loss

A commonly used loss is the Multiclass Support Vector Machine (SVM) loss. In statistical learning this is more commonly known as the Hinge Loss. The SVM loss is designed in such a way that it wants the correct class for each image to have a score higher than the incorrect classes by some fixed margin Δ. More precisely, the multiclass SVM loss for the i-th example with label y_i can be given by:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta),$$

where $s_j = f(x_i, W)_j$ is the score for the j-th class computed for image i. Here L_i consists of $K - 1$ components, each representing an incorrect class. A component will make no contribution to the loss if the calculated class score for the corresponding incorrect class is less than the correct class score by a margin of Δ, *i.e.* $s_{y_i} - s_j > \Delta$. It will make a positive contribution otherwise. As an example, suppose we have three predicted class scores for an image $s = [4, 5, -3]$ and that the second class is the true label. Let $\Delta = 2$. The loss computed for this image will then consist of 2 components:

$$
\begin{aligned}
L_i &= \max(0, 4-5+2) + \max(0, -3-5+2) \\
&= 1 + 0
\end{aligned}
$$

We see that although the predicted class score for class 1 was smaller that the predicted class score for the true label, class 2, it was still within a margin of $\Delta = 2$ and there had a positive contribution to the loss. The predicted class score for class 3 was far lower than predicted class score for the true label and therefore did not make any contibution to the loss. In summary, the SVM loss function wants the score of the correct class to be larger than the incorrect class scores by at least Δ, if not, we will accumulate a loss.

Note that the loss is typically evaluated on a set of images and not just one, as we have describe thus far. The average loss of a set with N images can be written as $L = \frac{1}{N}\sum_{i=1}^{N} L_i$. Another variation of the SVM loss is to replace the $\max(0,\cdot)$ term with the term, $\max(0,\cdot)^2$, which results in the squared hinge loss or the L_2-SVM loss. This penalises violated margins more heavily and may work better in some cases. [https://arxiv.org/abs/1306.0239]

There is still one problem with the SVM loss described thus far. Suppose we have found a weight matrix W that correctly classifies all input images and by the correct margins, *i.e.* $L_i = 0$, $\forall i$, then setting the weight matrix to λW, for $\lambda > 1$ will have the same solution. This means the solution to the optimisation problem is not unique. It would make the optimisation task easier if we could remove this ambiguity. This can be done by adding a penalty term to the loss function, also know as regularisation. The mosty common regularisation penalty, $R(W)$, is the L_2-norm:

$$R(W) = \sum_k \sum_l W_{k,l}^2,$$

which is simply the sum of the squared elements of the weight matrix. The full SVM loss can now be defined as:

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W).$$

The two components of the loss can be called the *data loss* and the *regularisation loss*. λ determines how much regularisation should be done. If λ is large, more regularisation will take place. The value of λ is typically determined through cross-validation.

The regularisation penalty ensures a unique (or less solutions?) solution to the optimisation problem by restricting the weight parameters in size. Greater weight parameters will result in bigger loss, if everything else remain constant. Another appealing property is that penalising large weights tends to improve generalisation, because it means that no input dimension can have a very large inlfuence on the scores all by itself.

Typically, only the weight parameters are regularised, since the bias terms do not control the strength of influence of an input dimension. Howerver, in practice the often turns out to have a negligible effect.

To return to the value of Δ - it turns out that Δ and λ control the same trade-off and therefore we can safely set $\Delta = 1$ and only use cross-validation for determining λ. This might not seem obvious, but the key to understanding this is to realise that the weights in W have a direct influence on the class scores and therefore also on the differences between them. If all the elements in W are shrinked, all the differences in class scores will shrink and if all the elements are scaled up, the opposite will happen. Therefore, the margin Δ becomes meaningless in the sense that the weights can shrink or stretch to match Δ. Thus the only real trade-off is how large we allow the weights to be and this we specify through λ.

1.4.3.2 Softmax Classifier

The linear classifier combined with the SVM loss we call the SVM classifier. We will now look at the Softmax Classifier, which is the linear classifier combined with a different loss function. In statistics, the softmax classifier is better known as the multiclass logistic regressor. The biggest difference between the SVM classifier and the softmax classifier is that the latter gives a slightly more intuitive ouput in the form of normalised class probabilities, instead of the uncalibrated and less interpretable output of the SVM classifier. The loss function used for the softmax classifier is the *cross-entropy loss*:

$$\begin{aligned} L_i &= -\log \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \\ &= -f_{y_i} + \log \sum_j e^{f_j}. \end{aligned}$$

As before, the full loss is the mean of L_i over the whole dataset with an additional regularisation penalty.

To see where this loss function comes from, first consider the softmax function:

$$h_j(\boldsymbol{z}) = \frac{e^{z_j}}{\sum_k e^{z_k}}.$$

$h_j(\boldsymbol{z})$ squeezes the elements of the real-valued vector, $\boldsymbol{z}$, to fit in the range of $[0, 1]$ and that their sum always add to 1. Now, in information theory, the cross-entropy between a ‘true’ distribution p and an estimated distribution q is defined as:

$$H(p, q) = -\sum_x p(x) \log q(x).$$

Consider the case where the 'true' distribution, p, is a vector of zeros except at the y_i-th position, where the value is 1, and the estimated distribution, q, is the estimated class probabilities, $q = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$. Clearly, $H(p,q)$ then simplifies to L_i. Thus the softmax classifier minimises the cross-entropy between the estimate class probabilities and the true distribution.

In the probabilistic interpretation of this classifier, we are minimising the negative log likelihood of the correct class, which can be interpreted as performing *maximum likelihood estimation* (MLE). From this view, the term $R(W)$ can be interpreted as coming from a Gaussian prior over the weight matrix, W, where instead of MLE we are performing *maximum a posteriori*.

To be clear, the softmax classifier interprets the scores computed by f to be the unnormalised log probabilities. Therefore, it undergoes the exponentiating and division (to become the normalized probabilities) before being used as input the cross-entropy loss.

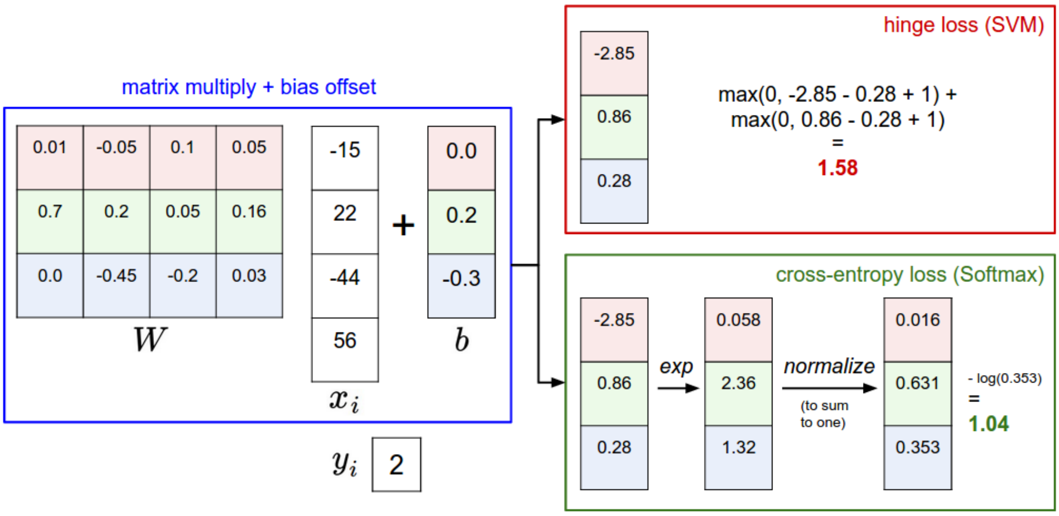


Figure 1.4: Figure to help with interp. Explain.

Note that although we used the term 'probabilities' to describe the output the softmax classifier, these are not probabilities in the statistical sense. They do sum to 1 and are in the range of $[0,1]$, but they are still technically confidence scores rather than probabilities, *i.e.* their order is interpretable but not their absolute values. The reason for this is that they depend heavily on the regularisation strength determined by λ. The higher λ is, the more uniform the probabilities become.

SVM and Softmax comparable. See http://cs231n.github.io/linear-classify/

> remember, only single label multiclass classification has been considered thus far and that some of these do not hold for multilabel classification.

1.4.4 Optimisation

From the previous sections we learned that the key components for the image classification task is the score function and the loss function. We looked at the linear mapping of raw pixel values to class scores and various loss functions, such as the hinge loss and cross-entropy loss, to evaluate the mapping against the ground truth labels. Putting all of this together, the SVM classifier can be reduced to the problem of minimising the loss:

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [\max(0, f(\boldsymbol{x}_i; W)_j - f(\boldsymbol{x}_i; W)_{y_i} + 1)] + \alpha R(W),$$

where $f(\boldsymbol{x}_i; W) = W\boldsymbol{x}_i$. This process of minimising the loss is also known as optimisation, which is the third key component. Optimisation is the process of finding the set of parameters W that minimise the loss function.

Once we get to convolutional neural networks, the only major difference is the use of a more complicated score function. The loss and optimisation components remain mostly unchanged.

> Visualise a loss function in 2-dimensions to give idea of how it looks. [http://cs231n.github.io/optimization-1/]
>
> SVM classifier has a convex loss function. Whole research field in convex optimisation. When we get to more complex neural netorks, the loss becomes non-convex.
>
> The loss functions we use are technnically non-differentiable, since there are 'kinks' in the loss function (gradients not define everywhere). However, the subgradient still exists and is commonly used instead. [https://en.wikipedia.org/wiki/Subderivative]

For this discussion on how to minimise the loss function with respect to W, we will use the SVM loss. The methods discussed may seem odd, since it is a convex optimisation problem. We only use this example for simplicity, since when we get to complex neural networks, the optimisation will not be a convex problem.

The core idea of this approach to minimise the loss with respect to W is that of iterative refinement - start with a random W and then iteratively refining it to get a lower loss. Finding the best set of weights, W is hard, but the problem of refining a specific set of weights to only be slightly better, is much easier.

A helpful analogy is that of the blindfolded hiker, who is on a hilly terrain, trying to reach the bottom. The height of the terrain represents the loss achieved. A possible strategy for the hiker to reach the bottom would be to test a step into a random direction and the only take the step if it leads downhill.

In optimisation terms, we can start with a random initialisation of W, generate random perturbations δW to it and if the loss ath the perturbed $W + \delta W$ is lower, we will perform an update. This approach is better than a random search of W but still inefficient and computationally expensive.

It turns out that it is actually not necessary to randomly search for a good direction to move towards. The best direction can be determined mathematically. This best direction along which the weights should change corresponds to the direction of steepest descend and is related to the gradient of the loss function. In the hiking analogy, this approach roughly corresponds to feeling the slope of the hill below our feet and stepping down the direction that feels the steepest.

In one-dimensional functions, the slope is the instantaneous rate of change of the function at any specified point. The gradient is a generalisation of slope for multi-dimensional functions and is simply a vector of slopes, better known as derivatives, for each dimension in the search space. Mathematically, the expression for the derivative of a 1-dimensional function with respect to its input is:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

When the function of interest take a vector of numbers instead of a single number, we call the derivatives partial derivatives. The gradient is simply the vector of partial derivatives in each dimension.

There are two approaches to computing the gradient: the **numerical gradient** and the **analytic gradient**. Their pro's and con's are discussed in the following section.

1.4.4.1 Computing the Gradient Numerically

Iterate over all dimensions one by one, make a small change, h, along that dimension and calculate the partial derivative of the loss function along that dimension by seeing how much the function changed. Ideally, we want h to be as small as possible, since the mathematical formulation requires $h \to 0$. In practice it often works better to compute the numeric gradient using the centered difference formula: $\frac{f(x+h)-f(x-h)}{2h}$.

Note that the update of W should be made in the negative direction of the gradient, since we wish to decrease the loss function.

The gradient tells us the direction in which the function has the steepest rate of increase, but it does not tell us how far along this direction we should step, *i.e.* what is the value of the step size? This value is also known as the *learning rate* and we will soon learn that it is one of the most important hyperparameters of a neural network. Choosing a small step size in the direction of steepest descent will ensure consistent but slow progress. A large step in this direction may lead to a quicker descent but also has the risk of overshooting the optimal point.

The obvious downfall of this approach (in addition to that is only an approximation) is that we need to calculate the gradient in each direction/dimension. Neural networks have millions of parameters and therefore optimising them in this manner is clearly not feasible.

1.4.4.2 Computing the Gradient Analytically

The second way to compute the gradient is analytically using Calculus. A direct formula for the gradient can be derived and it also very fast to compute. This approach is more error prone to implement which is why in practice it is very common to perform a *gradient check*, which is the comparision of the analytic gradient to the numeric gradient to chech the correctness of the implementation.

By using the SVM loss for a single data point as an example:

$$L_i = \sum_{j \neq y_i} \left[\max(0, \boldsymbol{w}_j^T \boldsymbol{x}_i - \boldsymbol{w}_{y_i}^T \boldsymbol{x}_i + \Delta) \right].$$

Now, we want to differentiate the function with respect to the weights. Taking the gradient *w.r.t.* $\boldsymbol{w}_{y_i}$, gives:

$$\nabla_{\boldsymbol{w}_{y_i}} L_i = - \left(\sum_{j \neq y_i} \mathbb{I}(\boldsymbol{w}_j^T \boldsymbol{x}_i - \boldsymbol{w}_{y_i}^T \boldsymbol{x}_i + \Delta > 0) \right) \boldsymbol{x}_i,$$

where $\mathbb{I}$ is the indicator function. This is simply the data vector scaled by the negative of the number of classes scores that did not meet the desired margin. The gradient with respect to the other rows of W where $j \neq y_i$ is:

$$\nabla_{\boldsymbol{w}_j} L_i = \mathbb{I}(\boldsymbol{w}_j^T \boldsymbol{x}_i - \boldsymbol{w}_{y_i}^T \boldsymbol{x}_i + \Delta > 0) \boldsymbol{x}_i.$$

Determining these equations are the tricky part. Once this is done, it is easy to implement the expressions and use them to perform gradient updates.

1.4.4.3 Gradient Descent

The procedure of repeatedly evaluating the gradient and then performing a parameter update is called *gradient descent*. This is by far the most common and established way of optimising neural network loss functions. Although there are some ‘bells and whistles’ to add to this algorithm, the core ideas remains the same when optimising neural networks.

One of the advantages of gradient descent is that a weight update can be made by only evaluating the gradient over a subset of the data, called *mini-batch gradient descent*. This is extremely helpful for large-scale applications, which are almost the norm for Deep Learning, since it is not necessary to compute the full loss function over the entire dataset. This leads to faster convergence and allows for the processing of large datasets that are too big to fit into a computer's memory. A typical batch consists of 64/128/256 data points, but it depends on the computational power at hand. The gradient computed using a mini-batch is only an approximation of the gradient of the full loss. This seems to be sufficient in practice since the data points/images are correlated.

The specification of the mini-batch size is not very important and is usually determined based on memory constraints. Usually they are in powers of two, because in practice many vectorised operation implementations work faster when ther inputs are sized in powers of 2. 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, this is much less common, since it is more efficient to calculate the gradient in larger batches compared to only using one example. However, it is still widely acceptable to use the term SGD even though you are referring mini-batch gradient descent. This is actually the norm.

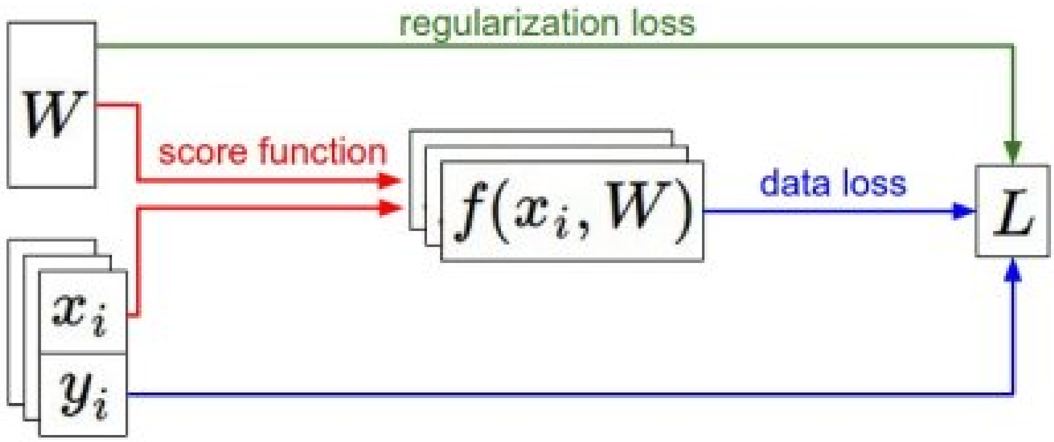


Figure 1.5: Good visual summary of data flow.

1.4.4.4 Backpropagation

Way of computing gradients of expressions through recursive application of the chain rule. Critical to understanding the optimisation of neural networks.

The core problem for this section is: We are given some function $f(\boldsymbol{x})$, where $\boldsymbol{x}$ is a vector of inputs, and we are interested in computing the gradient of f at $\boldsymbol{x}$, *i.e.* $\nabla f(\boldsymbol{x})$. In our case, f corresponds to the loss function (*e.g.* SVM loss) and the inputs $\boldsymbol{x}$ will consists of the training data and the neural network weights.

Consider this simple example to introduce some of the conventions. Suppose we have the following function $f(x,y) = xy$. The partial derivative for either input is then:

$$\frac{\partial f}{\partial x} = y,$$
$$\frac{\partial f}{\partial y} = x$$

These indicate the rate of change of f with respect to x and y respectively surrounding an infinitelly samll region near a particular point. For example, if $x = 2$ and $y = -5$, then $f(x,y) = -10$. The derivative on x is -5, which tells us that if we were to increase the value of x by a tiny amount, the effect on the whole expression would be to decrease by 5 times that amount.

As used before, the vector of partial derivatives is called the gradiemt, ∇f. So for the previous simple example we have $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] = [y, x]$. What follows are another two simple examples that will prove to be useful in later discussions.

For $f(x,y) = x + y$, $\nabla f = [1,1]$, and if $f(x,y) = \max(x,y)$, then $\nabla f = [\mathbb{I}(x \geq y), \mathbb{I}(y \geq x)]$. Technically, *nablaf* for the latter function is called a subgradient, since the derivative for $\max(x,y)$ is not defined everywhere (?).

1.4.4.5 Compound Expressions with the Chain Rule

Now for the calculating of a more complicated expression, we will use the chain rule. Consider the expression $f(x,y,z) = (x+y)z$. Note that this expression can be decomposed into two expressions: $q = x + y$ and $f = qz$. From the previous simpler examples, we saw how to calculalte the gradient for these simple expression of addition and multiplication separately. But what we are really interested in is how to calculate the gradient of f *w.r.t.* its inputs, x, y, z. This can be done using the *chain rule*. According to the chain rule, $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$, and similarly for $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial z}$. This can be viewed as the simplest form of backpropogation.

Suppose we want to compute the gradient at inputs $x = -2$, $y = 5$ and $z = -4$. First, we make a *foreward pass* to compute the outputs from the given inputs, *i.e.* $q = 3$ and then $f = -12$. These values are shown in green in the circuit diagram. The following step is to make a *backward pass* (backpropogation), which is to start at the end and recursively apply the chain rule to copmute the gradients, shown in red in the circuit diagram, all the way to the inputs of the circuit. In the example, $\frac{\partial f}{\partial f} = 1$, $\frac{\partial f}{\partial z} = 3$, $\frac{\partial f}{\partial q} = -4$, $\frac{\partial f}{\partial x} = -4$ and $\frac{\partial f}{\partial y} = -4$. The gradients can be thought of as flowing backwards through the circuit.

Each circle in the diagram can be referred to as a gate. Notice that every gate (the addition gate $(+)$ and the multiplication gate $(*)$) gets some inputs and can right away compute its output value and the local gradient of its inputs

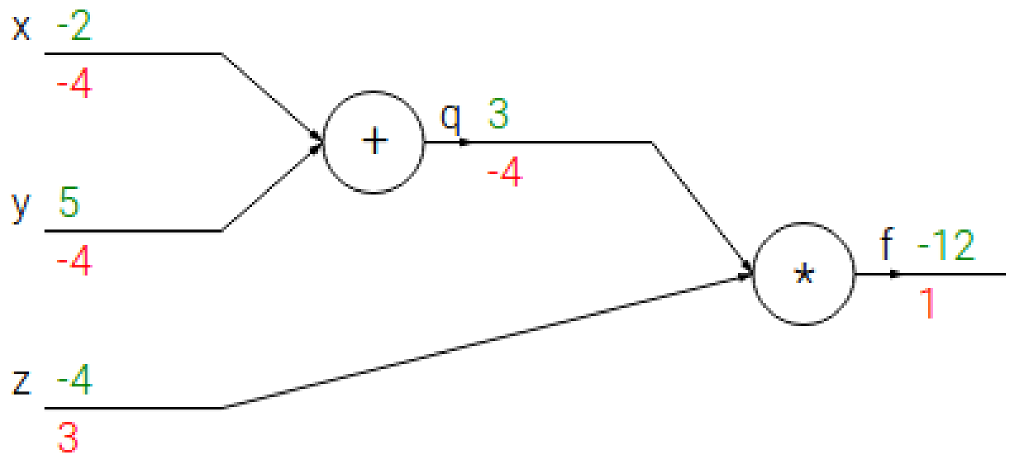


Figure 1.6: Simple circuit diagram to visualise backpropogation.

with respect to its output value. This is done completely indpendently without being aware of any of the details of the full circuit that they are embedded in. However, during backpropogation the gate will eventually learn about the gradient of its ouput value on the final output of the entire circuit. According to the chain rul, the gate should take that gradient and multiply it into every gradient it normally computes for all of its inputs. Let us look at the example again to make this clear.

The (+) gate received inputs $[2, -5]$ and computed output 3. It also computed its local gradient with respect to both of its inputs, which is 1, since it is an addition operation. The rest of the circuit copmuted the final value to be -12. During the backward pass, the (+) gate learns that the gradient for its output was -4. It then takes that gradient and multiplies it to all of the local gradients for its inputs, which results in -4 and -4. This implies that if x, y were to decrease (responding to their negative gradients) then the (+) gate's output would decrease, which in turn makes the $(*)$ gate's output increase. Thus backpropogation can be thought of as gates communicating to each other through the gradient signal whether they want their outputs to increase or decrease, so as to make the final output higher.

1.4.4.6 Modularity

We introduced addition gates and multiplication gates, but any kind of differentiable function can act as a gate. We can also group multiple gates into a single gate or decompose a function into multiple gates whenever it is convenient. Consider the following expression to illustrate this:

$$f(\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}.$$

This function is actualy a common piece in a neural network, but for now we can view it as mapping from inputs $\boldsymbol{x}, \boldsymbol{w}$ to a single number. The function is made up of multiple gates, and aside from the ones already discussed (addition, multiplication and max), they are:

$$
\begin{aligned}
f(x) &= \frac{1}{x} && \Longrightarrow \frac{df}{dx} = -\frac{1}{x^2} \\
f_c(x) &= c + x && \Longrightarrow \frac{df}{dx} = 1 \\
f(x) &= e^x && \Longrightarrow \frac{df}{dx} = e^x \\
f_a(x) &= ax && \Longrightarrow \frac{df}{dx} = a
\end{aligned}
$$

where c, a are constants. The full circuit for this expression is then:

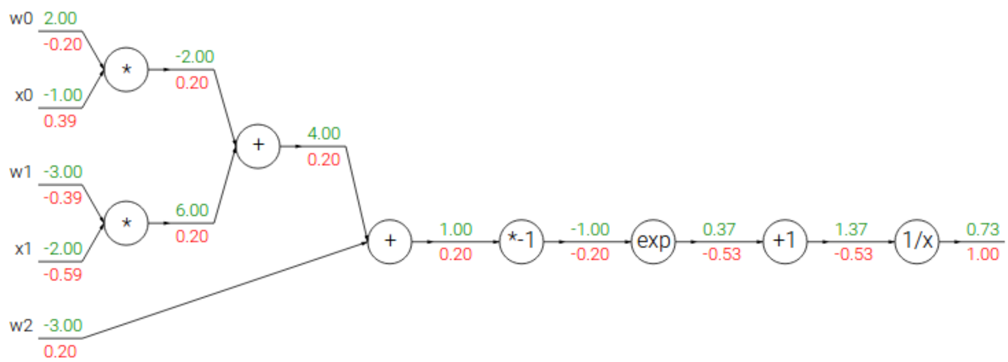


Figure 1.7: Sigmoid circuit.

The long chain of functions (gates) on the dot product of $\boldsymbol{x}$ and $\boldsymbol{w}$ is the decomposition of the *sigmoid function*:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

The derivative of the sigmoid function simplifies to a very convenient expression:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}}\right)\left(\frac{1}{1 + e^{-x}}\right) = (1 - \sigma(x))\sigma(x).$$

Therefore in any real practical application it would be very useful to group the operations of the tail chain into a single gate.

1.4.4.7 Patterns in Backward Flow

It is interesting to note that in many cases the backward-flowing gradient can be interpreted on an intuitive level. Take the three most commonly used gates in neural networks, (add, mul, max), as an example. All of them have very simple interpretations in terms of how they act during backpropogation. Consider the following example circuit:

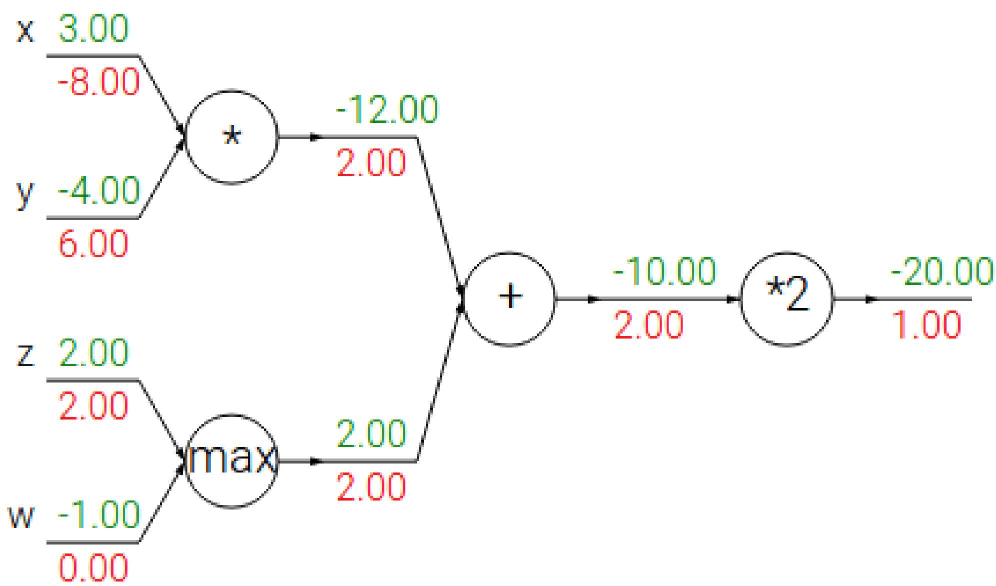


Figure 1.8: example circuit for interpretation.

From the diagram above, the following patterns should be clear:

- The add gate always takes the gradient on its output and distributes it equally to all of its inputs, regardless of what their values were during the forward pass. This is because the local gradient for the add operation is always 1 for all its inputs.
- The max gate routes the gradient to exactly one of its inputs, the input that had the highest value during the forward pass. This because the local gradient for a max gate is 1 for the highest value and 0 for all other values.
- The multiply gate switches the gradients of its inputs and then multiply it by its output gradient.

Notice that if one of the inputs to the multiply gate is very small and the other is very big, then the multiply gate will do something slightly unintuitve:

it will assign a relatively huge gradient to the small input and a tiny gradient to the large input. This is good to know, since in linear classifiers where the weights are multiplied by the inputs, it means that if the inputs are multiplied by a 1000, then the gradient on the weights will be 1000 times larger and you would have to lower the learning rate by that factor to compensate. This show how important preprocessing is for the optimisation of a classifier.

The above sections were concerned with single variables, but all concepts extend in a straight-forward manner to matrix and vector operations. However, one must pay closer attention to dimensions and transpose operations.

- http://cs231n.github.io/

1.5 Outline

The motivations and objectives of the thesis has been discussed and the problem of image classification has been introduced. The rest of the thesis will follow the following outline. Chapter 2 introduces deep neural networks for image classifcation. Chapter 3 reviews the field of multi-label classification. The objective of this chapter is to find approaches that could help extend DNNs for MLC. Chapter 4 reviews approaches to deep learning for multi-label image classification. Chapter 5 evaluates proposals on benchmark datasets. The results will also be compared to the findings in the literature. The thesis is concluded in Chapter 6 with a summary of the work done in this project, general discussion of the results and literature and what directions can be followed for future research.

this is super rough. give more detail

Chapter 2

Deep Neural Networks

2.1 Introduction

In chapter 1 we have introduced all of the basic components for building a Neural Network. Recall the linear classifier that mapped the inputs, $\boldsymbol{x}$, to a vector of class scores, $\boldsymbol{s}$, $\boldsymbol{s} = W\boldsymbol{x}$, where W is a matrx of weights. Here W has K rows and p columns corresponding to the number of classes and size of the inputs respectively. As mentioned in chapter 1, a Neural Network has a more complicated mapping from the inputs to the class scores. An example Neural Network would instead have a mapping like, $\boldsymbol{s} = W_2 \max(0, W_1\boldsymbol{x})$. This time, for example, W_1 could be a matrix transforming the inputs to a 100-dimensional vector, thus of size $100 \times p$. The function $\max(0, \cdot)$ introduces a non-linearity that is applied element-wise. It simply thresholds all values below zero to zero. There are several types of non-linearities that can be applied, but this is a common choice. Finally, W_2, is a matrix of size $K \times 100$, mapping the intermediate vector to the final class scores. The key difference here is the non-linearity. If it is left out, the two matrices could be collapsed into one and therefore the predicted class scores would again be a linear function of the input. W_1, W_2 is learned through stochastic gradient descent (SGD), their gradients are derived with the chain rule and computed with backpropogation.

The above mapping is an example of a two-layer network. A three-layer neural network may look something like this:

$$\boldsymbol{s} = W_3 \max(0, W_2 \max(0, W_1\boldsymbol{x})).$$

Now there are two non-linearities and W_1, W_2, W_3 are all parameters to be learned. Their sizes, which determine the size of the intermediate layers (vectors), are seen as hyperparameters and how they can be determined will be discussed shortly. In the next section we will show where the name, Neural Networks, come from and ...

2.2 Biological Motivation and Connections

Originally primarily inspired by the goal of modelling biological neural systems, but has since diverged and become a matter of engineering and achieving good results in Machine Learning tasks.

Coarse model of biological neural systems and how they can be modelled.

2.3 Common Activation Functions

2.3.1 Sigmoid

2.3.2 Tanh

2.3.3 ReLu

- Leaky ReLu

2.3.4 Maxout

- mention where we will discuss ELU and SELU

2.4 Architectures

Layerwise organisation, naming conventions, size

Representational power -> universal approximators.

More on size, overfitting and generalisation, regularisation

-
  - https://arxiv.org/abs/1706.01350

2.5 Setup

2.5.1 Data Preprocessing

- mean subtraction
- normalisation
- pca and whitening
- leakage

2.5.2 Weight Initialisation

- all zero
- small random
- calibrating the variances

- sparse initialisation
- initialising biases

2.5.3 Batch Normalisation

- https://arxiv.org/abs/1502.03167

2.5.4 Regularisation

- L1
- L2
- maxnorm
- Dropout: http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf, http://papers.nips.cc/paper/4882-dropout-training-as-adaptive-regularization.pdf
- noise in forward pass
- bias regularisation
- per layerregularisation

2.6 Loss Functions

2.6.1 Classification

2.6.2 Attrubute Classification

2.6.3 Regression

2.6.4 Structered Prediction

2.7 Learning

Process of learning the parameters and finding good hyperparameters.

- practical tips for learning

2.7.1 Monitoring

- loss function + learning rate
- train/val acc
- ration of weights

- activation per layer
- first layer viz

2.7.2 Parameter Updates

- momentum
- Nesterov
- decay
- adaptive: adagrad, rmsprop, adam
- something on cyclical?
- yellowfin: https://arxiv.org/abs/1706.03471

2.7.3 Freezing Layers

- https://arxiv.org/abs/1706.04983

2.8 Hyperparameter Optimisation

2.9 Evaluation

2.9.1 Ensembles

- same model diff initialisations
- top models through cv
- different checkpoints
- running average of parameters
- the paper on ensembling from cyclical minima
- maybe tta
- Pseudo-Labelling and Knowledge-Distillation

2.10 ConvNet Layers

2.10.1 Convolutional Layer

2.10.2 Pooling Layer

- mixed pool: https://pdfs.semanticscholar.org/de66/4f22dd4c7b4c15ac4a52513004aee55765ff.pdf and https://arxiv.org/pdf/1509.08985.pdf, can try implementation from https://github.com/fchollet/keras/issues/2816
- maxout?

2.10.3 Normalisation Layer

2.10.4 Fully Connected Layer

- also optiion to replace with conv layer

2.11 ConvNet Architectures

2.11.1 Layer Patterns

2.11.2 Layer Sizing Patterns

2.11.3 Famous Architectures

- AlexNet
- VGG

2.11.3.1 Residual Networks

Residual Networks were shown to be able to scale up to thousands of layers and still have improving performance. However, each fraction of performance improvement is at a cost of almost double the depth. Therefore training these deep residual networks has a problem of diminishing feature reuse, which make them very slow to train. (Zagoruyko and Komodakis, 2016) showed that wide, shallow networks work significantly better than their deeper variants, in terms of accuracy and training time.

See code at: https://github.com/titu1994/Wide-Residual-Networks or https://github.com/asmith26/wide_resnets_keras

- DenseNet
- Inception (?)
- Dirac Nets (?)

2.12 Visualizing CNN's

- https://github.com/raghakot/keras-vis
- http://yosinski.com/deepvis

2.12.1 Activations and First Layer Weights

2.12.2 Images with Maximum Activation

2.12.3 t-SNE Embedding

2.12.4 Occluding

- more resources: http://cs231n.github.io/understanding-cnn/

2.13 Transfer Learning

2.13.1 Feature Extractor

2.13.2 Fine-Tuning

2.13.3 Pretrained Models

- probably also something on attention mechanisms for understanding SRNs:
  - Show, attend and tell: Neural image caption generation with visual attention
  - Stacked attention networks for image question answering.
  - Learning transferrable knowledge for semantic segmentation with deep convolutional neural network

Chapter 3

Multi-Label Classification

3.1 Introduction

Multi-label (ML) learning belongs to the supervised learning paradigm and can be viewed as a generalisation of the traditional single-label learning 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 belonging 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 ML learner that can assign a set of labels to an observation. 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 learner aims to find which single label l_k, $k = 1, 2, \ldots, K$, belongs to a given observation, a ML learner is capable of assigning a set of labels $L \subseteq \mathcal{L}$ to the observation.

According to (Zhang and Zhou, 2014), ML learning can be considered a sub-problem of a wider framework, called multi-target learning, covering all problems where an observation is associated with multiple outputs. When the output variables are binary, it is a ML learning problem. But problems also exist where the output variables are multi-class or numerical and in these settings the problems are respectively known as multi-dimensional learning and multi-output regression. It is also possible that the output variables are combinations of the aforementioned types.

The birth of the ML field (around 1999) came from the need to assign multiple labels to text documents. Contributions in (Schapire and Singer, 1999) and (**?**) adapted a boosting algorithm to handle ML data. (**?**) defined a ranking based SVM to deal with ML problems in the areas of text mining and also bioinformatics. (Lewis *et al.*, 2004) released an important benchmark collection for ML text classification. Another highly cited ML SVM implementation is (Boutell *et al.*, 2004), with application in scene/image classification. (Zhang *et al.*, 2006) showed how to apply neural networks to a ML problem and (Zhang and Zhou, 2007) adapted the KNN algorithm for ML input. The first overview on the subject was given in (Tsoumakas and Katakis, 2007) where the author discussed the most relevant ML learning approaches. Then came applications to music, (Trohidis and Kalliris, 2008) and (Turnbull *et al.*, 2008). (Vens *et al.*, 2008) showed how to use decision trees for hierarchical ML classification. Important papers introducing unique ML approaches are (**?**), (Fürnkranz *et al.*, 2008) and [Read2011a]. A crucial step for ML learning was to make it accesible and useable to more reasearchers. The authors of (Tsoumakas *et al.*, 2011) developed a Java library for ML learning. Later on (**?**) did a empirical study on the most important ML algorithms up to that date, comparing 12 multi-label learning methods using 16 evaluation measures over 11 benchmark datasets. More recent extensive reviews of ML learning are given in (Zhang and Zhou, 2014) and (Gibaja and Ventura, 2014).

not sure if I want the above paragraph

The rapid growth of the ML learning (see Figure 3.1) is probably owed to the vast and expanding range of ML application domains, the biggest being text and multimedia categorisation especially those generated and/or stored on the web. Other application domains common to ML learning 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). We are interested in applying ML to the image classification domain, which will be the main focus of this chapter. Approaches in other domains will also be looked at if they are transferable to our application.

The key challenge in ML learning is to exploit dependecies amongst labels, *e.g.* using the information on the relevance of label l_i to predict label l_j, $i, j \in \{1, 2, \ldots, K\}$, $i \neq j$. This is especially difficult for a ML learner when there are many labels. Fortunately, this is not the case whith our dataset, which only has a total of 17 labels. However, it is not uncommon for ML datasets to have hundreds of thousands of labels. Proof of this can be found at this ML data repository or the recent YouTube Video Classification Challenge (Abu-El-Haija *et al.*, 2016) also hosted on Kaggle. Algorithms that can accurately and efficiently model label dependence on these datasets are rare (Sorower). This is a focus area of recent ML 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.

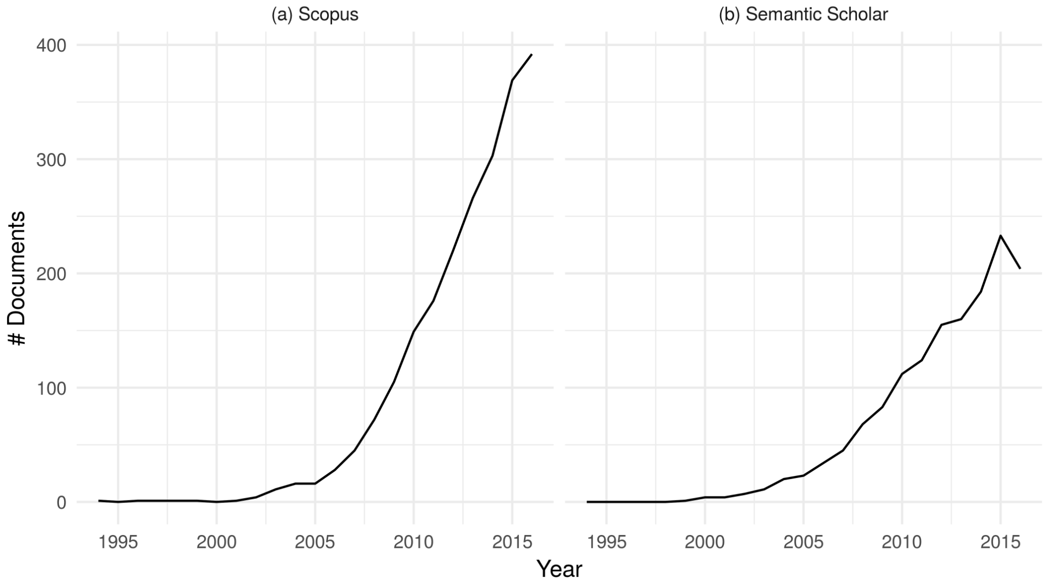


Figure 3.1: Line graphs illustrating the rise in multi-label learning publications per year for two databases. The database searches were done on 24-03-2017. The searches were not identical since they were limited to the search features of the databases. (a) The search on Scopus (cite) was for all documents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

As should be expected, the ML framework has a few concepts novel to the single-label case which should be reviewed before looking at the algorithms for ML learning. In this chapter, the core notation for the thesis will be introduced and a clear definition of the task of ML learning will be given. Then, a deep look is taken into the unique properties of ML data and how these might affect the performance of classifiers. The concepts of label correlation and class imbalance will also be introduced, however, how to deal with these will be discussed in the next chapter (for now). Finally, we will get to the evaluation metrics of ML algorithms. This is an important topic in ML learning, often neglected in the literature [cite]. After completing this chapter, the reader will have a good basis to be able to move on to the discussion of ML learning algorithms.

3.2 The Task of Multi-Label Learning

A more formal definition of the ML learning task will be given in the following chapter. However, it is important to note that we will define the utlimate task of ML learning as the assigning of multiple labels to an observation. ML learning covers to very similar approaches, namely, ML classification and ML ranking. ML classification algorithms output whether or not labels are relevant to an observation (binary) and ML ranking algorithms outputs a real-valued score assigned to each label indicating its relative importance to an observation. Thus with ML ranking, for each observation we seek a list of labels ordered by their scores representing the confidence in how relevant they are to the specific observation. 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. (see (Zhang and Zhou, 2014) for a more brief description)

3.2.1 Notation

The following notation will be used throughout the thesis. Define the input matrix as

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_1^\intercal & \boldsymbol{x}_2^\intercal & \dots & \boldsymbol{x}_n^\intercal \end{bmatrix},$$

where n is the number of observations and p is the number of features. $\boldsymbol{x}_i^\intercal$ represents the p-dimensional vector that forms the i-th row of X. For a text classification problem, x_{ij} might indicate the number of times a word j appeared in document i. Define the label or output matrix as

$$Y = \begin{bmatrix} y_{11} & y_{12} & \dots & y_{1K} \\ y_{21} & y_{22} & \dots & y_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \dots & y_{nK} \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_1^\intercal & \boldsymbol{y}_2^\intercal & \dots & \boldsymbol{y}_K^\intercal \end{bmatrix} = \begin{bmatrix} \boldsymbol{Y}_{(1)} & \boldsymbol{Y}_{(2)} & \dots & \boldsymbol{Y}_{(K)} \end{bmatrix},$$

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, $k = 1, \dots, K$, 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 ML data set will be defined as $D = \begin{bmatrix} X & Y \end{bmatrix}$, which contains the n input-output pairs, $\{(\boldsymbol{x}_i, \boldsymbol{y}_i) \,|i = 1, \dots, n\}$. Note that, $\boldsymbol{y}_i = (y_1, y_2, \dots, y_K)$, $y_k \in \{0, 1\}$, used here is the label vector, however, it is also common to use the label set 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.

3.2.2 Classification and Ranking

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. The accuracy aforementioned is a topic that will be discussed shortly. The measurement thereof is another unique problem for ML classification.

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 $f(\boldsymbol{x}_i)$.

3.2.2.1 Threshold Calibration

++ explained at the end of this chapter + maybe not in detail here + 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?)

Let $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ define a multi-label dataset. $\boldsymbol{x}$, is the feature/input/instance vector of an observation and is given by a p-dimensional real-valued vector, $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)$, *i.e.* $\boldsymbol{x} \in \mathcal{R}^p$. Each instance, $\boldsymbol{x}$ is associated with a subset of labels $L \in 2^{\mathcal{L}}$, where $2^{\mathcal{L}}$ represents the powerset of the full set of labels, $\mathcal{L} = \{l_1, l_2, \ldots, l_K\}$. The subset L is represented as an indicator vector $\boldsymbol{y} = (y_1, y_2, \ldots, y_K)$, where $y_k = 1$ if $l_k \in L$ or else $y_k = 0$, for $k = 1, 2, \ldots, K$. We assume examples in $\mathcal{D}$ to be independently and identically distributed (*i.i.d.*) from $P(\boldsymbol{X}, \boldsymbol{Y})$. Let h define a multi-label classifier, which is a mapping,

$$h : \boldsymbol{X} \to \boldsymbol{Y}$$

(not sure about this notation). The risk of h is defined as the expected loss over the joint distribution $P(\boldsymbol{X}, \boldsymbol{Y})$:

$$R_L(h) = E_{\boldsymbol{XY}} \left[L\left(\boldsymbol{Y}, h(\boldsymbol{X})\right)\right],$$

where $L(.)$ is a multi-label loss function. The MLC task boils down to given training data, $\mathcal{D}$, drawn independently from $P(\boldsymbol{X}, \boldsymbol{Y})$, learn a classifier h that minimizes the risk with respect to a specific loss function, *i.e.*

$$h^* = \arg\min_h E_{\boldsymbol{XY}} \left[L\left(\boldsymbol{Y}, h(\boldsymbol{X})\right)\right] = \arg\min_h E_{\boldsymbol{X}} \left[E_{\boldsymbol{Y}|\boldsymbol{X}} \left[L\left(\boldsymbol{Y}, h(\boldsymbol{X})\right)\right]\right],$$

where h^* is the so-called risk-minimizing model and can be determined in a pointwise way by the risk minimizer,

$$h^*(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}} E_{\boldsymbol{Y}|\boldsymbol{X}} \left[L(\boldsymbol{Y}, \boldsymbol{y}))\right].$$

Note, here we allow $h(\boldsymbol{x})$ to take on real values, *i.e.* $h(\boldsymbol{x}) \in \mathcal{R}^K$, for the sake of generality. This is to cover multi-label ranking functions and multi-label classifiers that output real real values before thresholding.

3.3 Evaluation Metrics

(Tsoumakas and Vlahavas) 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.

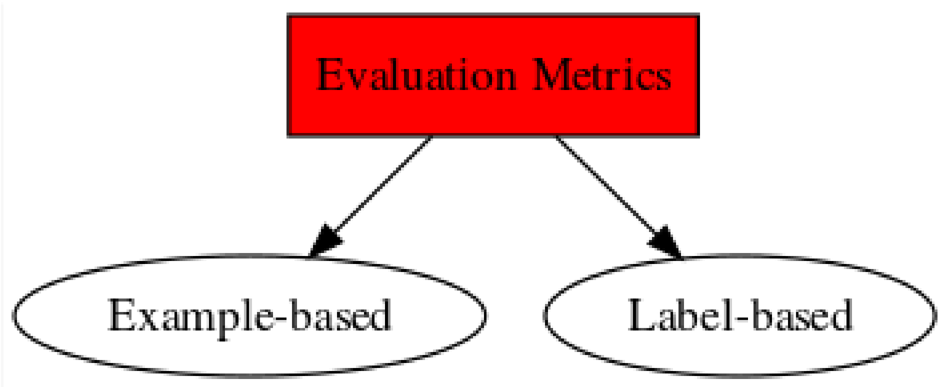


Figure 3.2: Categorisation of the taxonomy of MLL evaluation metrics

3.3.1 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:

3.3.2 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 (Tsoumakas and Vlahavas) 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 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), (Read *et al.*, 2011*a*), (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.

3.4.1 Two types of label dependence

As mentioned, most mutli-label learning papers display merely an intuitive understanding of *label dependence*, in the sense that in predicting a specific label, the information on the rest of the labels may be helpful. For example in an image recognition problem, if a picture is labelled with *beach* and *ocean*, *sand* will most likely be a relevant label. Clearly, this understanding is insufficient to gain advances in the multi-label learning literature (later on it will also be pointed out why this may indeed not make intuitive sense). In this section, a formal statistical definition of the two types of label dependence will be given. First, we briefly revisit the task of multi-label classification (MLC), in mathematical(?) terms.

3.4.1.1 Marginal vs. conditional dependence

First note that we denote the conditional distribution of $\boldsymbol{Y} = \boldsymbol{y}$ given $\boldsymbol{X} = \boldsymbol{x}$ as

$$P(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x}) = P(\boldsymbol{y}|\boldsymbol{x})$$

and the corresponding conditional marginal distribution of Y_k (conditioned on $\boldsymbol{x}$) as

$$P(Y_k = b|\boldsymbol{x}) = \sum_{y_i = b} P(\boldsymbol{y}|\boldsymbol{x}).$$

(can probably also write as $P(Y_k|\boldsymbol{x})$ since b is either 0 or 1?)

(Dembcz *et al.*, 2012) defines two types of dependence among lables, namely, conditional dependence and marginal dependence. Their definitions follow:

Definition 1 *A random vector of labels* $\boldsymbol{Y} = (Y_1, Y_2, \ldots, Y_K)$ *is called marginally independent if*

$$P(\boldsymbol{Y}) = \prod_{k=1}^{K} P(Y_k). \tag{3.4.1}$$

Marginal dependence is also known as unconditional dependence and can be thought of as a measure of the frequency of co-occurrence among labels. Conditional dependence captures the dependence of the labels given a specific observation $\boldsymbol{x}$.

Definition 2 *A random vector of labels is called conditionally independent, given* $\boldsymbol{x}$ *if*

$$P(\boldsymbol{Y}|\boldsymbol{x}) = \prod_{k=1}^{K} P(Y_k|\boldsymbol{x}). \tag{3.4.2}$$

The conditional joint distribution of a random vector $\boldsymbol{Y} = (Y_1, Y_2, \ldots, Y_K)$ can be expressed by the product rule of probability $(P(AB) = P(A|B)P(B))$:

$$P(\boldsymbol{Y}|\boldsymbol{x}) = P(Y_1|\boldsymbol{x}) \prod_{k=2}^{K} P(Y_k|Y_1, \ldots, Y_{k-1}, \boldsymbol{x}). \tag{3.4.3}$$

A similar expression can be given for $P(\boldsymbol{Y})$. If $Y_1, Y_2, \ldots, Y_K$ are conditionally independent, then Equation 3.4.3 will simplify to Equation 3.4.2.

Marginal and conditional dependence are closely related - it can be written as:

$$P(\boldsymbol{Y}) = \int_{\mathcal{X}} P(\boldsymbol{Y}|\boldsymbol{x}) d\mu(\boldsymbol{x}), \tag{3.4.4}$$

where μ is the probability measure on the input space $\mathcal{X}$ induced by the joint probability distribution P on $\mathcal{X} \times \mathcal{Y}$. Marginal dependence can roughly be viewed as an 'expected dependence' over all instances. Nevertheless, marginal dependence does not imply conditional independence, or *vice versa*. Two examples from (Dembcz *et al.*, 2012) are given to illustrate this.

Example 1 *Suppose two labels,* Y_1 *and* Y_2*, are independently generated from* $P(Y_k|\boldsymbol{x}) = (1 + \exp(-\phi f(\boldsymbol{x}))^{-1}$*, where* ϕ *controls the Bayes error rate. Thus, by definition, the two labels are conditionally independent with conditional joint distribution,* $P(\boldsymbol{Y}|\boldsymbol{x}) = P(Y_1|\boldsymbol{x}) \times P(Y_2|\boldsymbol{x})$*. However, as* $\phi \to \infty$*, the Bayes error tends to zero and the marginal dependence increases to an almost deterministic case of* $y_1 = y_2$*. Showing, conditional independence does not* *imply marginal independence.*

Example 2 *Suppose two labels, Y_1 and Y_2, are to be predicted by using a single binary feature, x_1. Let the joint distribution $P(X_1, Y_1, Y_2)$ be given by the following table:*

x_1	y_1	y_2	P
0	0	0	0.25
0	0	1	0.00
0	1	0	0.00
0	1	1	0.25
1	0	0	0.00
1	0	1	0.25
1	1	0	0.25
1	1	1	0.00

Thus, the labels are not conditionally independent,

$$P(Y_1 = 0, Y_2 = 0|x_1 = 1) = 0 \neq P(Y_1 = 0|x_1 = 1) \times P(Y_2 = 0|x_1 = 1) = 0.25 \times 0.25,$$

but it can be shown that they are indeed marginally independent. For example,

$$P(Y_1 = 0, Y_2 = 0) = 0.25 = P(Y_1 = 0) \times P(Y_2 = 0) = 0.5 \times 0.5.$$

This holds for all the combination of labels, showing that marginal independence does not imply conditional independence.

This distinction between marginal and conditional dependence is crucial in the attempt to model label dependence in multi-label classification. We describe a multi-output model with the following notation, similar to (Hastie *et al.*, 2009):

$$Y_k = h_k(\boldsymbol{X}) + \epsilon_k(\boldsymbol{X}), \tag{3.4.5}$$

for all $k = 1, 2, \ldots, K$. $h_k : \boldsymbol{X} \to \{0, 1\}$ will be referred to as the structural part and $\epsilon_k(\boldsymbol{x})$ as the stochastic part of the model. Note that a common assumption in multi-variate regression (real-outputs) is that

$$E[\epsilon_k(\boldsymbol{x})] = 0. \tag{3.4.6}$$

for all $\boldsymbol{x} \in \boldsymbol{X}$ and $k = 1, 2, \ldots, K$. This is not a reasonable assumption in mutli-label classification (Dembcz *et al.*, 2012) - the distribution of the noise terms can depend on $\boldsymbol{x}$ and two or more noise terms can depend on each other. Classifier h_k might also be very similar to h_l, $l \neq k; l = 1, 2, \ldots, K$. Thus there are two possible sources of label dependence: the structural part and the stochastic part of the model.

It seems that marginal dependence between labels is caused by the similarity between the structural parts. This assumption is made since it is reasonable to assume that the structural part will dominate the stochastic part. Suppose there exists a function $f(.)$ such that $h_k \approx f \circ h_l$, *i.e.*

$$h_k(\boldsymbol{x}) = f(h_l(\boldsymbol{x})) + g(\boldsymbol{x}), \tag{3.4.7}$$

with $g(.)$ being negligible in the sense that $g(\boldsymbol{x}) = 0$ with high probability. Then this $f(.)$-*dependence* between the classifiers is likely to dominate the averaging process in Equation 3.4.4, compared to $g(.)$ and the stochastic parts. This is what happens in Example 1 when $\phi \to \infty$. Thus we see that even if the dependence between h_k and h_l is only probable, it can still induce a dependence between the labels Y_k and Y_l (verstaan nie presies wat hier bedoel word nie). Another example illustrating idea is given from (Dembcz *et al.*, 2012).

Example 3 *Consider a problem with a 2-dimensional input* $\boldsymbol{x} = (x_1, x_2)$, *where* x_i *is uniformly distributed in* $[-1, 1]$ *for* $i = 1, 2$, *and two labels,* Y_1, Y_2, *determined as follows.* Y_1 *is set to 1 for all positve values of* x_1, *i.e.* $Y_1 = I(x_1 > 0)$. *The second label is generated similarly but with the decision boundary of* Y_1 $(x_1 = 0)$ *rotated by an angle of* $\alpha \in [0, \pi]$ *(give illustration). In addition, let the two error terms of the model be independent and both flip the label with a probability of* 0.1. *If* α *is close to zero, the labels will almost be identical and a high correlation will be observed between them. But if* $\alpha = \pi$, *the decision boundaries of the labels are orthogonal and a low correlation will be observed.*

With regards to Equation 3.4.7, in Example 3, $f(.)$ is the identity function and $g(.)$ given by the ± 1 in the regions between the decision boundaries. From this point of view, marginal dependence can be seen as a kind of soft constraint that a learning algorithm can exploit for the purpose of regularization (Dembcz *et al.*, 2012). (verstaan nie wat dit beteken nie)

For the conditional dependence, it seems that the stochastic part of the model is the cause. In Example 3, Y_1 and Y_2 is conditionally independent because the error terms are assumed to be independent. However, if there is a close relationship between ϵ_1 and ϵ_2, this conditional independence will be lost. (Dembcz *et al.*, 2012) proves the proposition that a vector of labels is conditionally dependent given $\boldsymbol{x}$ if and only if the error terms in Equation 3.4.5 are conditionally dependent given $\boldsymbol{x}$, *i.e.*

$$E\left[\epsilon_1(\boldsymbol{x}) \times \cdots \times \epsilon_K(\boldsymbol{x})\right] \neq E\left[\epsilon_1(\boldsymbol{x})\right] \times \cdots \times E\left[\epsilon_K(\boldsymbol{x})\right].$$

(Include proof?) It should also be noted that conditional independence can also cause marginal dependence because of Equation 3.4.4. Thus the similarity between models is not the only source of of marginal dependence.

What we have learned thus far is that there is a difference between marginal and conditional label dependence. The presence of marginal dependence does not imply conditional label dependence and *vice versa*. If label correlations are observed it can only be assumed that marginal dependence between the labels exist. It does not necessarily imply that there are any dependencies among the error terms (although it could be the cause). On the other hand, if conditional dependence is observed, one can safely assume that there are dependencies among the error terms. Next, we see how to exploit both types of label dependence to improve predictive accuracy.

3.4.2 Link between label dependence and loss minimization

One can view the MLC task from different persepectives in terms of loss minimizations. (Dembcz *et al.*, 2012) describes three such views, determined by the type of loss function to be minimized, the type of dependence taken into account and the distinction between marginal and joint distribution estimation. The three views and the main questions to consider for each of them are:

1. The individual label view: How can we improve the predictive accuracy of a single label by using information about other labels?
2. The joint label view: What type of non-decomposable MLC loss functions is suitable for evaluating a multi-label prediction as a whole and how to minimize such loss functions?
3. The joint distribution view: Under what conditions is it reasonable to estimate the joint conditional probability distribution over all label combinations?

3.4.2.1 The individual label view

With this view, the goal is to minimize a loss function that is label-wise decomposable and we want to determine whether or not it will help taking label relationships into account. The most common and intuitive label-wise decomposable loss function is the Hamming loss, which is defined as the fraction of labels whose relevance is incorrectly predicted:

$$L_H\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = \frac{1}{K}\sum_{k=1}^{K} I\left(y_k \neq \hat{y}_k\right). \tag{3.4.8}$$

Equation 3.4.8 is only the Hamming loss for one observation. To compute the Hamming loss over an entire dataset, Equation 3.4.8 is averaged over all the observations.

It is easy to see that the Hamming loss is minimized when

$$\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_K),$$

where

$$\hat{y}_k = \arg\max_{y_k \in \{0,1\}} p(y_k|\boldsymbol{x}),$$

for $k = 1, 2, \ldots, K$. This shows that it is enough to take only the conditional marginal distribution $P(Y_k|\boldsymbol{x})$ into account to solve the problem, at least on a population level. Thus the Hamming loss is minimized by BR. (Dembcz *et al.*, 2012) also gives a similar result for label-wise decomposable loss functions in general (thus also relevant for F-measure, AUC, *etc.*). This result implies that the multiple single label predictions problem can be solved on the basis of $P(Y_k|\boldsymbol{x})$ alone. Hence, with a proper choice of base classifiers and parameters for estimating the conditional marginal probabilities, there is in principle no need for modelling conditional dependence between the labels. However, in cases where the base classifiers are inadequate, dependence between the errors will exist and BR will give a suboptimal solution (make sure this statement is used correctly). Methods exist to improve BR in these situations and will be discussed shortly.

3.4.2.2 The joint label view

Here we are interested in non-decomposable (label-wise) MLC loss functions such as rank loss and the subset 0/1 loss. We discuss when they are appropriate and how to minimize them. First, consider the rank loss. Suppose the true labels constitute a ranking in which all relevant labels ideally precede all irrelevant ones and $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), \ldots, h_K(\boldsymbol{x}))$ is seen as a ranking function representing a degree of label relevance sorted in a decreasing order. The rank loss simply counts the number of label pairs that disagree in these two rankings:

$$L_r\left(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})\right) = \sum_{(k,l);y_k>y_l} \Big(I\left(h_k(\boldsymbol{x}) < h_l(\boldsymbol{x})\right) + \frac{1}{2} I\left(h_k(\boldsymbol{x}) = h_j(\boldsymbol{x})\right) \Big). \quad (3.4.9)$$

This function is not convex nor differentiable, thus an alternative would be to minimize a convex surrogate like the hinge or exponentional function. However, (Dembcz *et al.*, 2012) proves that it is enough to minimize Equation 3.4.9 by sorting the labels by their probability of relevance:

Theorem 1 *A ranking function that sorts the labels according to their probability of relevance, i.e. using the scoring function $\boldsymbol{h}(.)$ with $h_k(\boldsymbol{x}) = P(Y_k = 1|\boldsymbol{x})$, minimizes the expected rank loss.*

(include proof?) This implies again (just like in the case for the label-wise decomposable loss functions) that, in principle, it is not necessary to know the joint label distribution $P(\boldsymbol{Y}|\boldsymbol{x})$ when training a multi-label classifier, *i.e.* risk-minimizing predictions can be made without any knowledge about the conditional dependency between labels. Thus, to minimize the rank loss, one can simply use any approach minimizing the single label losses. Note this results does not hold for the normalized version of rank loss.

Next, we look at the extremely stringent multi-label loss function, the subset 0/1 loss:

$$L_S\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = I\left(\boldsymbol{y} \neq \hat{\boldsymbol{y}}\right). \tag{3.4.10}$$

Although most would agree that this is not a fair measure for MLC performance, since it does not disinguish between almost correct and completely wrong, it is still interesting to study with regards to exploiting label dependence. The risk-minimizing prediction for Equation 3.4.10 is given by the mode of the distribution:

$$h_s^*(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} P(\boldsymbol{Y}|\boldsymbol{x}). \tag{3.4.11}$$

This implies that the entire distribution of $\boldsymbol{Y}$ given $\boldsymbol{X}$ is needed to minimize the subset 0/1 loss. Thus a risk minimizing prediction requires the modelling of the joint distribution and hence the modelling of the conditional dependence between labels. Later on we will show an important results that under independent outputs, minimizing the Hamming loss and the subset 0/1 loss is equivalent, implying that BR will indeed also minimize the subset 0/1 loss (consider to show it here).

The cases for F-measure loss and the Jaccard distance is a bit more complicated and will not be discussed here. (give citation of where this can be found)

3.4.2.3 The joint distribution view

We just saw that minimzing the subset 0/1 loss requires the estimation of the entire conditional joint distribution, $P(\boldsymbol{Y}|\boldsymbol{X})$. Generally, if the joint distribution is known, a risk-minimizing prediction can be derived for any loss function in an explicit way:

$$h^*(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}} E_{\boldsymbol{Y}|\boldsymbol{x}}\left[L(\boldsymbol{Y}, \boldsymbol{y})\right].$$

In some applications modelling the joint distribution may result in using simpler classifiers, potentially leading to a lower cost and a better performance compared to directly estimating marginal probabilities by means of more complex classifiers. Nevertheless, it remains a difficult task. One has to estimate 2^{K} values to estimate for a given $\boldsymbol{x}$.

Theoretical insights into MLC

- proposition (with proof in paper): The hamming loss and subset 0/1 loss have the same risk-minimizer, *i.e.* $\boldsymbol{h}_H^*(\boldsymbol{x}) = \boldsymbol{h}_s^*(\boldsymbol{x})$, if one of the following conditions holds: (1) Labels $Y_1, \dots, Y_K$ are conditionally independent, *i.e.* $P(\boldsymbol{Y}|\boldsymbol{x}) = \prod_{k=1}^{K} P(Y_k|\boldsymbol{x})$. (2) The probability of the mode of the joint probability is greater than or equal to 0.5, *i.e.* $P(\boldsymbol{h}_S^*(\boldsymbol{x})|\boldsymbol{x}) \geq 0.5$.
- corollary (with proof in paper): In the separable case (*i.e.* the joint conditional distribution is deterministic, $P(\boldsymbol{Y}|\boldsymbol{x}) = I(\boldsymbol{Y} = \boldsymbol{y})$), the risk minimizers of the hamming loss and subset 0/1 loss coincide.

MLC algorithms for exploiting label dependence

- in general not able to yield risk-mininizing predictions for multi-label losses but is well suited for loss functions whose risk-minimizer can solely be expressed in terms of marginal (conditional) distributions.
- may be sufficient, but exploiting marginal dependencies may still be beneficial especially for small-sized problems.
- several methods that exploit similarities between structural parts of the label models.
- general scheme:

$$\boldsymbol{y} = \boldsymbol{b}(\boldsymbol{h}(\boldsymbol{x}), \boldsymbol{x}), \tag{3.4.12}$$

where $\boldsymbol{h}(\boldsymbol{x})$ is the binary relevance learner and $\boldsymbol{b}(.)$ is an additional classifier that shrinks or regularizes the solution of BR. Or

$$\boldsymbol{b}^{-1}(\boldsymbol{y}, \boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x}), \tag{3.4.13}$$

where the output space is first transformed and then the BR classifiers are trained and then transformed back to original. + Stacking follows first scheme. Form of regularization or feature expansion. Not clear which inputs should all be use for second level. + compressive sensing

Experimental evidence

- marginal independence: stacking does improve on BR, CC similar to SBR, LP also bad. Error increases with number of labels. hamming and subset 0/1 coincide.
- conditional independence: again loss functions coincide. SBR improves over BR, even higher when structural parts are more similar.Supports theoretical claim that the higher the structural similarties the more prominent effect of stacking. Study rest of results.

Nou opsomming van (Read and Hollmén, 2015) - sodra klaar, probeer in hoofstuk inkorporeer.

Introduction

- n-th feature vector $\boldsymbol{x}^{(n)} = [x_1^{(n)}, \ldots, x_p^{(n)}]$, where $x_j \in \mathcal{R}$, $j = 1, \ldots, p$.
- in the traditional binary classification task we are intersted in having a model h to provide a prediction for test instances $\tilde{\boldsymbol{x}}$, *i.e.* $\hat{y} = h(\tilde{\boldsymbol{x}})$. In MLC there are K binary output class variables (labels) and thus $\hat{\boldsymbol{y}} = [\hat{y}_1, \ldots, \hat{y}_K] = h(\boldsymbol{x})$.
- probabilistic speaking h seeks the expecctation $E[\boldsymbol{y}|\boldsymbol{x}]$ of unknown $p(\boldsymbol{y}|\boldsymbol{x})$. This task is typically posed as a MAP estimate of the joint posterior mode
$$\hat{\boldsymbol{y}} = [\hat{y}_1, \ldots, \hat{y}_K] = h(\tilde{\boldsymbol{x}}) = \arg \max_{\boldsymbol{y} \in \{0,1\}^p} p(\boldsymbol{y}|\tilde{\boldsymbol{x}})$$
This corresponds to minimizing the susbet 0/1 loss.
- $h_{BR}(\tilde{\boldsymbol{x}}) := [h_1(\tilde{\boldsymbol{x}}), \ldots, h_K(\tilde{\boldsymbol{x}})]$
- entirety of ML literature point out that BR obtain suboptimal performance because it assumes labels are independent.
- several approaches attempt to correct/regularize BR, SBR.
- others attempt to learn the labels together, LP. $\hat{\boldsymbol{y}} = h_{LP}(\tilde{\boldsymbol{x}})$
- another example is CC done using a greedy search:
$$h_{CC}(\tilde{\boldsymbol{x}}) := [h_1(\tilde{\boldsymbol{x}}), h_2(\tilde{\boldsymbol{x}}, h_1(\tilde{\boldsymbol{x}})), \ldots, h_K(\tilde{\boldsymbol{x}}, \ldots, h_{K-1}(\tilde{\boldsymbol{x}}))]$$
- PCC formulates CC as the joint distribution using the chain rule,
$$h_{CC}(\boldsymbol{x}) := \arg \max_{\boldsymbol{y}} p(y_1|\boldsymbol{x}) \prod_{k=2}^{K} p(y_k|\boldsymbol{x}, y_1, \ldots, y_{K-1})$$
and show that it is indeed possible to make a Bayes-optimal search with guarantees to the optimal solution for 0/1 loss. Several search techniques exist to make the seach optimal, but greedy is still popular.
- order and structure of chains in cc is the main focus point.
- although in theory the chain rule holds regardless of the order of variables, each $p(y_k|\boldsymbol{x}, y_1, \ldots, y_{K-1})$ is only an approximation of the true probability because it is modelled from finite data under a constrainded class of model, and consequently a different indexing of labels can lead to different results in practice.
- many approaches try to find the best order and show better empirical results, but the reason why is not quite clear

- LP can be viewed as modelling the joint probability directly,

$$h_{LP}(\boldsymbol{x}) := \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}, \boldsymbol{x})$$

- two main points from previous papers: (1) the best label order is impossible to obtain from observational data only. (2) the high performance of classifier chains is due to leveraging earlier labels in the chain as additional feature attributes.

The role of label dependence in multi-label classification

- marginal dependence: frequency of co-occurence among labels
- conditional dependence: after conditioning on the input
- modelling complete dependence is intractable
- rather attempt pairwise marginal dependence or use of ensemble.
- many new methods do not outperform each other over a reasonable amount of datasets.
- improvements of prediction on standard multi-label datasets reached a plateau (maybe investigate).
- question the logic, if the ground truth label dependence could be known and modelled, multi-label predictive performance would be optimal and therefore as more technique and computational effort is invested into modelling label dependence, the lead of the new methods over BR and other predecessors will widen.
- BR might be underrated
- modelling label dependence is a compensation of lack of training data and one could only assume that given infinite data two separate binary models on labels y_k and y_l could achieve as good performance as one that models them together.
- the 'intuitive' understanding actually seems quite flawed: if we take two labels and wish to tag images with them, the assumption that label dependence is key to optimal multi-label accuracy is analogous to assuming that an expert trained for visually recognising one label will make optimum classifications only if having viewed the classiication of an expert trained on the other label.
- in reality, modelling label dependence only helps when a base classifier behind one or more labels is inadequate.
- depends on the base classifier
- there is no guarantee that an ideal structure based on label dependence can be found at all given any amount of training data.
- see XOR problem
- take the view that BR can perform as well as any other method when there is no dependence among the outputs given the inputs.

- not to say that BR should perform as well as other methods if there is no dependence *detected*. Due to noisy data or insufficient model dependence may be missed or even introduced.
- if a ML method outperforms BR under the same base classifier then we can say that it using label dependence to compensate for the inadequacy in its base classifiers.
- attempt to remove the dependence among the labels
- dependence generated by inadequate base classifiers

Binary relevance as a state-of-the-art classifier

- CC and LP are representative of PT problems. Succesful on many fronts and can be builded on. Still has some drawbacks. Discusses them.
- BR less parameters to tune.
- multi-label classifiers can be comprised of individual binary models that perform equally as well as models explicitly linked together based on label dependence or even a single model that learns labels together (intrinsic label dependence modelling).
- claim this is the case for example and label based metrics. (not what the previous paper found)
- proposition with proof: given $X = x$, there exists a classifier $h'_2(x) \approx \arg\max_{y_2 \in \{0,1\}} p(Y_2|X)$ that achieves at least as small error as classifier $h_2(x) \approx \arg\max_{y_2 \in \{0,1\}} p(Y_2|Y_1, X)$, under loss $L(y_2, \hat{y}_2) = I(y_2 \neq \hat{y}_2) = I(y_2 \neq h_2(x))$. Instances of X, Y_1, Y_2 are given in the training data but only $\tilde{x}$ is given at test time. (see proof in paper)
- This means that if we are interested in a model for any particular label, best accuracy can be obtained in ignorance of other labels.
- proposition and proof: under observations $X = x$, there exists two individually constructed classifiers $h'_1 \approx \arg\max_{y_1} p(Y_1|X)$ and $h'_2 \approx \arg\max_{y_2} p(Y_2|X)$ such that under 0/1 loss, $[h_1(x), h_2(x)] \equiv \hat{\boldsymbol{y}} \equiv \boldsymbol{h}(x)$ are equivalent, where $\boldsymbol{h} \approx \arg\max_{[y_1,y_2]} p(Y_1, Y_2|X)$ models labels together. Instances of X, Y_1, Y_2 are given in the training data but only x (tilde) is given at test time. (see proof in paper)
- following examples, X represents some document and Y_1, Y_2 represent the relevance of two subject categories for it. Latent variable Z represents the unobservable current events which may affect both the observation X and the decisions for labelling it. (illustration of all of the scenarios)
- ignore case where input and all labels are independent.
- case of conditional independence - a text document is given independently to two human labelers who each independently identify if the document is relevant to their expert domain.

$$
\begin{aligned}
p(\boldsymbol{y}, x) &= p(y_1, y_2) \\
&= p(y_1|x)p(y_2|y_1, x) \\
&= p(y_1|x)p(y_2|x)
\end{aligned}
$$

which obviously can be solved with BR, where $h_k(\tilde{x}) := \arg\max_{y_k} p(y_k|\tilde{x})$.

- a text document is labelled by the first labeller and afterwards by the second expert - potentially biasing the decision to label relevance or not with this second label. If we do not impose any restriction on any $h_k(x)$, it is straightforward to make some latent $z \equiv h_1(x)$ such that $h_2(x, z) \equiv h_2(x, h_1(x))$. We speak of equivalence in the sense that given Z we can recover Y_2 to the same degree of accuracy (probably compared to case without Z). In this analogy the second labeller must learn also the first labeller's knowledge and thus makes the first labeller redundant. If we drop Y_1 we return to the original structure.
- two experts label a document X but both are biased by each other and - possibly to alternate degrees - by an external source of information Z. Can also introduce latent variables Z_1, Z_2 to break the dependence between the labels.
- note the dependence between any variable can be broken by introducing hidden variables not just the label variables. Hence we can further break dependence between X and Y_1 in the same way - if we desire.
- universal approximation: with a finite number of neurons, even with even with a linear output layer, a network can approximate any continuous function. Implies for ML - given a large enough but finite feature representation in the form of a middle layer, any of the labels can be learned independently of the others, *i.e.* a linear BR layer can suffice for optimal classification performance.
- to summarise: if we find dependence between labels it can be seen as a result of marginalizing out hidden variables that generated them. Also, we can add hidden variables to remove the dependence between labels.
- this does not mean we have a method to learn this structure. Which is learning latent variables powerful enough.
- EM and MCMC sampling under energy models to learn latent variables by minimizing the energy and thus maxmimizing the joint probability with observed variables. (iterative procedures).
- unsupervised part more difficult than supervised
- **existing methods to obtain conditional independence among labels.**
- task: making outputs independent of each other by using a different input space to the original such that a simpler classifier can be employed to predict outputs.
- deep learning to learn a powerful higher-level feature representations of the data. (uses multiple hidden layers)
- in MLC the labels can be seen as high-level feature representations.
- **the equivalence of loss metrics under independent outputs**
- if outputs are independent of each other given the input, then minimizing Hamming loss and 0/1 loss is equivalent.

- the risk of Hamming loss is minimized by BR

$$\hat{y}_k = \arg \max_{y_k \in \{0,1\}} p(y_k|\boldsymbol{x})$$

for each label. The 0/1 loss on the other hand, is minimized by taking the mode of the distribution,

$$\hat{\boldsymbol{y}} = \arg \max_{\boldsymbol{y} \in \{0,1\}^K} p(\boldsymbol{y}|\boldsymbol{x})$$

equivalently written as

$$\hat{\boldsymbol{y}} = \arg \max_{\boldsymbol{y} \in \{0,1\}^K} p(y_1|\boldsymbol{x}) \prod_{k=2}^{K} p(y_k|\boldsymbol{x}, y_1, \ldots, y_{K-1}).$$

- Noting that when all outputs are independent of each other given the input ($p(y_k|\boldsymbol{x}, y_l) \equiv p(y_k|\boldsymbol{x})$), then for all k, l it becomes

$$\begin{aligned}\hat{\boldsymbol{y}} &= \arg \max_{\boldsymbol{y} \in \{0,1\}^K} \prod_{k=1}^{K} p(y_k|\boldsymbol{x}) \\ &= \left[\arg \max_{y_1 \in \{0,1\}} p(y_1|\boldsymbol{x}), \ldots, \arg \max_{y_K \in \{0,1\}} p(y_k|\boldsymbol{x})\right].\end{aligned}$$

- here input refers to the input into the model and not the original features.
- we can replace the input with hidden variables derived from the original feature space in order to make them independent. If this is successful, the above holds, and using BR will achieve the same result as CC on either measure.
- suppose only the third of three outputs is successfully made independent, then prediction of independent models is optimizing

$$\hat{\boldsymbol{y}} = \left[\arg \max_{y_1, y_2 \in \{0,1\}^2} p(y_1, y_2|\boldsymbol{x}), \arg \max_{y_3 \in \{0,1\}} p(y_3|\boldsymbol{x})\right].$$

- if this is the case it could be handled elegantly by RAkELd - disjoint labelset segmentations RAkEL. But detecting these mixed dependence sets is difficult.
- RAkEL and ECC benefit from the ensemble effect of reducing variance of estimates but it is not clear what loss measure is being optimized.

Classifier chains augmented with synthetic labels (CCASL)

- difficult to search for good order in CC
- if 'difficult' label is at start of chain, all other labels may suffer.
- present a method that adds synthetic labels to the beginning of the chain and builds up a non-linear representation, which can be leveraged by other classifiers further down the chain. CCASL

- create H synthetic labels.
- many options - they used threshold linear unit (TLU) to make binary, can also try others like ReLU with continuous output. or sigmoid and radial basis.
- the synthetic labels can be interpreted as random cascaed basis functions, except that at prediction time the values are predicted and thus we refer to them as synthetic labels.
- synthetic label $z_k = I(a_k > t_k)$ with activation values

$$a_k = \left([B * W]^T_{k,1:(p+(k-1))} \cdot \boldsymbol{x}'_k\right)$$

  where W is a random weight matrix (sampled from multivariate normal) with identically sized masking matrix B where $B_{i,j} \sim Bernoulli(0.9)$, input $\boldsymbol{x}'_k = [x_1, \ldots, x_p, z_1, \ldots, z_{k-1}]$ (not the same k as label index), and threshold $t_k \sim \mathcal{N}(\mu_k, \sigma_k \cdot 0.1)$
- want to use synthetic labels at beginning of chain to improve prediction of the real labels.
- $\boldsymbol{y}' = [z_1, \ldots, z_H, y_1, \ldots, y_K]$ and from the predictions $\hat{\boldsymbol{y}}'$ we extract the real labels $\hat{\boldsymbol{y}} = [\hat{y}'_{H+1}, \ldots, \hat{y}_{H+K}] = [\hat{y}_1, \ldots, \hat{y}_K]$.
- $\hat{y}_j = \arg\max_{y_j \in \{0,1\}} p(y_j | x_1, \ldots, x_p, z_1, \ldots, z_H, y_1, \ldots, y_{j-1})$
- use LR as base classifier
- label order less of an issue.
- does well on complex non linear synthetic data - overfits on simple linear synthetic data.
- lots of tunable parameters
- few hidden labels are necessary for CCASL, empirical suggests $H = K$.
- **CCASL + BR**
- guards against overfitting, removes connections among the output
- advantages of BR, stacking and CC
- no back prop necessary.
- **CCASL+AML**
- CCASL strucutre is powerful for modeling non-linearities. CCASL+BR regularizes but otherwise does not offer a more powerful classifier.
- whereas we created synthetic labels from feature space, we can do the same from the label space.
- layer of binary nodes which are feature functions created from the label space for each subset
- see rest in paper.
- section on other network based literature
- back prop bad
- simply using a powerful non-linear base classifier may remove the need for transformations of the feature space altogether.

Experiments

- done in python and sklearn
- synthetic dataset and music, scene, yeast, medical, enron, reuters (max K = 103)
- 10 iterations for each datset 60/40 split
- report parameters
- all out-perform BR and CC
- BR_{RF} does best under hamming loss! RF are adequately powerful to model each layer
- CCASL are quite expensive
- the main advantage brought by modelling label dependence via connections among outputs is that of creating a stronger learner.
- did not investigate ensembles

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.

- [Tsoumakasf] use the ϕ coefficient to estimate label correlations.

- (Sorower)

- 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)

3.4.3 Theoretical Results

- 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

3.5 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.

- 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

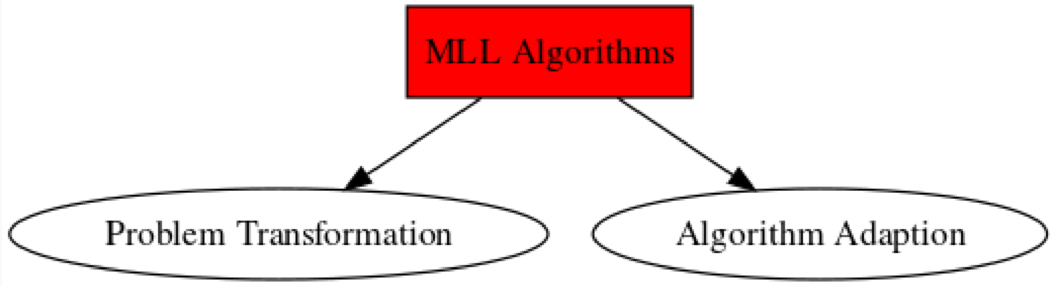


Figure 3.3: Categorisation of multi-label learning taxonomy (this is just an example)

- 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 (Tsoumakas and Vlahavas).

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.

3.5.1 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 (Read *et al.*, 2011*a*): 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 (Read *et al.*, 2011*a*): 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 (Tsoumakas *et al.*), (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).

3.5.2 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 (Tsoumakas and Vlahavas). 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.

3.5.3 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 (Read *et al.*, 2011*a*). 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).

3.6 Algorithm Adaption Approaches

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).

3.6.1 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

3.6.2 Multi-Label Decision Tree (ML-DT)

- basic idea
- procedure
- psuedo-code
- remarks: first-oders; efficient; improve with pruning and or ensembling; computational complexity

3.6.3 Ranking Support Vector Machine (Rank-SVM)

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

3.6.4 Collective Multi-Label Classifier (CML)

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

3.7 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.

3.7.1 Ensemble of Classifier Chains

In a response to this (referring to CC), the ensembles of classifier chains (ECC) was suggested by (Read *et al.*, 2011*a*). 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 (Read *et al.*, 2011*a*) (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.

3.7.2 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. (Tsoumakas and Vlahavas) 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 (Tsoumakas and Vlahavas) 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.

3.7.3 Summary

- 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 (Sorower)
- complement of this paper would be a broad empirical study

3.8 Threshold Calibration

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

3.9 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

Chapter 4

Multi-Label Deep Neural Networks

4.1 Introduction

We have seen that Deep Convoltional Neural Networks have achieved great success on single-label image classificaiton problems. This is because of their strong capability in learning discriminative features when trained on large datasets, which are also transferable to other image classification problems. Naturally, one might wonder whether these learnt features are useful in the MLC domain.

This may shed some light on the multi-label image classification problem.

Main questions:

- is the single-label features transferable to multi-label images?
- can we learn from label correlations?

A common approach that extends CNNs to MLC is to transform it into multiple single-label classification problems, which can be trained with the ranking loss pr the cross-entropy loss. However, when treating labels independently, these methods fail to model the dependency between multiple labels. (Xue *et al.*, 2011) has shown that multi-label classification problems ehibit strong label co-occurrence dependencies.

The most common and simple method is to take popular existing networks, *e.g.* VGG and ResNet, and replace the activation function of their final classification layer to a sigmoid activation and train the network by minimising the sum of the binary cross-entropies per label, as in (Grzeszick *et al.*, 2016) and (Szalkai and Grolmusz, 2017). As we have seen in Chapter 2, the sigmoid activation ensures that the output lies between 0 and 1, but unlike the softmax activation, the sum of these outputs do not add to 1. Therefore the sigmoid layer is more suitable for cases with potentially multiple labels.

The simple approach described above is mostly related to the BR approach, since direct relationships between labels are not taken into account. However, the weights of the network are learnt by taking all of the labels jointly into account. A pure BR approach would be to train a separate network for each label. If there are enough images per label, this approach should in theory be sufficient to classify the images. For instance, it should be possible to detect a *car* in an image indepently of the presence of a *road*. However, in practice, image classifiers are imperfect and could be improved by having information on the other labels.

4.2 Differences Between Single Label and Multiple Labels Images

However, these feature representations might not be optimal for images with multiple labels. For example, one of the labels might only relate to a very small region of the image. This label would most likely be underepresented by the features learned on single label datasets, since for single label images, the label is typically related to a large part, and usually in the centre, of the image.

In single label image datasets, it is quite fair to assume that the foreground objects are roughly aligned. This assumption more risky for images with multiple labels. A typical multi-label image has objects of different categories scattered around the image at different scales and orientations. Maybe show example images?

Interaction between objects.

Label space has been expanded and more training data is needed. But also more costly to annotate multi-label images.

4.3 Short History

- focus mostly on image classification, but can also mention other DNN which is possible to transfer to image classification.

Backpropogation for Multi-Label Learning (BP-MLL) (Zhang *et al.*, 2006) is claimed by the authors to be the first multi-label neural network. It was applied in text classification and function genomics, but could be applied to image classification aswell. As its name implies, BP-MLL is derived from the backpropogation algorithm through replacing its error function with a new function defined to capture the characteristics of multi-label learning, that is, the labels belonging to an instance should be ranked higher than those not belonging to that instance. It views each output node as binary classification task, and relies on the architecture and loss function to exploit the dependency across labels. It was later expanded by (Nam *et al.*, 2013) with state-of-the-art learning techniques such as dropout. Also se (Hu *et al.*, 2015) for structured inference NN which uses concept layers modeled with label graphs.

Extensions of conventional DNNs to suit multi-label image problems are either architectural adaptions or the optimisation of different loss functions more suited for MLC. We will first look at the latter.

4.4 Non-Deep Learning Approaches

Traditional bag-of-words model can be decomposed into multiple modules:

- feature representation: generate global representations for images
- feature extraction: Extracted from dense grids and sparse interest points - SIFT, Histogram of Oriented Gradients and Local Binary Patterns
- feature coding: Qunatise extracted features - Vector Qunatisation, Sparse Coding and Gaussian Mixture Models.
- feature pooling: Feature aggregation methods - Spatial Pyramid Matching
- classification: SVM or Random Forest
- context modelling: the use of context information such as spatial location of object and background scene from global view can considerably improve the performance

These learnt features are not always optimal.
Where does VLAD and Fisher vectors fit in?

4.5 Optimising Multi-Label Loss functions

The first proposed approach to extend neural networks to MLC (in general), named Backpropogation for Multi-Label Learning (BP-MLL), was a modification of the loss function to be optimised. Consider the global error of the network on the training set as

$$E = \sum_{i=1}^{n} E_i,$$

where E_i is the error of the network on $\boldsymbol{x}_i$, which could be defined as:

$$E_i = \sum_{j=1}^{Q} (c_j^i - d_j^i)^2,$$

where $c_j^i = c_j(\boldsymbol{x}_i)$ is the actual output of the network on $\boldsymbol{x}_i$ on the j-th class, d_j^i is the desired output of $\boldsymbol{x}_i$ on the j-th class which takes the value of either +1 ($j \in Y_i$) or -1 ($j \notin Y_i$). If the weights of the network is learned through backpropogating the errors from this loss function, some important characteristics of multi-label learning are not considered. Here E_i only concentrates on indivdual label discrimination and it does not consider the correlations between the different labels. Therefore the authors of (Zhang *et al.*, 2006) suggested rewriting the global error function as follows:

$$E = \sum_{i=1}^{n} E_i = \sum_{i=1}^{n} \frac{1}{|Y_i||\bar{Y}_i|} \sum_{(k,l)\in Y_i \times \bar{Y}_i} \exp(-(c_k^i - c_l^i)).$$

Here $\bar{Y}_i$ is the complementary set of Y_i in $\mathcal{Y}$ and $|\cdot|$ measures the cardinality/size of a set. $c_k^i - c_l^i$ measures the difference between the outputs of the network on one label belonging to $\boldsymbol{x}_i$ ($k \in Y_i$) and one label not belonging to it ($l \in \bar{Y}_i$). Therefore, the bigger the difference, the better performance. The negation of this difference is fed to the exponential function in order to severly penalise the i-th error term if the output on the label belonging to $\boldsymbol{x}_i$, c_k^i, is much smaller than the output on the label not belonging to $\boldsymbol{x}_i$, c_l^i. The summation is over all pairs of labels where the one belongs to $\boldsymbol{x}_i$ and the other does not. This is then normalised by the denominator, $|Y_i||\bar{Y}_i|$, which is the total number of such possible pairs.

The minimisation of this global error will lead the network to ouput larger values for labels belonging to the training instance and smaller values for those not belonging to it. It is shown in their paper that this error function is closely related to the rankining loss criterion. The specifics of the minimisation of this error function with gradient descent and backpropogation is beyond the scope of this thesis and can be found in the original paper (**?**). In the paper, BP-MLL show superiority to the well-established multi-label learning algorithms of that time. Although the algorithms were tested on the applications of text classification and functional genomics, it could also be applied to image classification, since the loss function is not affected by the application domain.

Is it acceptable to claim that this loss function forces the NN to learn label dependencies? (Oquab *et al.*, 2015) claims the following. Treating a multi-label classification problem as K independent classification problems is often inadequate because it does not model label correlations. This is not an issue here because the classifiers share hidden layers and therefore are not independent. Such a network can model label correlations by tuning the overlap of the hidden state distribution given each label.

- See (Nam *et al.*, 2013) for clearer explanation of rank loss minimisation. They also show how thresholding needs to be done with this network.

Plenty of years later, the authors of (Nam *et al.*, 2013) showed that BP-MLL's ranking loss could efficiently and effectively be replaced with the commonly used cross entropy error function and demonstrate that several advances in neural network training that have been developed in the realm of deep learning can be effectively employed in this setting. With these new developments in architectures, it is better to optimise the cross-entropy rather than the BP-MLL loss. Apparently the BP-MLL loss function is not consistent with the rank loss and cross entropy is. Cross-entropy is also computationally more efficient. Their experiments are more trustworthy because of comparsions over more datasets with more measures.

One of the first approaches to extending CNNs to a multi-label image classification problems was by minimising loss functions more suited for MLC, more specifically, the multi-label ranking loss (Gong *et al.*, 2013). Found that weighted approximate ranking loss worked best for CNNs. Showed more than 10% increase to conventional BoW methods on NUS-WIDE dataset. Still need to read the rest of the article but apparently an effective model requires lots of training samples (they did not use transfer leanring).

They show that a significant performance gain could be obtained by combining convolutional architectures with approximate top-k ranking objectives.

Focus on loss functions tailored for multi-label prediction tasks. The first loss function they considered was the softmax loss adopted to the MLC context. Consider the convolutional network, $f(\cdot)$, where the convolutional layers and dense connected layers filter the images. The output of $f(\cdot)$ is a scoring function of the data point $\boldsymbol{x}$, that produces a vector of activations. The posterior probability of an image, $\boldsymbol{x}_i$, and class j (out of K possible classes) can be expressed as

$$p_{ij} = \frac{\exp(f_j(\boldsymbol{x}_i))}{\sum_{k=1}^{K} \exp(f_k(\boldsymbol{x}_i))},$$

where $f_j(\boldsymbol{x}_i)$ is the activation value for image $\boldsymbol{x}_i$ and class j. The KL-divergence between the predictions and the ground-truth probabilities can then be minimised. The ground-truth probabilities can be obtained by normalising $\boldsymbol{y}$ as $\boldsymbol{y}/||\boldsymbol{y}||_1$, where $\boldsymbol{y}$ is a binary vector of size K indicating class presence/absence (1/0). (not sure about this ground truth transformation - seems like it is unncessary). Let the ground-truth probability for image i and class j be defined as $\bar{p}_{ij}$. The cost function then to be minimised is

$$J = -\frac{1}{m}\sum_{i=1}^{n}\sum_{j=1}^{K} \bar{p}_{ij}\log(p_{ij}) = -\frac{1}{m}\sum_{i=1}^{n}\sum_{j=1}^{c+}\frac{1}{c+}\log(p_{ij}),$$

where $c+$ dentes the number of positive labels for each image. Technically, then $c+$ is dependent on i, but the authors chose this notation for ease of exposition.

The second loss was a simple modification of a pairwise-ranking loss, which takes multiple labels into account. The aim was to rank positive labels to always have higher scores than negative labels. This led to the problem of minimising

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

where $c-$ is the number of negative labels. (can improve on this indexing). During the backpropogation they computed the sub-gradient of this loss function. One limitation of this loss is that it optimise the area under the ROC curve (AUC) but does not directly optimise the top-k annotation accuracy.

The third loss function they experimented with is a multi-label variant of the weighted approximate ranking (WARP) loss, which uses a sampling trick to optimise top-k annotation accuracy. It specifically optimises the top-k accuracy for annotation by using a stochastic sampling approach. It minimises

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

where $L(\cdot)$ is a weighting function for different ranks and r_j is the rank for the j-th class for image i. The weighting function used is defined as:

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

with $\alpha_1 \geq \alpha_2 \geq \cdots \geq 0$. They defined α_j as $1/j$. It is clear that $L(\cdot)$ will assign a small weight to the loss if a positive label is ranked top in the label list. Howeverm if a positive label is not ranked top, $L(\cdot)$ will assign a much larger weight to the lossm which pushes the positive label to the top.

To find the rank r_j, they followed the following sampling method. For a positive label, continue sampling negative labels until a violation is found and let s be the number of trials sampled for negative labels. The rank was estimated by the following formulation

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

Still need to figure this out.

They only compared on NUS-WIDE with no standard errors and only on per class precision and recall and overall precision and recall. Might be valuable to compare on more datasets and with more metrics. Also, these were tested with older CNN architectures, so it may be unfair to compare novel approaches utilising state-of-the-art CNN architectures to these approaches. They found that WARP loss gave the best results.

(Zhu *et al.*, 2017*a*) used multi-label CNN for pedestrian attribute detection. The adaption they made to the CNN was also by redefining the loss function. The defined the loss function as

$$F = \sum_{k=1}^{K} \lambda_k G_k,$$

where G_k is the loss of the k-th attribute(/label). K is the total number of labels and $\lambda_k \geq 0$ is a parameter controlling the contribution of each label. In their experiments they set $\lambda_k = 1/K$ and defined G_k as

$$G_k = -\frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M^k} 1\{y_n^k = m\} \cdot \log \frac{\exp((w_m^k)^T x_n^k)}{\sum_{m=1}^{M} \exp((w_m^k)^T x_n^k)}.$$

N is the number of training samples and M^k represents the class number of k-th attribute. To avoid bias due to imbalanced data, they further extend G_k as follows:

$$G_k = -\frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M^k} 1\{y_n^k = m\} \cdot \beta_m^k \log \frac{\exp((w_m^k)^T x_n^k)}{\sum_{m=1}^{M} \exp((w_m^k)^T x_n^k)},$$

where $\beta_m^k = \frac{1/N_m^k}{\sum_{l=1}^{k} 1/N_l^k}$. N_m^k is the number of samples holding m-th class label of k-th attribute and it meets $\sum_{m=1}^{M} N_m^k = N^k$. Make sure first that this loss is much different to the others proposed and what the difference is between attribute and label. This might not be applicable to a pure MLC problem.

4.5.1 Sparsemax ML loss

(Martins and Astudillo, 2016) propose the sparsemax transformation. Sparsemax has the distinctive feature that it can return sparse posterior distributions, *i.e.* it may assign exactly zero scores to some of its ouput variables. This is a convenient feature for MLC, especially when the labelset is large. Added benefits of this transformation is that it preserves the attractive properties of the softmax: it is simple to evaluate, it is even cheaper to differentiate and it can be turned into a convex loss function.

Let $\Delta^{K-1} := \{\boldsymbol{p} \in \mathbb{R}^K | \mathbf{1}^T\boldsymbol{p} = 1, \boldsymbol{p} \geq \mathbf{0}\}$ be the $(K-1)$-dimensional simplex. We are interested in functions that map vectors in $\mathbb{R}^K$ to probability distributions in Δ^{K-1}, such that we can obtain label posterior probabilities from label scores. We have already seen the softmax function defined as

$$\text{softmax}_i(\boldsymbol{z}) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$

A limitation of the softmax transformation is that $\text{softmax}_i(\boldsymbol{z}) \neq 0$ for every $\boldsymbol{z}$ and i. This is disadvantageous in applications where a sparse probability distribution is desired. An alternative is to use the sparsemax transformation:

$$\text{sparsemax}(\boldsymbol{z}) := \arg\min_{\boldsymbol{p}\in\Delta^{K-1}} ||\boldsymbol{p} - \boldsymbol{z}||^2.$$

Therefore, the sparsemax returns the Euclidean projection of the input vector $\boldsymbol{z}$ onto the probability simplex. This projection is likely to hit the boundary of the simplex, in which the sparsemax($\boldsymbol{z}$) becomes sparse. In the paper they show that the sparsemax retains most of the important properties of the softmax.

Now we need to show how to use the sparsemax transformation to design a new loss function that resembles the logistic loss but can yield sparse distributions. The loss function associated with the softmax is the logisitc loss (or negative log-likelihood):

$$\begin{aligned} L_{\text{softmax}}(\boldsymbol{z}; k) &= -\log(\text{softmax}_k(\boldsymbol{z})) \\ &= -z_k + \log\left(\sum_{j=1} \exp(z_j)\right) \end{aligned}$$

Can find the gradient ... Need something similar for sparsemax.

The authors derived a convex and differentiable loss names the softmax loss:

$$L_{\text{softmax}}(\boldsymbol{z}; k) = -z_k + \frac{1}{2} \sum_{j \in S(\boldsymbol{z})} (z_j^2 - \tau^2(\boldsymbol{z})) + \frac{1}{2},$$

where τ^2 is the square of the threshold function (see paper). They show that in the binary case, the sparsemax reduces to the Huber classification loss. The generalisation to the MLC case can be given by:

$$L_{\text{sparsemax}}(\boldsymbol{z}; \boldsymbol{q}) = -\boldsymbol{q}^T \boldsymbol{z} + \frac{1}{2} \sum_{j \in S(\boldsymbol{z})} (z_j^2 - \tau^2(\boldsymbol{z})) + \frac{1}{2} ||\boldsymbol{q}||^2.$$

Details omitted here.

The results do not seem that convincing.

(Wang *et al.*, 2017) experimented with the Hinge, Euclidean and Cross-Entropy losses. They found that the ouput is very sparse and that the network struggled with finding positive images. To alleviate this problem they suggested using a positive/negative balancing factor, β_P, β_N, with the cross-entropy loss which results in the weighted cross-entropy loss:

$$L(f(\boldsymbol{x}), \boldsymbol{y}) = -\beta_P \sum_{y_c=1} \ln(f(\boldsymbol{x}_c)) - \beta_N \sum_{y_c=0} \ln(1 - f(\boldsymbol{x}_c)),$$

where $\beta_P = \frac{|P|+|N|}{|P|}$ and $\beta_N = \frac{|P|+|N|}{|N|}$. $|P|$ and $|N|$ are the number of positive and negative labels per batch of images, respectively. The weighted version worked better for them, especially for classes with fewer examples.

(Szalkai and Grolmusz, 2017) also used a weighting of the loss contribution per class to help the infrequent labels.

4.5.2 F-measure maximisation

- https://arxiv.org/pdf/1604.07759.pdf actually this is not a DNN loss so probably won't include it here.
- https://arxiv.org/pdf/1608.04802.pdf shows how to optimise for the F_β-measure. But they don't mention MLC so not sure if it is applicable.

4.5.3 Other

- see https://arxiv.org/pdf/1701.05616.pdf
- https://arxiv.org/pdf/1705.02315.pdf
- see https://arxiv.org/pdf/1706.07960.pdf for pseudo huber loss

4.6 Proposal Based Approaches

(Razavian *et al.*, 2014) and (Sermanet *et al.*, 2013) was the first to propose a CNN feature extraction approach. It consisted of feeding all the images of a multi-label image dataset to a CNN trained on ImageNet to get CNN activations as the off-the-shelf features for classification, where they used a SVM. Since ImageNet is a single label image classification problem, these features were not optimal for a multi-label problem (because of alignment and occlusion issues). An improvement to this idea is to annotate the images with bounding boxes indicating the presence of objects, such as in (Oquab *et al.*, 2014) and (Girshick *et al.*, 2013), but these bounding box annotations are very costly.

The HCP method described below requires no bounding box information for tranining and is robust to the possible noisy and/or redundant hypotheses. Much fewer hypotheses are alos required, giving a significant speed up in training.

If multiple labels are associated with a single image, it is fair to assume that the different labels are related to different visual regions of the image. Proposal based CNN methods attempt to cope with this problem. (Start with single to multi CNN paper). Proposal based methods are also very popular or object detection problems.

- Hypotheses-CNN-Pooling (HCP)
- takes an arbitrary number of object segment hypotheses as the inputs (use state-of-the-art objectiveness detection techniques, like BING)
- and then a shared CNN is connected with each hypothesis.
- The CNN output from each hypothesis is aggregated by max pooling
- Pro: no ground truth is required which makes labelling cheaper
- Pro: robust to noisy and/or redundant hypothesis (thanks to max pooling)
- Pro: can take an arbritrary number of hypothesis as input
- Pro: CNN can be pretrained on single image datasets.

However, these methods ignore semantic relations between labels. Next we will look at ways to capture these semantic relations in image classification.

4.7 RNN-CNN

- (Wang *et al.*, 2016)

- note end to end frameworks are proven to be very effective.

Traditional approaches to multi-label image classification learn independent classifiers for each category and employ ranking or thresholding on the classification results. These techniques work well, but fail to explicitly exploit the label dependencies in an image.

So far it seems that this is the first paper on CNN for MLC that attempts to exploit label correlations. Previous attempts to model label dependency are mostly based on graphical models. This approach is prohibitive with large labelsets. This paper explicitly model label dependencies with recurrent neural networks (RNNs) to capture higher-order label relationships while keeping the computational complexity tractable.

To avoid problems like overfitting, previous methods normally restrict CNN classifiers to share the same image features for each class. When using the same image features to predict multiple labels, objects that are small in the images are easily ignored or hard to recognise independently. The RNNs framework is designed to adapt the image features based on the previous prediction results, by encoding the attention models implicitly in the CNN-RNN structure. The idea behind it is to implicitly adapt the attentional area in images so the CNNs can focus its attention on different regions of the images when predicting different labels. Small objects are hard to recognise by itself, but can be easily inferred given enough contexts.

the following part may need to go under label embedding sections

In addition, many image labels have overlapping meanings. Exploiting the semantic redundancies reduce the computational cost and also imporves the generalisation ability because the labels with duplicate semantics can get more training data. The label semantic redundancy can be exploited by joint image/label embedding, which can be learned via canonical correlation analysis, metric learning or learning to rank methods. The joint image/lable embedding maps each label or image to an embedding vector in a joint low-dimensional Euclidean space such that the embeddings of semantically similar labels are close to each other, and the embedding of each image should be close to that of its associated labels in the same space. This is effective for exploiting label semantic redundancy because it essentially share classification parameters for semantically similar labels. But the label co-occurrence decpendency is largely ignored.

RNN-CNN is a unified framework for multi-label image classification which effectively learns both the semantic redundancy and the co-occurence dependency in an end-to-end way. The framework is as follows. The multi-label RNN model learns a joint low-dimensional image-label-embedding to model the semantic relevance between iamges and labels. The image embedding vectors are generated by a deep CNN while each label has its own label embedding vector. The high-order label co-occurence dpendency in this low-dimensional space is modeled with the long short term memory recurrent neurons, which maintains the information of label context in their internal memory states. The RNN framework computes the probability of a multi-label prediction sequentially as an ordered prediction path, where the a priori probability of a label each time step can be computed based on the image embedding and the output of the recurrent neurons. During prediction, the multi-label prediction with the highest probability can be approximately found with the beam search algorithm. This whole framework can be trained in an end-to-end fashion.

Other methods modelling label dependencies also only mostly model pairwise combinations.

Can visualise attentional regions.

Test on MS COCO, NUS-WIDE and PASCAL VOC 2007. No standard errors and report per class precision, recall, F1 and overall. Also MAP@ 10. Shows good performance but strangely does not use the best HCP on VOC2007 dataset to compare with. CNN-RNN and HCP2000 is actually quite equal. Should be tested on more datasets and maybe a combination should be considered. For example region proposals from HCP and label dependencies with RNN.

Have not read the full CNN-RNN paper. An extra detail is that the RNN require sequential input. Therefore the unordered labelset should be ordered. The original paper uses the frequent first order. (Jin and Nakayama, 2016) uses the rarest first order, which apparently helps with the classification of the less frequently occurring classes. Read the rest of the paper. This order problem is mostly probaly solved by (Chen *et al.*, 2017).

There are some extensions to this RNN-CNN idea. The first we will look at is given by (Liu *et al.*, 2016). Note, the CNN-RNN pattern is also commonly used in image captioning. The original CNN-RNN just discussed, utilises the weakly semantic CNN hidden layer or its transform as the image embedding that provides the interface between the CNN and RNN. This overstretches the RNN to complete two tasks: predicting the visual concepts of the image and modelling their correlations. This makes end-to-end training of the network slow and ineffective due to the difficulty of backpropogating gradients through the RNN to train the CNN. (Liu *et al.*, 2016) proposes a simple modification to improve the training of the RNN-CNN network. They propose a semantically regularised embedding layer as the interface between the CNN and RNN.

The original RNN-CNN uses the final feature layer of the CNN as an interface to the RNN. This has a number of adverse effects on learning an end-to-end recurrent image annotation model. First, since the CNN output feature is not explicitly sematincally meaningful, both the label prediction and label correlation modelling tasks are expected of the RNN model. This is difficult for the RNN especially when the number of labels is vast and their correlation rich. Also, altogether the CNN-RNN is a very deep network with only supervision at the final RNN layer. This makes convergence in training very slow.

(Liu *et al.*, 2016) proposes to replace the image embedding layer and introduces semantic regularisation to the CNN-RNN model in order to produce significantly more accurate results and faster training times. Basically they follow a multi-task learning framework where the CNN-RNN network is also supervised after the CNN model by an auxiliary loss function.

Another benefit is that these models output the labels directly instead of the probability scores. This avoids the challenging thresholding calibration.

Now we should look at (Chen *et al.*, 2017) which is an improvement of this that does not depend on the ordering of the LSTM.

A fundamental and challenging issue for MLC is to identify and recover the co-occurrence of multiple labels, so that satisfactory prediction accuracy can be expected. Despite its effectiveness (in single-label image classification), how to extend CNNs for solving multi-label classification problems is still a research direction to explore.

Because of the use of a LSTM, a predefined label order is required during training. The labels are usually ordered by label frequency which is not necessarily a good proxy for label correlations. There is another concern with the above CNN-RNN which is that labels of objects which are in smaller scales in images would often be more difficult to be recovered. Can use the attention map as a solution. Still does not solve the label order problem. The other problem is that there is an inconsistency between the training and testing procedures of the CNN-RNN. In training, the labels are selected from the image ground-truth list, whereas in testing, the labels are selected from the full labelset. In other words, if a label is incorrectly predicted during a time step during prediction, such an error would propogate during the recurrent process.

(Chen *et al.*, 2017) present a novel deep learning framework of visually attended RNN, which consists of visual attention and confidence-ranked LSTM. This network is able to identify regions of interest associated with each label (even the smaller objects get attended). The order of the labels can automatically learned without any prior knowledge or assumption (with the confidence scores of the attenion maps?). This also alleviates the inconsistent procedures between training and testing.

Actually, does not look like a huge empirical improvement over the original CNN-RNN (compared on the common 3 ML image datasets). Again they compare with methods that utilise older CNN architectures.

However, these methods do not capture spatial relations between labels. This is a challenging problem because most of the times these spatial locations are not known beforehand, *i.e* the images are not annotated with these spatial locations. Spatial Regularisation Networks (Zhu *et al.*) attempt to capture both semantic and spatial relations between labels without any prior knowledge on the spatial locations of each label. This is discussed in the next section.

4.8 Spatial Regularization Networks

- This paper provides a unified deep neural network for exploiting both semantic and spatial relationships between labels with only image-level supervisions
- SRN generates attention maps for all labels and captures the underlying relations between them via learnable convolutions.
- Also aggregates regularised classification with original classifcation from RN101.
- tests on 3 benchmark datasets that show sota performance and great generalisation capability.
- can be trained end-to-end

In short, the SRN learns separate attention maps for each label, which associates related image regions to each label. By performing learnable convolutions on the attention maps of all labels, the SRN captures the underlying semantic and spatial relations between labels and act as a spatial regulariser for multi-label classification.

The attention mechanism adaptively focuses on related regions of the image when the deep networks are trained with spatially related labels (segmentation?). Intuitively this seems likely to work for ML image classification problems.

The main net has the same network structure as ResNet-101 [cite]. The SRN takes visual features from the main net as inputs and learns to regularise spatial relations between labels. Such relations are exploited based on the learned attention maps of each label. Label confidences from both the main net and SRN are aggregated to generate final classification scores.

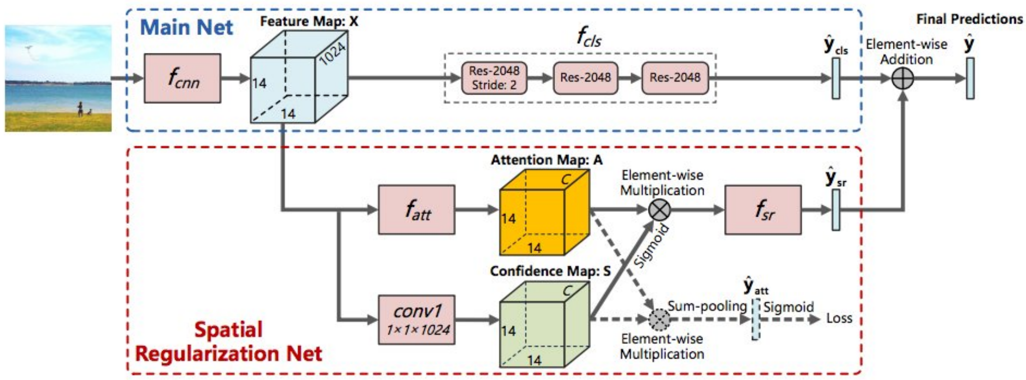


Figure 4.1: End-to-end architecture of the proposed SRN.

Not sure how in detail I should go here.

Used 224×224 sized images. The 14×14 feature map from layer named *res4b22_relu* of ResNet-101 is used as inputs to the SRN. The rest of the main net still continues to produce K class scores, which is later combined with the output of the SRN. The SRN is composed of two sub-networks. The first sub-network learns label attention maps with image-level supervisions and the second sub-network captures spatial regularisations of labels based on the learned attention maps.

Multiple image regions are semantically related to different labels. The regions locations are generally not provided, but it is desirable that more attention is paid to the related regions. SRN attempts to predict such related regions using the attention mechanism. The attention maps are then used to learn spatial regularisations for the labels. The attention map for label l related to an image should indicate the image regions related to l by displaying higher attention values to that region. The attention estimator is modeled as 3 convolutional layers with 512 kernels of 1×1, 512 kernels of 3×3 and K kernels of 1×1, where $K = |\mathcal{L}|$. The ReLU activation function is applied after the first two convolutional layers, and the softmax after the third.

Since ground-truth annotations of attention maps are not available, the network is learned with only image-level label annotations. A weighted global average is computed for each label attention maps (similar to global average pooling in ResNet). This results in a 1024 sized vector on which a linear classifier is learned to obtain class scores. They are learned by minimising the cross-entropy loss between these predicted class scores and the ground-truth labels. (seems like the attention maps work from example given in paper).

Also compute a $1 \times 1 \times 1024$ convolutional layer on the feature inputs to obtain a $14 \times 14 \times K$ confidence map. A and S are multiplied element-wise and then spatially sum-pooled to obtain the label confidence scores (after Sigmoid activation).

Then combine weighted confidence scores with attention map, by element-wise multiplication and feed as input to another series of convolutional layers. These sizes should be chosen carefully in order not to have too many parameters. Authors suggest 3 convolutional layers with ReLU, followed by one fully-connected layer. See Figure 4.2 for size of convolutions. Empirically showed the weighted attention maps work.

The final label scores are the weighted sum of the outputs of the main net and SRN, weighted by learnable paramter α. However, α can be set to 0.5 without observable performance drop.

The training is done in three parts. Training is done first by fine-tuning only the main net (but the full main net), which was pretrained on ImageNet. Then learning the attention map and confidence map simultaneously and then the convolution of the combination of the weighted attention map and confidence map. (This description will be much easier with notation).

Used data augmentation. Random crops of 256×256 image from the four corners and center, then rescaled to 224×224. See paper ref for detail.

Used SGD with batch size 96, momentum 0.9 and weight decay of 0.0005. Initial learning rate is set to 0.001 and decreases by factor 0.1 when validation

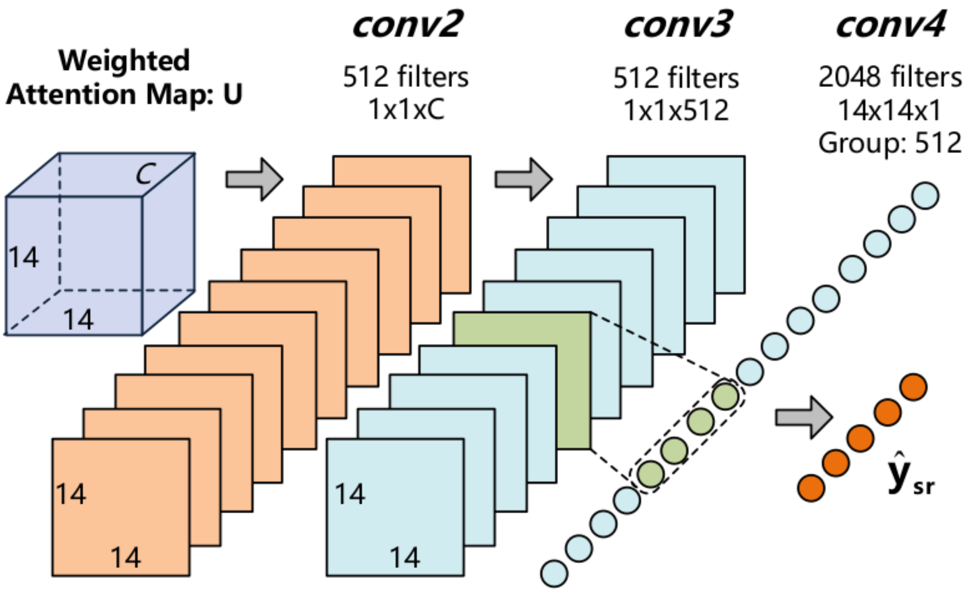


Figure 4.2: fst

loss reaches a plateau, until 0.00001. Testing is done by resizing image to 224×224.

Evaluates on 3 benchmark datasets, with multiple measures and shows promising results. Compared to pure ResNet and CNN-RNN and is the best on almost all measures.

Here is another paper on exploiting spatial relations: https://arxiv.org/pdf/1612.01082.pdf. However, I think they are using Recurrent Nets.

Attention maps interpretable.

(Wang *et al.*, 2017) designed a DCNN for multi-label classification of diseases from chest x-rays. They used a transition layer between the pretrained CNN activations and the classification layer. They also experimented with the Hinge Loss, Euclidean loss and Cross-entropy loss. See the section on loss functions above. The other things they tried were using an alternative pooling layer, named Log-Sum-Exp (LSE) pooling. Can be seen as weighted combination of max and average pooling. There is an optimal weight that achieve the best results, but I suspect it might be quite variable and difficult to determine. Also see their approach to heat map localisation.

4.9 Mixture of Experts

- see https://arxiv.org/pdf/1409.4698.pdf for MOE for MLC
- https://arxiv.org/pdf/1707.01408.pdf
- https://arxiv.org/pdf/1707.03296.pdf

4.10 Context Gating

In multi-label classification settings, an example may be annotated many labels. Some labels tend to appear in the same example at the same time, some tend not to. Such information can be used to improve the performance of multi-label classification models (Wang *et al.*, 2017).

The winner (Miech *et al.*, 2017) of the YouTube 8M video classification challenge [1], also hosted on Kaggle, proposed a network unit to better capture non-linear interdependencies between features as well as among output variables. The context gating layer transforms the input represetation X into a new representation, Y, in the following way:

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

where $X \in \mathbb{R}^n$ is a vector of input feature activations, σ is the element-wise sigmoid activation and $\circ$ is the element-wise multiplication operator. $W \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ are trainable parameters. The vector of weights ouput by $\sigma(\cdot)$ acts as a set of learnt gates with values between 0 and 1 on the individual dimensions of the input feature X.

There are two reasons why this transformation be effective. First, it introduces non-linear interactions among activations of the input representation. Second, it recalibrates the strengths of different activations of the input representation through a self-gating mechanism. [Miech2017] used conext gating to both transform the feature vector before passing it to the classification module, and after the classification layer to capture the prior structure of the output label space.

The aim of the context gating of the feature vector is to capture the dependencies among the features. For example, the context gating can learn to suppress features likely to be on 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.

The aim of the context gating unit after the classification layer is to downweight unlikely combinations such as 'Car' and 'Makeup'. It does this by reweighting the output probabilities. Note, the improvements observed was only in terms of global average precision. It will be interesting to see how it performs with other metrics and on other datsets.

4.11 Chaining

The second place solution to the YouTube-8M competition can be found in (Wang *et al.*, 2017). The main contribution made in terms of MLC their method

[1] https://www.kaggle.com/c/youtube8m

of capturing interactions between labels, named Chaining. They also propose methods for capturing multi-scale information and attention pooling. Their chaining unit is inspired by classifier chains discussed in Chapter 3. As noted in the section on classifier chains, the direction of dependence for the labels is unknown and an ensemble of CC are usually used to alleviate this problem. (Read and Hollmen, 2014) proposes a network structure that mimics CC, but this results in very deep networks, at least as deep as the number of labels.

In a chaining model, several representations are joined by a chain of classifiers (they used MoEs). The predictions are projected to features of lower dimension and used in the following stages (probably only necessaray if there are many labels). The output of each chaining unit is supervised by auxiliary cross-entropy losses to speed up convergence. The final loss is a weighted average of the auxiliary losses and the final prediction loss. They found that allocating 10-20% to the auxiliary losses gave the best results. The number of chaining units can be experimented with. Again, this method is only tested on the yt8m dataset and only in terms of the GAP.

To me it seems that this process is more related to stacked BR classifiers (BR+,2BR) instead of CCs.

Their method to exploit multiple scales is to build the chaining unit on different feature representations of multiple scales (after different convolution-pooling combinations).

- https://arxiv.org/pdf/1707.03296.pdf

4.12 Label Concept Learning

- https://arxiv.org/pdf/1707.01408.pdf

4.13 Label Processing Layer

- https://arxiv.org/pdf/1706.07960.pdf

4.14 Nearest Labelset

- see https://arxiv.org/pdf/1702.04684.pdf for nearest labelset approach

4.15 Label Embedding Approaches

- suggestion from CNN-RNN paper: (Gong *et al.*, 2012), (Weston *et al.*, 2011)

Multi-label classification can also be achieved by learning a joint image/label embedding. Multiview Canonical Correlation Analysis [] is a three-way canonical analysis that maps the image, label, and the semantics into the same latent space. WASABI [] and DEVISE [] learn the joint embedding using the learning to rank framework with WARP loss. Metric learning [] learns a discriminative metric to measure the image/label similarity. Matrix completion [] and bloom filter [] can also be employed as label encodings. These methods effectively exploit the label semantic redundancy, but they fall short on modeling the label co-occurrence dependency.

- Learning Deep Latent Spaces for Multi-Label Classification: https://arxiv.org/pdf/1707.00418.pdf
- Direct Binary embedding: https://arxiv.org/pdf/1703.04960.pdf
- cost sensitive: https://arxiv.org/pdf/1603.09048.pdf

4.16 Unsorted

- (?) for structured inference neural networks with label relations
- https://arxiv.org/pdf/1704.08756.pdf for stratification
- https://pdfs.semanticscholar.org/92f5/bd6aa3544c36490e2dac798513055233b02c.pdf for Multi-Label Transfer Learning with Sparse Representation
- http://www.cripac.ia.ac.cn/irds/People/lwang/M-MCG/Publications/2013/YH2013ICIP.pdf for Mutli-Task but not very good paper.

Chapter 5

Results (/Application)

5.1 Introduction

Main aim is to compare methods on more datasets and in terms of more evaluation metrics. The datasets to be evaluated on are described in Appendix A. These are the most popular and most recent multi-labelled image datasets.

- mention the importance of evaluating on multiple ml datasets with different properties.
- mention the importance of comparing in terms of many evaluation metrics. Which am I going to use here? Probably one of each, example-based, label-based (micro and macro).

Where possible, the basic framework will be to extract features from input images with pretrained (on ImageNet) CNN. The proposals in the literature will be built on top of these features (see transfer learning in Chapter 2). This will not give the best results because these features are trained for single label images. However, they are sufficient for comparison purposes and are much less time consuming. (Maybe I can indentify the most promosing method and see what effect fine-tuning convolutional layers will have on it. This will also allow other bells and whistles such as data augmentation.) The architecture chosen to do the feature extraction is VGG16 because of its simplicity and proven effectiveness in transfer learning.

- maybe also shrink images for faster computations.
- add time taken for learning as a metric.
- 5- or 10-fold cross-validation for more accurate errors and standard deviations.
- baseline: sigmoid classification layer optimising binary cross-entropy

5.2 Optimising Multi-Label Loss Functions

- see the effect of optimising ML loss functions on different metrics.

5.3 Label Embedding Approaches

- evaluate the effect of label embedding approaches
- does it help with class imbalance?

5.4 Novel Architectures

- evaluate the following classification heads:
- spatial regulrisation network
- cnn-rnn and improved version
- Hypothesis CNN Pooling
- Context gating
- Chaining
- Label Processing layers

5.5 Discussion

- is there a 'best' method?
- are there any patterns?
- do they perform as reported by original papers?

Chapter 6

Conclusion

- summary
- contributions
- reccomendations
- limitations
- future work

Chapter 7

Things that need a place:

- Feature learning
- one-shot learning:
  - https://github.com/sorenbouma/keras-oneshot
  - https://github.com/fchollet/keras/blob/master/examples/mnist_siamese_graph.py
  - https://sorenbouma.github.io/blog/oneshot/
- multi-task learning:
  - https://arxiv.org/abs/1706.05137
- relational learning:
  - https://arxiv.org/pdf/1706.01427.pdf
- AutoML:
  - https://research.googleblog.com/2017/05/using-machine-learning-to-explore.html

Appendices

Appendix A

Benchmark Datasets for Multi-Label Image Classification

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. Recently, a decent amount of ML data sets has been published, but not without critique. (Luaces *et al.*) argues that the MULAN[1] ML data set repository does not have data sets that are truely ML and that most of the data sets are very similar to each other. Most of the data sets have low cardinality and low label dependence. The problem with this is that these data sets may not show the true performance of ML algorithms. In (Gibaja and Ventura, 2015) the authors also comments on the lack of thorough, comparative empirical studies on these benchmark sets.

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

mention something of simulating ML data. Hard

- Simulating (Tomás *et al.*, 2014) (also gives citations to other papers)
- partitioning mentioned in (Gibaja and Ventura, 2015) - referred to (Sechidis *et al.*, 2011)
- (Luaces *et al.*) Therefore they created a ML data generator to simulate ML data on which algorithms can be evaluated.
- very important for stratification: https://arxiv.org/pdf/1704.08756.pdf and more ML metrics

A.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

[1] A Java library for ML learning - http://mulan.sourceforge.net/datasets-mlc.html.

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 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:

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

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 (Read *et al.*, 2011*b*) 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). ((Read *et al.*, 2011*b*) 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 (Read *et al.*, 2011*b*) as:

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

where 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.

Some rules that they found that I might refer to later:

- for micro-AUC target evaluation measure if label cardinality of training data is above 3.028 then the 2BR method (among the single-classifiers) should be used.
- Another example for an extracted rule is for ranking loss evaluation measure: if minimum of label entropies is zero (i.e. there is at least one certain label in the training set), number of labels is less than 53 and skewness of label cardinality is below or equal to 2.49 then the EPS method (among ensembles) should be used.
- create table with properties of data to be used in this study.
- (Read *et al.*, 2011*b*) defines a complexity measure as $n \times p \times K$
- (**?**) long list of datasets. Other than MULAN: Plant and Human, Slashdot, LangLog, IMDB
- (Sorower)
- https://manikvarma.github.io/downloads/XC/XMLRepository.html
- yelp dataset: http://www.ics.uci.edu/~vpsaini/
- also new yt8m

A.2 Amazon

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[2] applications and supposedly allows for clear distinction between water and vegetation in satellite images, for example.

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,

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

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[3]. 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.

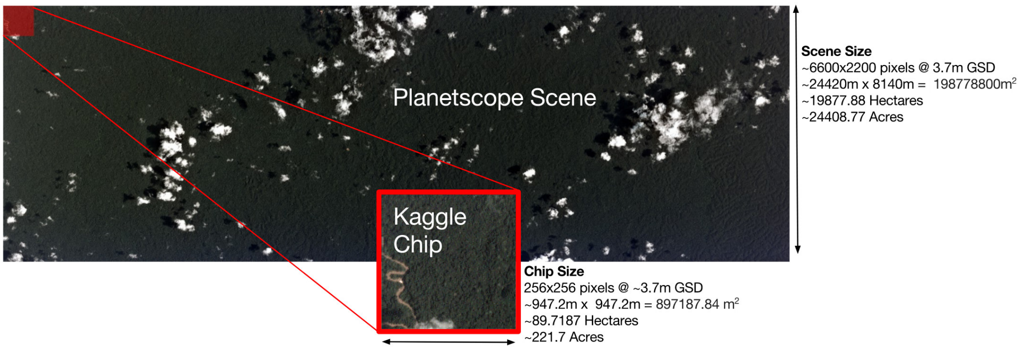


Figure A.1: Schematic of the image collection process.

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

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

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.

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

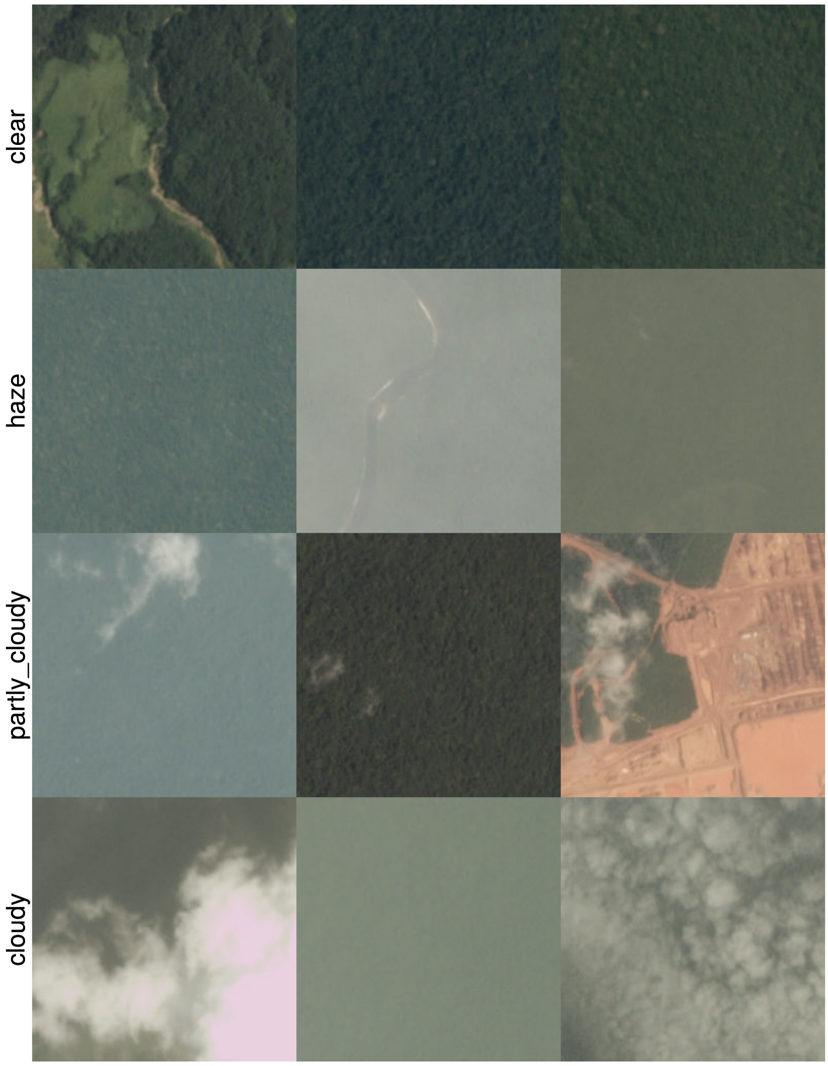


Figure A.2: Examples of chips with atmospheric labels. These (along with all the other chips plotted throughout the thesis) are the JPEG conversions of the original 4-band, 16-bit images.

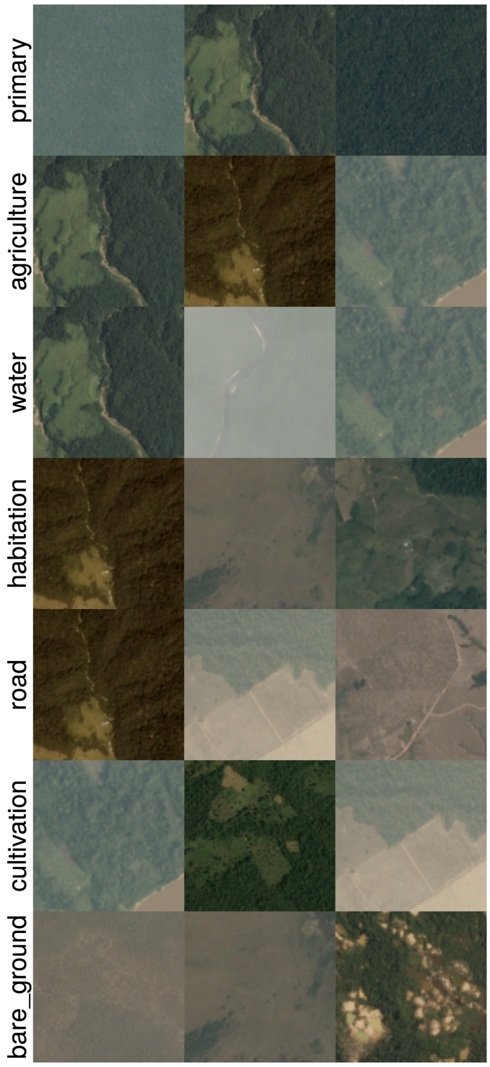


Figure A.3: Examples of chips with common land cover/use labels.

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 NUS-WIDE

- Nus-wide: a real-world web image database from national university of singapore (Chua *et al.*, 2009)

A.4 Pascal VOC

- 2007: The pascal visual object classes (voc) challenge (Everingham *et al.*, 2010)
- 2012: (Everingham *et al.*, 2012)

A.5 MS-COCO

- Microsoft coco: Common objects in contex (Lin *et al.*, 2014)

A.6 WIDER-Atribute

- Human attribute recognition by deep hierarchical contexts (Li *et al.*, 2016)

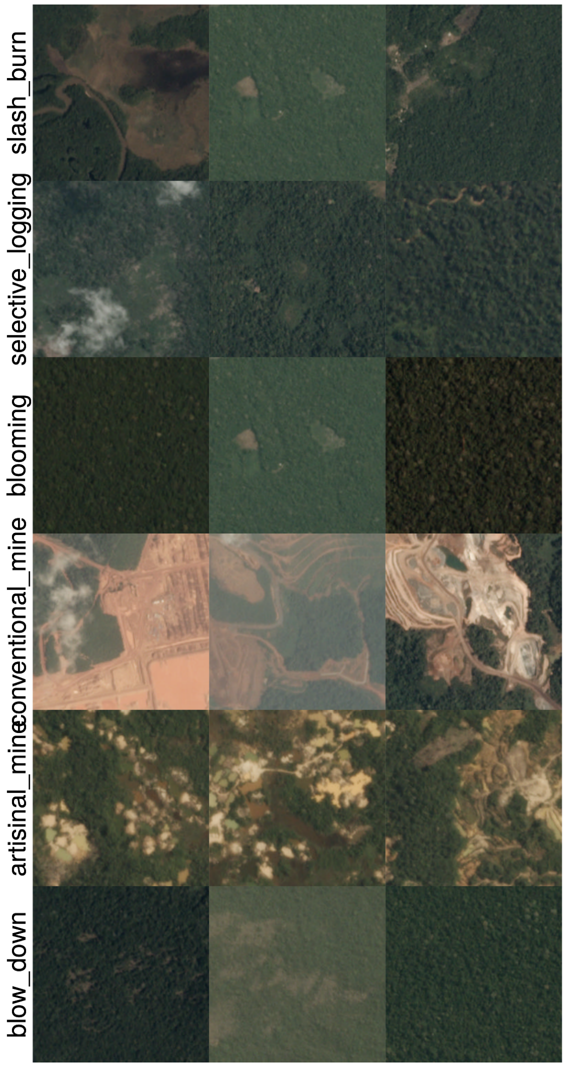


Figure A.4: Examples of chips with less common land cover/use labels.

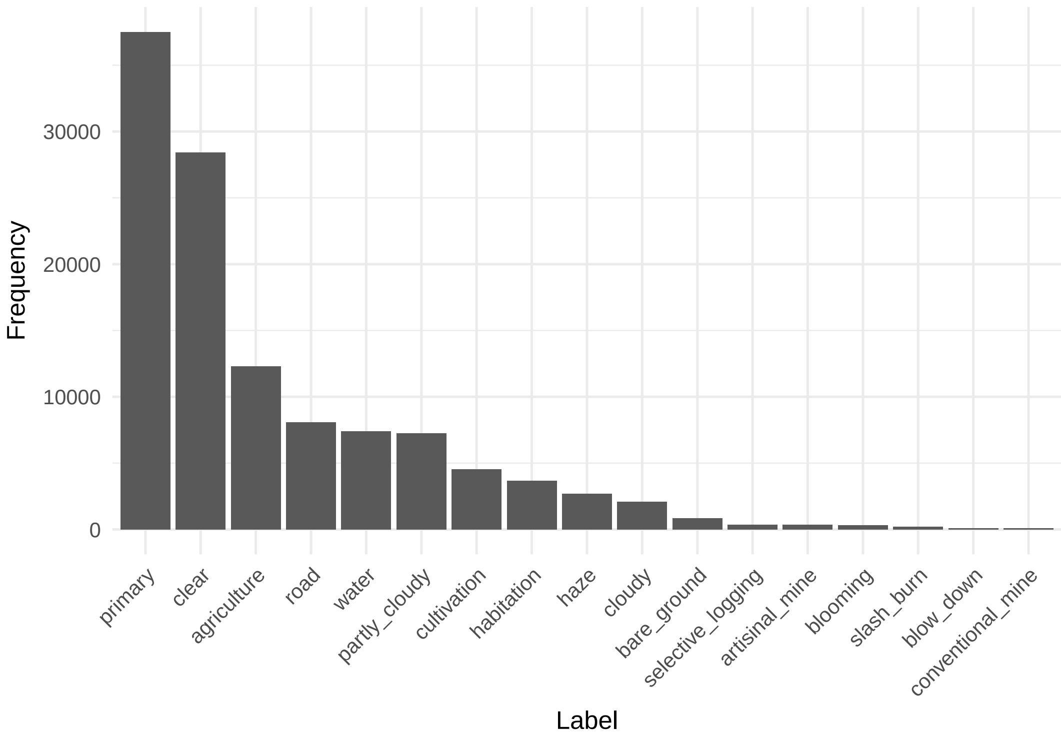


Figure A.5: Class distribution of the labels in the training set.

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

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

Boutell, M.R., Luo, J., Shen, X. and Brown, C.M. (2004). Learning multi-label scene classiÿcation. *Pattern Recognition*, vol. 37, pp. 1757–1771.
Available at: www.elsevier.com/locate/patcog

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.

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

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

Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M. and Thrun, S. (2017 Feb). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, vol. 542, no. 7639, pp. 115–118. ISSN 0028-0836. Letter.
Available at: http://dx.doi.org/10.1038/nature21056

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

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

Fürnkranz, J., Hüllermeier, E., Loza Mencía, E., Brinker, K., Fawcett Fürnkranz, T.J., Loza Mencía, E., Hüllermeier, E. and Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Mach Learn*, vol. 73, no. 73, pp. 133–153.

Available at: http://download.springer.com/static/pdf/878/art{%}253A10.1007{%}252Fs10994-008-5064-8.pdf?originUrl=http{%}3A{%}2F{%}2Flink.springer.com{%}2Farticle{%}2F10.1007{%}2Fs10994-008-5064-8{&}token2=exp=1490609445{~}acl={%}2Fstatic{%}2Fpdf{%}2F878{%}2Fart{%}25253A10.1007{%}25252Fs10994-008-506

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

Girshick, R.B., Donahue, J., Darrell, T. and Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, vol. abs/1311.2524.
Available at: http://arxiv.org/abs/1311.2524

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

Gong, Y., Ke, Q., Isard, M. and Lazebnik, S. (2012). A multi-view embedding space for modeling internet images, tags, and their semantics. *CoRR*, vol. abs/1212.4522.
Available at: http://arxiv.org/abs/1212.4522

Grzeszick, R., Sudholt, S. and Fink, G.A. (2016). Optimistic and pessimistic neural networks for scene and object recognition. *CoRR*, vol. abs/1609.07982.
Available at: http://arxiv.org/abs/1609.07982

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/

Hu, H., Zhou, G., Deng, Z., Liao, Z. and Mori, G. (2015). Learning structured inference neural networks with label relations. *CoRR*, vol. abs/1511.05616.
Available at: http://arxiv.org/abs/1511.05616

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

Jin, J. and Nakayama, H. (2016). Annotation order matters: Recurrent image annotator for arbitrary length image tagging. *CoRR*, vol. abs/1604.05225.
Available at: http://arxiv.org/abs/1604.05225

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

Lewis, D.D., Yang, Y., Rose, T.G., Li, F. and Lewis, F.L. (2004). RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, vol. 5, pp. 361–397.
Available at: http://www.jmlr.org/papers/volume5/lewis04a/lewis04a.pdf

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

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

Liu, F., Xiang, T., Hospedales, T.M., Yang, W. and Sun, C. (2016). Semantic regularisation for recurrent image annotation. *CoRR*, vol. abs/1611.05490.
Available at: http://arxiv.org/abs/1611.05490

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.

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). Author's personal copy An extensive experimental comparison of methods for multi-label learning.
Available at: http://www.elsevier.com/copyright

Martins, A.F.T. and Astudillo, R.F. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. *CoRR*, vol. abs/1602.02068.
Available at: http://arxiv.org/abs/1602.02068

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

Oquab, M., Bottou, L., Laptev, I. and Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pp. 1717–1724. IEEE Computer Society, Washington, DC, USA. ISBN 978-1-4799-5118-5.
Available at: http://dx.doi.org/10.1109/CVPR.2014.222

Oquab, M., Bottou, L., Laptev, I. and Sivic, J. (2015 June). Is object localization for free? – Weakly-supervised learning with convolutional neural networks. In: *IEEE Conference on Computer Vision and Pattern Recognition*. Boston, United States.
Available at: https://hal.inria.fr/hal-01015140

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.

Razavian, A.S., Azizpour, H., Sullivan, J. and Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, vol. abs/1403.6382.
Available at: http://arxiv.org/abs/1403.6382

Read, J. and Hollmen, J. (2014). A Deep Interpretation of Classifier Chains. *Advances in Intelligent Data Analysis Xiii*, vol. 8819, pp. 251–262. ISSN 0302-9743.
Available at: http://jmread.github.io/papers/Read,Holmen-ADeepInterpretationofClassifierChains.pdf

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

Read, J., Pfahringer, B., Holmes, G. and Frank, E. (2011*a*). Classifier chains for multi-label classification. *Machine Learning*, vol. 85, no. 3, pp. 333–359. ISSN 08856125. arXiv:1207.6324.

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

Schapire, R.E. and Singer, Y. (1999). Improved Boosting Algorithms Using Confidence-rated Predictions.
Available at: http://web.cs.iastate.edu/{~}honavar/singer99improved.pdf

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

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R. and LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, vol. abs/1312.6229.
Available at: http://arxiv.org/abs/1312.6229

Sorower, M.S. (). A Literature Survey on Algorithms for Multi-label Learning.

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.

Szalkai, B. and Grolmusz, V. (2017 March). Near perfect protein multi-label classification with deep neural networks. *ArXiv e-prints*. 1703.10663.

Tomás, J.T., Spolaôr, N., Cherman, E.A. and Monard, M.C. (2014). A framework to generate synthetic multi-label datasets. *Electronic Notes in Theoretical Computer Science*, vol. 302, pp. 155–176. ISSN 15710661.
Available at: http://ac.els-cdn.com/S1571066114000267/1-s2.0-S1571066114000267-main.pdf?{_}tid=207a475a-25c4-11e7-9d47-00000aacb35d{&}acdnat=1492691174{_}037266698571d8a927f3feb0eb432995

Trohidis, K. and Kalliris, G. (2008). Multi-Label Classification of Music Into Emotions. *Proc. ISMIR*, vol. 2008, pp. 325–330.
Available at: http://ismir2008.ismir.net/papers/ISMIR2008{_}275.pdf

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., Gr, G.A., Gr, E.A., Vilcek, J. and Gr, V.A. (2011). MULAN: A Java Library for Multi-Label Learning Eleftherios Spyromitros-Xioufis Ioannis Vlahavas. *Journal of Machine Learning Research*, vol. 12, pp. 2411–2414.
Available at: http://www.jmlr.org/papers/volume12/tsoumakas11a/tsoumakas11a.pdf

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

Tsoumakas, G., Katakis, I. and Vlahavas, I. (). A Review of Multi-Label Classification Methods.

Tsoumakas, G. and Vlahavas, I. (). Random k-Labelsets: An Ensemble Method for Multilabel Classification.

Turnbull, D., Barrington, L., Torres, D. and Lanckriet, G. (2008). Semantic Annotation and Retrieval of Music and Sound Effects. *IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, vol. 16, no. 2.
Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.297.2154{&}rep=rep1{&}type=pdf

Vens, C., Struyf, J., Schietgat, L., Džeroski, S. and Blockeel, H. (2008). Decision trees for hierarchical multi-label classification. *Machine Learning*, vol. 73, no. 2, pp. 185–214. ISSN 08856125.
Available at: https://lirias.kuleuven.be/bitstream/123456789/186698/4/hmc.pdf

Wang, H.-D., Zhang, T. and Wu, J. (2017 June). The monkeytyping solution to the youtube-8m video understanding challenge. *ArXiv e-prints*. 1706.05150.

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

Weston, J., Bengio, S. and Usunier, N. (2011). Wsabie: Scaling up to large vocabulary image annotation. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pp. 2764–2770. AAAI Press. ISBN 978-1-57735-515-1.
Available at: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-460

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

Xue, X., Zhang, W., Zhang, J., Wu, B., Fan, J. and Lu, Y. (2011). Correlative multi-label multi-instance image annotation. In: *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pp. 651–658. IEEE Computer Society, Washington, DC, USA. ISBN 978-1-4577-1101-5.
Available at: http://dx.doi.org/10.1109/ICCV.2011.6126300

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *CoRR*, vol. abs/1605.07146.
Available at: http://arxiv.org/abs/1605.07146

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

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

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

Zhu, F., Li, H., Ouyang, W., Yu, N. and Wang, X. (). Learning Spatial Regularization with Image-level Supervisions for Multi-label Image Classification.
Available at: https://arxiv.org/pdf/1702.05891.pdf

Zhu, J., Liao, S., Lei, Z. and Li, S.Z. (2017 February*a*). Multi-label convolutional neural network based pedestrian attribute classification. *Image Vision Comput.*, vol. 58, no. C, pp. 224–229. ISSN 0262-8856.
Available at: https://doi.org/10.1016/j.imavis.2016.07.004

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