

**University of Adelaide**

# **Deeplearning for multi-label scene classification**

by

**Junjie Zhang**

A thesis submitted in fulfillment for the  
degree of Master

Under Supervised by  
Chunhua Shen and Javen Shi  
School of Computer Science

February 2016

# **Declaration of Authorship**

I, Junjie Zhang, declare that this thesis titled and the work presented in it are my own.  
I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Adelaide.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at the University of Adelaide or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

---

Date:

---

## *Abstract*

Scene classification is an important field in computer vision. For similar weather condition, there are some obstacles for extracting features from outdoor images. In this thesis, we present a novel approach to classify cloudy and sunny weather. Inspired by recent study of a deep convolutional neural network and the spatial pyramid matching, we generate a model based on the ImageNet dataset. Starting with parameters learned from the other classification task, we fine-tune the model via the outdoor images. Experiment demonstrates that our classifier can achieve the state of the art accuracy.

Multi-label learning is a variant of supervised learning where the task is to learn a set of examples, which can belong to multiple classes. This is a version of popular multi-class classification problems in which each sample has one class label only. It can apply in a wide range of applications, which include text categorisation, semantic image labelling etc.. A lot of research work has been done on multi-label learning with different approaches. In this thesis, I trained a neural network from scratch up based on the generated artificial images. The model is learned by minimising error function based on the Hamming distance, through the backpropagation. The model has high capability of generalisation.

## *Acknowledgements*

I am grateful to my major supervisor, Prof. Chunhua Shen, and co-supervisor, Dr. Qinfeng Shi, whose expertise, understanding, and warmly support made it possible for me to work on the neural network that is of great interest to me. It is a pleasure working with them. Prof. Shen's dedication and keen interest to help his students had been solely and mainly responsible for completing my work. His timely advice, meticulous scrutiny, and scholarly advice have helped me to accomplish the tasks.

I would like to thank for my co-supervisor, Qinfeng Shi, for his time and effort on helping me understand research work and knowledge of machine learning.

I thank PhD candidate Teng Li who cooperated to work on the Weather Classification project. We set up experiment environment, analysed test results and discussed approaches. His prompt inspirations, timely suggestions with kindness, and enthusiasm work attitude have helped to achieve the perfect classification accuracy.

I am extremely thankful to research fellow, Guoshen Lin, for his kind help and discussion about the weather classification and the multi-label classification.

I would also like to thank staffs and visitors in the Australian Center for Visual Technologies (ACVT). Attending the reading group has enriched my knowledge in Computer Vision and learned a lot from them.

Finally, I would like to warmly thank the staffs in the School of Computer Science at the University of Adelaide. Julie Mayo, Sharyn Liersh and Jo Rogers have done excellent jobs on administration which helps students to be focus on their research work.

# Contents

<b>Declaration of Authorship</b>	<b>i</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>ix</b>
<b>Abbreviations</b>	<b>x</b>
<b>I Weather Classification</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Overview . . . . .	1
1.2 Statistical Pattern Recognition . . . . .	2
1.3 Artificial Neural Networks (ANNs) . . . . .	2
1.4 Weather Classification . . . . .	2
<b>2 Background</b>	<b>4</b>
2.1 Related Work . . . . .	4
2.2 Single-Layer ANNs . . . . .	5
2.3 Multi-Layer Networks . . . . .	8
2.4 Stochastic Gradient Descent (SGD) . . . . .	10
2.5 Backpropagation . . . . .	11
2.5.1 Training protocols . . . . .	13
2.6 Overfitting and Regularization . . . . .	13
2.6.1 Weight Decay . . . . .	15
2.6.2 Dropout . . . . .	15
2.7 Softmax Classifier . . . . .	16
2.7.1 Practical issues . . . . .	18
2.7.2 Error function . . . . .	18
2.8 Convolutional Neural Networks (CNN) . . . . .	19
2.8.1 Layers in CNN . . . . .	19
2.9 Spatial Pyramid Matching (SPM) . . . . .	21
2.10 Transfer Learning . . . . .	23

<b>3 Methodology</b>	<b>25</b>
3.1 Dataset . . . . .	25
3.2 Data Argumentation . . . . .	26
3.3 Spatial Pyramid Pooling (SPP) . . . . .	26
3.4 Convolutional Neural Networks Architecture . . . . .	27
<b>4 Experiment</b>	<b>30</b>
4.1 Training Neural Networks . . . . .	30
4.2 Fine-tuning Model . . . . .	31
4.3 Companion Experiments . . . . .	32
4.4 Experimental Results . . . . .	32
4.5 Architecture Analysis . . . . .	34
4.6 Effects of SPP . . . . .	36
4.7 Error Results . . . . .	37
4.8 Conclusion and Future Work . . . . .	37
<b>II Multilabel Learning</b>	<b>39</b>
<b>5 Introduction</b>	<b>40</b>
5.1 Overview . . . . .	40
5.2 Multi-Label Learning . . . . .	41
<b>6 Background</b>	<b>44</b>
6.1 Evaluation Metrics . . . . .	44
6.1.1 Example-based Metrics . . . . .	45
6.1.2 Label-based Metrics . . . . .	46
6.2 Learning Algorithms . . . . .	47
6.2.1 Problem Transformation Methods . . . . .	47
6.2.1.1 Binary Relevance (BR) . . . . .	47
6.2.1.2 Classifier Chains (CC) . . . . .	48
6.2.2 Algorithm Adaptation Methods . . . . .	49
6.2.2.1 Multi-label k-Nearest Neighbour (ML-kNN) . . . . .	49
6.2.2.2 Collective Multi-label Classifier (CML) . . . . .	50
<b>7 Methodology</b>	<b>52</b>
7.1 Artificial Dataset . . . . .	52
7.1.1 Generating Images . . . . .	53
7.2 Artificial Neural Networks (ANNs)	53
7.2.1 Network Architecture . . . . .	54
7.2.2 Error Function . . . . .	56
7.2.3 Cross Entropy . . . . .	58
7.2.4 Training and Testing . . . . .	60
<b>8 Experiment</b>	<b>63</b>
8.1 Dataset . . . . .	63
8.2 Details of Network . . . . .	63
8.3 Results . . . . .	66

8.4 Conclusion and Future Work . . . . .	68
--	----

<b>Bibliography</b>	<b>69</b>
---------------------	-----------

# List of Figures

2.1	Diagram of a perceptron [1]. . . . .	5
2.2	Threshold function . . . . .	6
2.3	Linear function . . . . .	6
2.4	Sigmoid function . . . . .	6
2.5	Tanh function . . . . .	6
2.6	The error surface for a single layer neural network[2]. . . . .	8
2.7	Two types of dataset. The left one can be separated by a single layer neural network. The right one cannot be separated by a single neural network. Generated from[3]. . . . .	8
2.8	Diagram of a feedforward neural network[4]. . . . .	9
2.9	The error surface for a multi-layer neural network[2]. . . . .	10
2.10	A multi-layer neural network can separate a complicated dataset. Generated from[3]. . . . .	11
2.11	Overfitting example, the left one has a decent generalization performance and the right one is overfitting[5]. . . . .	14
2.12	Illustration of dropout [6]. . . . .	16
2.13	The left is a fully connect regular neural network. The right is a CNN in 3 dimensions[7]. . . . .	19
2.14	Diagram for depth in a convolutional layer[7]. . . . .	20
2.15	Diagram of the Spatial Pyramid Matching[8]. . . . .	22
2.16	The left is traditional machine learning method. The right is transfer learning[9]. . . . .	23
3.1	2 Figures from the ImageNet[10]. . . . .	25
3.2	2 Figures from the Weather Dataset[11]. . . . .	26
3.3	A set of cropped patches from original image . . . . .	27
3.4	Diagram of the SPP layer[12] . . . . .	28
3.5	Architecture of the AlexNet[13] . . . . .	28
4.1	Learning Process . . . . .	33
4.2	Finetune Process . . . . .	33
4.3	ROC Curve . . . . .	34
4.4	A cloudy image and the feature maps from convolutional layers . . . . .	35
4.5	A sunny image and the feature maps from convolutional layers . . . . .	36
4.6	Visualisation of feature maps from the CNN model and the SPP model. The upper images are from the CNN model and the lower images are from the fine-tuned model. . . . .	37
4.7	Histogram distribution of vectors from FC7. The left is from CNN model and the right is from the SPP model. . . . .	37

4.8	Misclassified images[11]. . . . .	38
5.1	Example Image . . . . .	40
7.1	Colour Wheel Diagram[14] . . . . .	52
7.2	Multilabel samples and the RGB colour histograms. Three labels mean red, green and blue sequentially. . . . .	54
7.3	Network Topology For Multi-lable Classification . . . . .	55
8.1	Network Topology . . . . .	64
8.2	The test results for different number of hidden neurons. Blue circle for Sensitivity. Black plus for Specificity. Cyan star for Harmonic Mean. Red dot for Precision. Green x for F1 Score. . . . .	66
8.3	The learning speed for 200 neurons in the hidden layer. Blue circle for Sensitivity. Black plus for Specificity. Cyan star for Harmonic Mean. Red dot for Precision. Green x for F1 Score. . . . .	67
8.4	The ROC curve for 200 hidden neurons . . . . .	67
8.5	The ROC curve for 200 hidden neurons . . . . .	68

# List of Tables

3.1	Architecture of the model . . . . .	29
4.1	The first test is extracting features from the pre-trained CNN model and training a SVM classifier. The second test is similar with the first except for extracting features from the CNN model with a SPP layer. The third test is fine-tuning the model with AlexNet architecture. The fourth test is fine-tuning the CNN model with a SPP layer . . . . .	33
5.1	Multilabel $Y_1, \dots, Y_L \in 2^L$ . . . . .	41
8.1	Test results for different number of hidden neurons. . . . .	66

# Abbreviations

<b>SVM</b>	Support Vector Machine
<b>ANNs</b>	Artificial Neural Networks
<b>ROI</b>	Rogin of Interest
<b>SIFT</b>	Scale Invariant Feature Transform
<b>SGD</b>	Stochastic Gradient Descent
<b>BP</b>	BackPropagation
<b>MLE</b>	Maximum Likelihood Estimation
<b>MAP</b>	Maximum A Posteriori
<b>CNN</b>	Convolutional Neural Network
<b>ReLU</b>	Rectified Linear Unit
<b>SPM</b>	Spatial Pyramid Matching
<b>SPP</b>	Spatial Pyramid Pooling
<b>GPU</b>	Graphics Processing Unit
<b>BR</b>	Binary Relevance Classifier
<b>CC</b>	Classifier Chains Classifier
<b>CML</b>	Collective Multi-Label Classifier

## **Part I**

# **Weather Classification**

# Chapter 1

## Introduction

### 1.1 Overview

The computer is one of the most significant inventions in history. It provides huge power in the data processing field, like computer vision. Also, computer systems can aid people in daily life, for example driverless vehicles. However, the human brain still has compelling advantage in some fields, such as identifying our keys in our pocket by feel. The complex processes of taking in raw data and taking action based on the pattern are regarded as pattern recognition. Pattern recognition has been important for people in daily life for a long period and the human brain has developed an advanced neural and cognitive capacity for such tasks.

Weather classification is one of the most important pattern recognition tasks which relates to our work and lives. There are several major kinds of weather conditions, such as sunny, cloudy, rainy. People can classify them easily through their eyes. However, it is not an easy job for machines, especially in literature on computer vision.

In this thesis, we describe an approach to this problem of weather classification based upon the major big trend in machine learning, and more precisely, deep learning. It differs from the feature detectors method, which extracts features manually, and then trains a model to perform classification.

Compared to shadow learning which includes decision trees, SVM and naive bayes, deep learning passes input data through several non-linearity functions to generate features

and does classification based on those features. It generates a mapping and finds the optimal solution.

## 1.2 Statistical Pattern Recognition

In the statistical approach, a pattern is represented in terms of  $d$ -dimensional feature vectors and each element of the vector describes some subjects of the example. In brief, three components are essential for statistical pattern recognition. The first one is a representation of the model. The second one is an objective function used to evaluate the accuracy of the model. The third one is an optimisation method for learning a model with minimum errors.

## 1.3 Artificial Neural Networks (ANNs)

Artificial neural networks were proposed in the mid-20th century. The term is inspired by the structure of neural networks in the brain. It is one of the most successful statistical learning models used to approximate functions. Learning with ANNs yields an effective learning paradigm and has achieved cutting-edge performance in object classification.

The single-layer ANNs have shown great success for a simple model. For increasingly complex models and applications, multi-layer ANNs have exhibited the power of features learning. With hardware being developing at the same time, the demand for more efficient optimising and model evaluation methods has increased promptly.

Recent development in ANNs has greatly advanced the performance of state-of-the-art visual recognition systems. With implementing deep convolutional neural networks, ANNs achieve top accuracy in the ImageNet Challenge. The model has been used in related fields and performs well in pattern recognition.

## 1.4 Weather Classification

Weather classification is an important task in daily life. In this thesis, we are focused on two classes of weather conditions, sunny and cloudy.

There are some obstacles in front of weather classification. First, because the number of inputting pixels is high, for example a  $500 \times 500$  RPG image means 750,000 pixels, computation is expensive. Second, some simple middle level image characters are difficult to be recognised by machines, such as light, shading, sun shine. It is still not easy to detect these characters with high accuracy. Third, there are no decisive features, for example brightness and lightness, to classify scenes. For example, sun shine can be both found in sunny and cloudy weather. Last but not least, outdoor images have various backgrounds.

# Chapter 2

## Background

### 2.1 Related Work

There has been much research done on the task of weather classification. Most works follow the procedure of extracting features and performing classification [15–18]. Some works use low-level features, like colour [19], texture [20, 21]. Some works use filters or segmentation [20, 22] to extract high-level features, such as sky, shadow [11]. Some works use statistical measurement methods [12, 16] to analyse low-level feature distributions.

Generally, the traditional methods follow three basic steps[16, 23]. First, some Regions of Interest (ROIs), for example, sky and shadow, are extracted from a weather image. Then, histogram descriptors are used to represent the difference between them. Finally, a classifier, e.g., SVM, is built based on the extracted features.

Most previous works are based on the discriminative model. They extract human recognisable features to classify images. This type of shadow learning depends mainly on the quality of features and human’s prior knowledge. An image without prior features or poor features is hard to be classified. Furthermore, the methods require a lot of work on data pre-processing and are not flexible. The approaches depend on structural information to categorise an image into one of the classes. Structural information is concluded from illumination invariant features, like SIFT. However, previous works have limitation on classifying diverse images. It is hard to list all the factors which determinate the weather conditions.

## 2.2 Single-Layer ANNs

Artificial neurons were introduced as information processing devices more than fifty years ago[24]. Following the early work, perceptrons were deployed in layers to do pattern recognition tasks. A lot of resources were invested in the research capability of learning perceptrons theoretically and experimentally. As shown in Figure 2.1, a perceptron computes a weighted summation of its  $n$  inputs and then thresholds the result to give a binary output  $y$ . We can treat  $n$  input as an vector with  $n$  elements, and represent the vector as either class A(for output +1) or class B(for output -1).

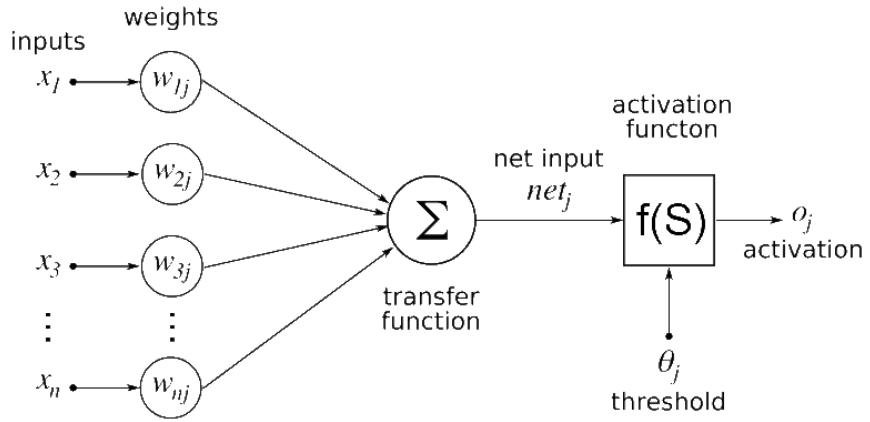


FIGURE 2.1: Diagram of a perceptron [1].

Each output is updated according to the equation:

$$y_i = f(h_i) = f \left( \sum_j w_{ij} x_j \right) \quad (2.1)$$

where  $x_j$  is the  $j$ th input,  $y_i$  is the  $i$ th output, and  $h_i$  is the net input into node. The weight  $w_{ij}$  connects the  $j$ th input and the  $i$ th output, and the threshold function  $f(h)$  is the activation function and usually makes up the form

$$f(x) = sign(x) = \begin{cases} -1 & h < 0 \\ 1 & h \geq 0 \end{cases} \quad (2.2)$$

and it is plotted out in figure 2.2

Besides the threshold function, there are several deterministic action functions.

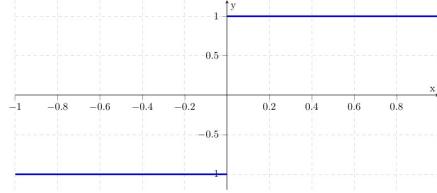


FIGURE 2.2: Threshold function

- Linear function

$$f(x) = x \quad (2.3)$$

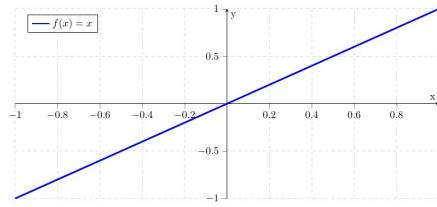


FIGURE 2.3: Linear function

- Sigmoid function

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.4)$$

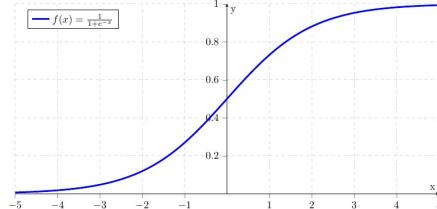


FIGURE 2.4: Sigmoid function

- Tanh function

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.5)$$

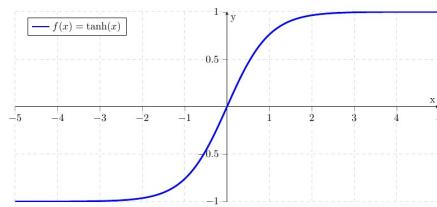


FIGURE 2.5: Tanh function

We can also represent equation 2.2 in vector notation, as in

$$y = f(h) = f(w \cdot x) \quad (2.6)$$

where  $w$  and  $x$  can be regarded as  $n \times 1$  dimensional column vectors, and  $n$  is the dimension number of input data. The term  $w \cdot x$  in equation 2.6 constructs a  $(n - 1)$ -dimension hyperplane which passes the origin. The hyperplane can be shifted by adding an parameter to equation 2.1, for example

$$y = f(h) = f(w \cdot x + b) \quad (2.7)$$

We can have the same effect by putting a constant value 1 and increasing the size of  $x$  and  $w$  by one. The extra weight  $w$  with fixed value 1 is called *bias weight*. It is adaptive like other weights and provides flexibility to hyperplane. Then we get:

$$y = f(h) = f\left(\sum_{i=0}^n w_i x_i\right) \quad (2.8)$$

The aim of learning is to find a set of weights  $w_i$  so that:

$$\begin{aligned} y &= f\left(\sum_{i=0}^n w_i x_i\right) = 1 \quad x \in ClassA \\ y &= f\left(\sum_{i=0}^n w_i x_i\right) = 0 \quad x \in ClassB \end{aligned}$$

The single layer neural network classifier is simple to implement, while it can only support a linear decision boundary. We can build a simple neural network to acquire intuition behind the mathematical theory. The network has no bias and one neuron which means it has one weight, say  $w_1$ . We implement a logistic sigmoid activation function on the multiply of input data and weight  $w_1$ . Therefore, the network can map the input data  $a_0$  onto an output  $a_{out}$  based on the function

$$a_{out} = f(a_0 w_1) \quad (2.9)$$

where  $f()$  is the logistic function. Supposing that an input data 1 maps to an output data 1, we can compute the value of the error function for each possible value of  $w_1$ . Feeding value of  $w_1$  in range  $(-10, 10)$ , we can plot error surface in figure 2.6.

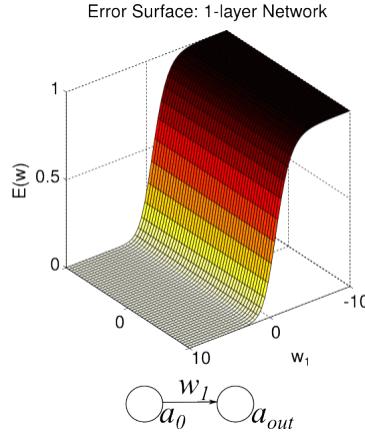


FIGURE 2.6: The error surface for a single layer neural network[2].

The single layer neural network has a decision boundary which is linear. However, it is clear this is a very limited class of decision boundary. It can be illustrated by two types of datasets in figure 2.7 .



FIGURE 2.7: Two types of dataset. The left one can be separated by a single layer neural network. The right one cannot be separated by a single neural network. Generated from[3].

## 2.3 Multi-Layer Networks

Single layer networks have some important limitation in terms of representing range of functions. We are seeking to learn the nonlinearity as the linear discriminant. To improve the representation capability, we can stack layers network. This is the approach of multi-layer neural networks. multi-layer neural networks implement linear discriminants via

mapping the input data to a nonlinear space. They can use fairly simple algorithms to learn the form of the nonlinearity from the training data.

In the thesis, we limit multi-layer neural networks in the subset of feedforward networks. Feedforward neural networks can provide a general mechanism for representing nonlinear functional mapping between a set of input data and a set of labels. The figure 2.8 is a feedforward neural network having two layers of adaptive weights.

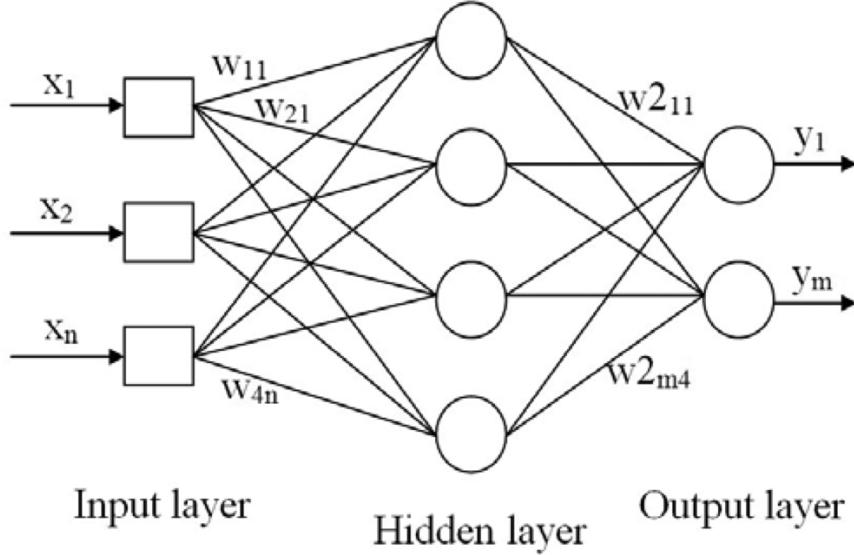


FIGURE 2.8: Diagram of a feedforward neural network[4].

In the example, the middle column perceptrons act as hidden neurons. The network has  $n$  inputs, 4 hidden neurons and  $m$  output neurons. The network diagram represents the function in the form

$$y_m = \hat{f} \left( \sum_{j=0}^m w_{j4}^{(2)} f \left( \sum_{i=0}^n w_{4i}^{(1)} x_i \right) \right) \quad (2.10)$$

In equation 2.10, the outer activation function could be different with the inner one.

There are some choices for activation functions, the sigmoid and the tanh functions are related and continuous activation functions. The logistic function, also named the sigmoid function, can be represented as

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.11)$$

Its outputs lie in the range  $(0, 1)$ . We can do a linear transformation  $\hat{x} = x/2$  on the input data and a linear transformation  $\hat{y} = 2y - 1$  on the output. Then we can get an

equivalent activation function  $\tanh$  which can be represented:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.12)$$

The three layers neural network is capable of approximating any function with enough hidden neurons, which means the network with two layers of weights and the sigmoid function can provide a high degree of accuracy in classification problems.

Again, we can use a simple example to illustrate the power of the multi-layer neural network. In this example, we have one input, one hidden neuron and one output. There is no bias in the input layer and hidden layer. There are two weights existing in the network, say  $(w_1, w_2)$ , and the output can be calculated via

$$a_{out} = f(f(a_0 w_1) w_2) \quad (2.13)$$

where  $f()$  is the sigmoid function. With varying  $w_1$  and  $w_2$ , the error surface can be represented in figure 2.9. And the samples which cannot be separated by the single-layer

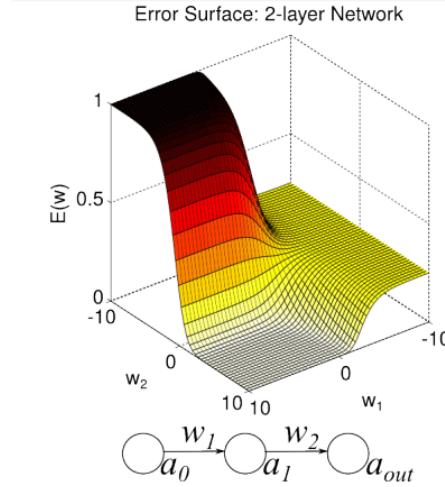


FIGURE 2.9: The error surface for a multi-layer neural network[2].

neural network can be done by a multi-layer neural network.

## 2.4 Stochastic Gradient Descent (SGD)

Because the weight space in neural networks is continuous, training can reach the optimal weights value through optimization algorithms, which means minimising loss value of

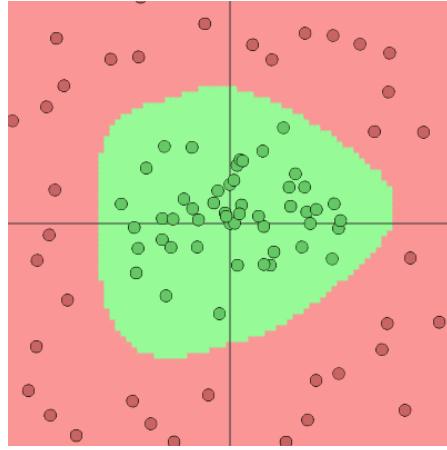


FIGURE 2.10: A multi-layer neural network can separate a complicated dataset. Generated from [3].

function

$$L(f_w) = \sum L(y, f_w(x)) \quad (2.14)$$

Gradient descent is a first-order optimisation algorithm which starts from a random point, and then finds a nearby minimum point. It can converge on a minimum possible loss.

The SGD is a subtype of gradient descent. It only considers a single training point and moves to a nearby point based on

$$w = w - \eta \Delta L(w) = w - \eta \sum_{i=1}^n \Delta L_i(w) \quad (2.15)$$

where  $\eta$  is the learning rate and  $L_i(w)$  is the value of the loss function at the  $i^{th}$  sample. The Robbins-Siegmund theorem [25] provides the approaches to almost surely convergence to a global minimum under relatively mild assumptions, moreover, it is fast and effective in practice.

## 2.5 Backpropagation

Multi-layer neural networks can represent mapping from the input data to the output classes. How to learn a suitable mapping method from the training dataset? And there is no explicit mapping between the outputs and the hidden neurons. This will be resolved by a popular learning algorithm, backpropagation.

Because networks have differentiable activation functions, the activation of the output neurons can be propagated to the hidden neurons with respect to weights and bias. If we have an error function, we can evaluate derivatives of the error and update the weights to minimise the error function via some optimization methods.

Backpropagation can be applied to find the derivatives of an error function related to the weights and bias in the network via two stages. First, the derivatives of the error functions, for instance sum-of-squares and Jacobian, with respect to the weights must be evaluated. Second, a variety of optimisation schemes can be implemented to compute the adjustment of weights based on derivatives. After passing data through the network, we can get the output result. It updates weight changes based on the gradient descent. Suppose the network has  $i$  inputs,  $h$  hidden neurons and  $k$  outputs. The update equation can be represented:

$$w(j+1) = w(j) + \Delta w(j) \quad (2.16)$$

where  $\Delta w(j)$  defined:

$$\Delta w(j) = -\eta \frac{\partial E}{\partial w} \quad (2.17)$$

For the hidden to output layer weights

$$\Delta w(jk) = -\eta \frac{\partial E}{\partial w_{jk}} = -\eta \delta_k y_j \quad (2.18)$$

where

$$\delta_k = \frac{\partial E}{\partial a_k} = (y_k - t_k)y_k(1 - y_k)$$

For the hidden layer weights

$$\Delta w(ij) = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \delta_j y_i \quad (2.19)$$

where

$$\delta_j = \frac{\partial E}{\partial a_j} = \sum_k \delta_k w_{jk} y_j (1 - y_j)$$

The  $\delta_j$  of a hidden neuron is based on the  $\delta_k$  of the output neurons to which it links. To minimise the error function  $E$  with the gradient descent, it needs the backwards propagation of errors.

### 2.5.1 Training protocols

In supervised learning, we have training dataset which are data with labels. We can use the neural networks to find the output of the training data and adjust the weights to optimal values. There are mainly three types of training protocols, stochastic, batch and online training. In stochastic training, we randomly choose samples from the training dataset and update weights every time, depending on the output of the network. In batch training, we use some samples and pass them through the network, and then update weights. In online training, each sample of the training dataset is used once and weights are updated each time. We usually represent one time of passing all training dataset through neural networks as one epoch.

It is worth noting batch learning. In large scale applications, the training dataset can be over millions of samples. It is time consuming to compute the loss function over all training datasets in order to update weights once. It is a practical approach that computes the gradient over a batch of training datasets. Does it have harmful effects on generalization? It depends on the correlations among the training dataset. Consider there are more than one million images in the ImageNet dataset which is made up of only 1000 categories. If the images in a batch are selected evenly from each category, the gradient from the batch is a good enough approximation of the gradient of the full training dataset. Therefore, batch learning leads to faster convergence by evaluating the batch gradients to update weights frequently.

## 2.6 Overfitting and Regularization

Overfitting is a common phenomenon that a model has high performance on the training dataset, but is poor on the testing dataset in classification tasks. Thus, a classification problem with two classes and two input variables, (left figure in 2.11), shows decent decision boundaries. With increasing complexity of the model, the decision boundaries become more complex and fit the training dataset extremely well. From the example in figure 2.11, it is clear that a model, whose complexity is neither too small nor too large, has the best generalization performance.

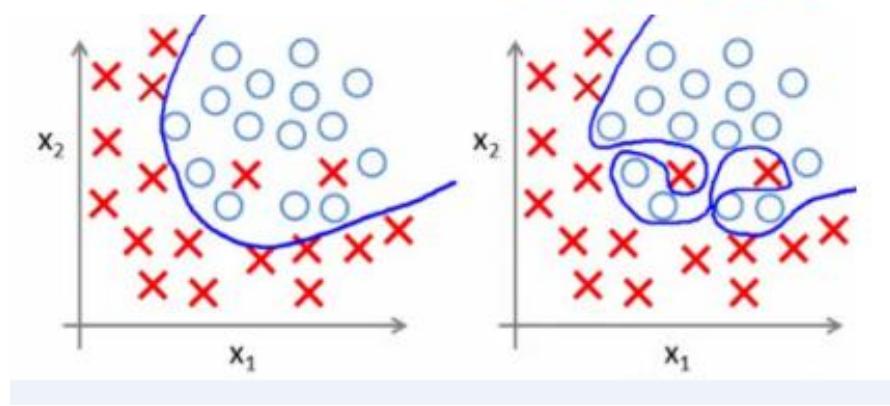


FIGURE 2.11: Overfitting example, the left one has a decent generalization performance and the right one is overfitting[5].

In order to find the optimal complexity of the neural network, there are mainly two approaches. One is to change the adaptive parameters, like neuron numbers in hidden layers. This is named structural stabilisation. It can be approached from two directions. We can start from a small network and increase the layer number or neuron number in the training process and arrive at an effective neural network architecture. The other one is to start from a large network and prune out layers or neurons in the training process to achieve the optimal network. Another fundamental basic method to control the complexity of a model is to use regularisation which includes adding a penalty term in the error function. The degree of regularisation can be adjusted by scaling the term via a multiplicative parameter.

Regularization helps to smooth decision boundary surface by introducing a penalty term  $\Omega$  to the error function

$$\hat{E} = E + \lambda\Omega \quad (2.20)$$

where  $E$  is the standard error function, the  $\lambda$  adjustment and the extent of the penalty  $\Omega$  effects on the solution. The task of the learning is to minimize the total error function  $\hat{E}$ . It needs to compute the derivatives of  $\Omega$  with respect to the weights efficiently. A model, which has high accuracy in the training dataset, has a small value for  $E$ . At the same time, the smooth error surface of neural networks give a small value for  $\Omega$ .

A number of regularisers have been performed in applications, like weight decay, early stopping, training with noise, weight sharing etc..

### 2.6.1 Weight Decay

In order to smooth the decision boundary surface, weight values should be small. The weight decay regularizer can be represented:

$$\Omega = \frac{1}{2} \sum_i w_i^2 \quad (2.21)$$

where the sum includes all weights and bias. Weight decay of this form leads to major improvements in empirical generalisation [26]. Intuitively, in equation 2.21, the smaller the weights are, the better the regulariser is. Usually, the derivatives of the total error function with respect to the weight are used to train the neural network. The network is trained by gradient descent in the continuous time limit. The weights will evolve with time  $t$

$$\frac{w}{t} = -\eta \nabla E = -\eta \lambda w \quad (2.22)$$

where  $\eta$  is the learning rate. And the equation has solution

$$w(t) = w(0) \exp(-\eta \lambda t) \quad (2.23)$$

and the exponential function reduces weights to zero quickly. Weight decay is also named L2 regularisation.

### 2.6.2 Dropout

Neural networks with deep layers and a large amount of neurons is a powerful learning machine. However, the more parameters the network has, the easier it is overfitting. Recently, dropout[6] is a simple and extremely effective regularisation technique which complements the other methods. At the training process, random neurons are selected with some probability  $p$  to update their associated weights, and the others are inactive. In other words, only a reduced network is trained, based on the input data at the training stage. At the testing process, there is no dropout applied and all neurons are active. The dropout method is replaced by performing a scaling of layer outputs by the same probability  $p$ . This method can keep the outputs of neurons to be the same in both training and testing process. For example, if a neuron has  $p$  probability to be dropped out in the training process, the neuron should give an output  $l$  without dropout in the

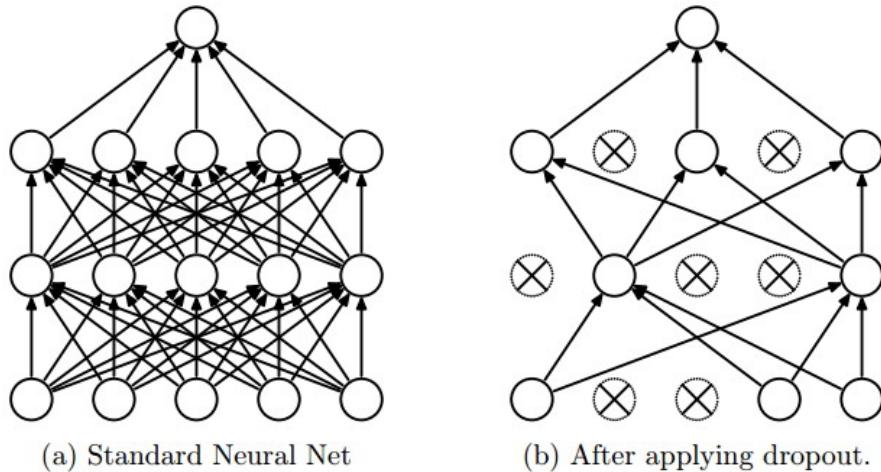


FIGURE 2.12: Illustration of dropout [6].

testing stage. Then we should apply  $(p * l + (1 - p) * 0)$  on the output, because the output has  $(1 - p)$  probability to be 0.

## 2.7 Softmax Classifier

The softmax function, also named normalised exponential, is a generalisation of the logistic function which squeezes a  $d$  dimensional arbitrary real values vector to a  $d$  dimensional vector of real values in the range  $(0, 1)$  that add up to 1. Because the softmax function is the gradient log normaliser of categorical probability distribution, it can be used in probabilistic multiclass classification problems.

The softmax function derives from log linear models and interprets the weights in terms of convenient odds ratios. It can constrain the input values of the final layer to be positive and sum of them to be 1.

The softmax layer begins the same way as the normal layer which forms the weighted inputs  $z_j^L = \sum_k w_{jk}^L x_k^{L-1} + b_j^L$  where  $L$  is the layer number,  $k$  is the input data number and  $j$  is the output neuron number. It, then, implements a softmax function to the  $z_j^L$  and activates the  $j$  output neuron:

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \quad (2.24)$$

The equation 2.24 implies that the output values are all positive and the sum of all values  $\sum_k a_k$  is 1.

The softmax classifier can be used to handle a multiclass classification. For a training dataset  $(x_1, y_1), \dots, (x_m, y_m), y_i \in \{1, 2, \dots, K\}$  of  $m$  labelled examples, the label  $y$  can have  $K$  different values.

To an unseen sample  $x$ , we will use a hypothesis to estimate the probability  $P(y = k|x)$  for each value  $k = 1, \dots, K$ . For example, we want to compute the probability of the class label on each of  $K$  different possible values. The neural network will then output a  $K$  dimensional vector which represents  $K$  estimated probabilities.

$$h_W(x) = \begin{bmatrix} P(y = 1|x; W) \\ P(y = 2|x; W) \\ \vdots \\ P(y = K|x; W) \end{bmatrix} = \frac{1}{\sum_{j=1}^K \exp(W^{(j)\top} x)} \begin{bmatrix} \exp(W^{(1)\top} x) \\ \exp(W^{(2)\top} x) \\ \vdots \\ \exp(W^{(K)\top} x) \end{bmatrix} \quad (2.25)$$

Where  $W^j$  are the weights of the model and the normalised distribution ensures that the sum is one.

On the one hand, the cross entropy can be used to interpret the softmax classifier. The cross entropy between actual distribution  $p$  and a predicted distribution  $q$  is represented as

$$H(p, q) = - \sum_x p(x) \log q(x) \quad (2.26)$$

Hence, the task of the softmax classifier is to minimise the cross entropy between the actual distribution and the predicted distribution.

On the other hand, the softmax classifier can be interpreted in probability view. Given a sample  $(x_i, y_i)$  and parameters  $W$ , we can compute the normalised probability:

$$P(y_i | x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \quad (2.27)$$

where  $f_{y_i}$  is the score predicted by the model with weights  $W$ . Therefore the normalised probabilities are computed by exponentiating the values and dividing the sum of all values. We can minimise the negative log likelihood of the ground truth labels which can be regarded as performing the Max Likelihood Estimation(MLE). Aside from MLE, the Maximum a posteriori(MAP) can also be used to evaluate the performance of a model.

### 2.7.1 Practical issues

From a numerical view, the exponential computation is easy overflow. Thus, the output of the softmax function is not numerically stable through computing  $e^{f_{y_i}}$  and  $\sum_j e^{f_{y_j}}$  directly. The implementation requires a normalisation trick. It is mathematically equivalent to multiplying a constant  $C$  with both the top and bottom of the fraction.

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C \sum_j e^{f_j}} = \frac{e^{f_{y_i} + \log C}}{\sum_j e^{f_j + \log C}} \quad (2.28)$$

where  $C$  can be any positive value. The constant  $C$  does not change the output value, but it can improve the numerical stability of the computation. An experienced choice of  $C$  is to set  $\log C = -\max f_j$ , and it can shift the vector  $f$  to preserve the highest value as 0.

### 2.7.2 Error function

An error function is used to evaluate the performance of a model. We will generate an error function for softmax regression. An indicator function,  $I\{\cdot\}$ , is introduced to represent the accuracy for each label. If the predicted result corresponds to the actual label, say  $y^{(i)} = k$ , the indicator function returns 1, otherwise 0. The error function will be defined:

$$L(W) = - \left[ \sum_{i=1}^m \sum_{k=1}^K I\{y^{(i)} = k\} \log \frac{\exp(W^{(k)\top} x^{(i)})}{\sum_{j=1}^K \exp(W^{(j)\top} x^{(i)})} \right] \quad (2.29)$$

where this generates the logistic regression error function

$$L(W) = - \left[ \sum_{i=1}^m (1 - y^{(i)}) \log(1 - h_W(x^{(i)})) + y^{(i)} \log h_W(x^{(i)}) \right] \quad (2.30)$$

$$= - \left[ \sum_{i=1}^m \sum_{k=0}^1 I\{y^{(i)} = k\} \log P(y^{(i)} = k | x^{(i)}; W) \right] \quad (2.31)$$

Similar to the logistic regression error function, the softmax error function sums over the predicted different  $K$  values of the classes. In the softmax regression, the posterior probability distribution can be represented:

$$P(y^{(i)} = k | x^{(i)}; W) = \frac{\exp(W^{(k)\top} x^{(i)})}{\sum_{j=1}^K \exp(W^{(j)\top} x^{(i)})} \quad (2.32)$$

It is not easy to solve equation 2.30. Usually an optimisation algorithm can approximate the optimal value. Taking derivative with respect to weights, we can get the entire gradient

$$\nabla_{W^{(k)}} L(W) = - \sum_{i=1}^m \left[ x^{(i)} \left( 1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; W) \right) \right] \quad (2.33)$$

We can take partial derivative of  $L(W)$  with respect to the  $j$ th element of  $W^{(k)}$ .

## 2.8 Convolutional Neural Networks (CNN)

Convolutional Neural Networks[27] are widely applied in image understanding and achieve top rank in an image classification competition[13]. Compared to regular neural networks, CNN architecture assumes that the inputs are images and pixels are related in region. Regular neural networks fully connect layers and the method causes expensive computation.

CNN architecture has neurons arranged in 3 dimensions, say width, height and depth. For example, there is an image which has dimensions  $32 \times 32 \times 3$ . The neurons in a layer will connect to a customised region of the previous layer. Moreover, the final output layer has dimensions  $1 \times 1 \times d$ , where  $d$  is the number of classes. The dimensions are reduced from 3072 to  $d$ . The output is a single vector of class scores.

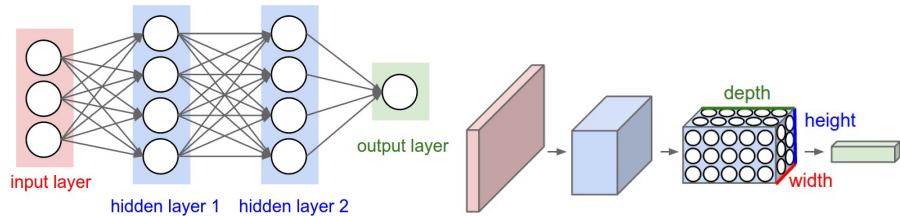


FIGURE 2.13: The left is a fully connect regular neural network. The right is a CNN in 3 dimensions[7].

### 2.8.1 Layers in CNN

CNNs have three main types of layers, convolutional layer, pooling layer and fully connected layer. Every layer transforms the input to the output via a differentiable function.

1. Convolutional Layer

2. ReLU Layer
3. Pooling Layer
4. Fully Connected Layer

Following layer by layer, CNN transforms an image from a set of pixel values to the final class scores.

**Convolutional Layer** is the key component of CNN and its output can be represented as neurons shaped in 3D volume. A set of learning filters make up the convolutional layer's parameters. Although the size of filter is flexible, it is usually small. During the feedforward process, each filter slides across the input volume and produces a 2-dimensional feature map. In each slide, the input and the filter have a dot product computation. If there are some specific feature at some spatial position, they can be learnt by filters.

Local connectivity is the key properties of the CNN. Regular neural networks use fully connecting layers. That is unaffordable for images, even with high capability hardware. Instead, each neuron connects a local region of the input only. There are two main benefits from local connectivity. One is reducing parameters significantly and controlling overfitting. The other one is from a key image property. Pixels are strongly correlated with nearby pixels. This can be regarded as a local receptive field which can retrieve information from subregions of the image.

To control the output volume arrangement, three parameters are introduced. They are depth, stride and zero-padding. In convolutional layer, depth controls the number of neurons which connect the same subregion of the input volume. All the neurons learn the different features from the input volume. For example, the neurons along the depth in the convolutional layer after the input data layer can activate existence of various edges, colour, etc.. Stride is another parameter which

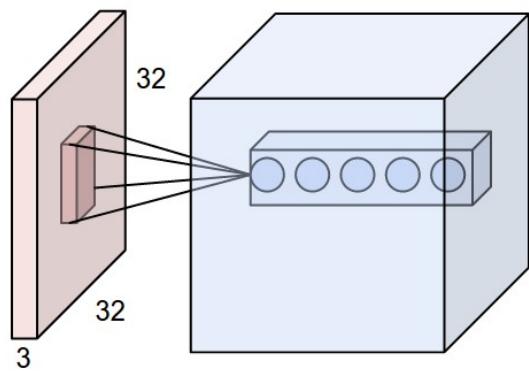


FIGURE 2.14: Diagram for depth in a convolutional layer[7].

controls the spatial position of the nearby depth column of neurons. The smaller the stride is, the more overlapping receptive fields are shared by the nearby columns. Last, zero-padding on border makes output volume to keep the same dimension with the input volume.

A scheme, named parameter sharing, is implemented in convolutional layer to limit the number of parameters. The scheme suppose that a filter, which is helpful to compute at some spatial positions, should be helpful to compute another position.

**ReLU Layer** is the abbreviation of Rectified Linear Units. The neurons in the layer operate the non-saturating activation function  $f(x) = \max(0, x)$  over the result of dot product in convolutional layers. The layer increases non linearity to network without losing the receptive fields of convolutional layers.

**Pooling Layer** is a mechanism of downsampling. It is usually appended after the convolutional layer to progressively decrease the spatial size of the feature map. It leads to decrease network parameters and makes the neurons in pooling layer to be relatively insensitive to small shifts of the image.

**Fully Connected Layer** takes feature maps, which have high level representation, from previous layers. Its difference with convolutional layer is that it connects to all neurons in previous layer instead of a receptive field only.

## 2.9 Spatial Pyramid Matching (SPM)

The spatial pyramid matching[8] is used to classify high-level semantic attributes, based on low-level features. The method subdivides an image in several different levels of resolution and counts features falling in each spatial bin. It extends bags of features and derives spatial information from the image.

Let two sets of vectors  $X$  and  $Y$  be in a  $d$ -dimensional feature space. The SPM is used to find the approximate correspondence between them. In brief, the SPM places a chain of grids over the feature space and takes the sum of number of matches occurring at each level of resolution. The points falling into the same grid are matched and the match points in finer resolutions have higher weights. Specifically, a chain of grids at resolutions  $0, \dots, L$ , have  $2^0, \dots, 2^L$  cells respectively.  $H_X^l$  and  $H_Y^l$  are the histograms

### Spatial Pyramid Matching (SPM)

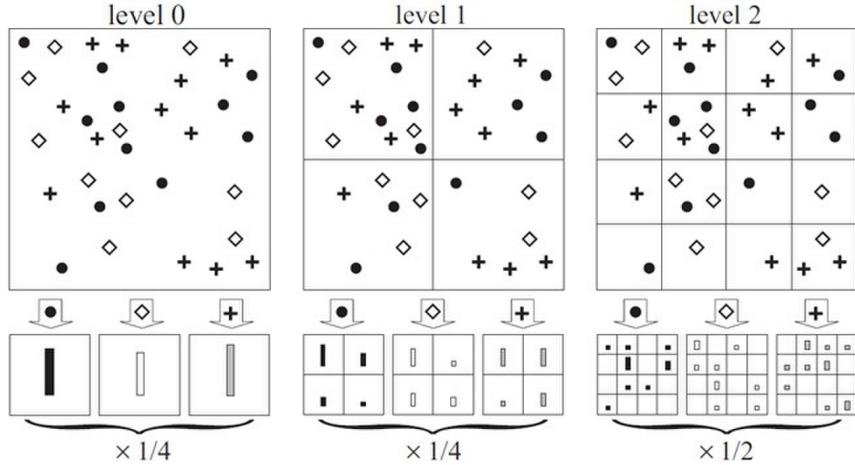


FIGURE 2.15: Diagram of the Spatial Pyramid Matching[8].

of  $X$  and  $Y$  at the  $l$  resolution, so that the number of points in the  $i$ th cell from  $X$  and  $Y$  can be represented as  $H_X^l(i)$  and  $H_Y^l(i)$ . The histogram intersection function denotes the number of matches at level  $l$

$$I(H_X^l, H_Y^l) = \sum_{i=1}^D \min(H_X^l(i), H_Y^l(i)) \quad (2.34)$$

Because the match points found in the  $l$  level include the match points found in the finer level  $l+1$ , the number of new match points at level  $l$  is  $I^l - I^{l+1}$ . The weight assigned to level  $l$  is  $\frac{1}{2^{L-l}}$ . Putting all together, the pyramid match kernel can be represented:

$$K^L(X, Y) = I^L + \sum_{l=0}^{L-1} \frac{1}{2^{L-l}} (I^l - I^{l+1}) \quad (2.35)$$

$$= \frac{1}{2^L} I^0 + \sum_{l=1}^L \frac{1}{2^{L-l+1}} I^l \quad (2.36)$$

We perform the SPM in a 2-dimensional image space and apply standard vector quantisation in the feature space. All feature vectors are quantised into  $M$  discrete types and only the same type of features match to one another. Two sets of 2-dimensional vectors,  $X_m$  and  $Y_m$ , represent the coordinate of the features of type  $m$  in the individual image. The match kernel of two images is the sum of total channel kernels

$$K^L(X, Y) = \sum_{m=1}^M K^L(X_m, Y_m) \quad (2.37)$$

## 2.10 Transfer Learning

In the literature on machine learning, the transfer learning[28] focuses on storing knowledge from one domain and applying it to a related problem. In other words, the relevant knowledge, learned from previous tasks, can be applied to new tasks. The closer a new job relates to previous knowledge, the more easily it can be solved. In contrast, other machine learning methods solve problems independently.

The transfer learning has three benefits. One is saving time of preprocessing data. Collecting and processing raw data are time consuming and expensive in each task. It can reduce volume of required data significantly, based on existing knowledge which is extracted from previous learning tasks. The other one is saving time of learning the model from scratch up. Usually, training a model from scratch up is time consuming. The transfer learning can save a lot of time. The last one is to avoid the risk of overfitting. With insufficient training data, a complicated model is easy to overfitting. The transfer learning method can control overfitting because of existing knowledge.

## Traditional ML vs. TL

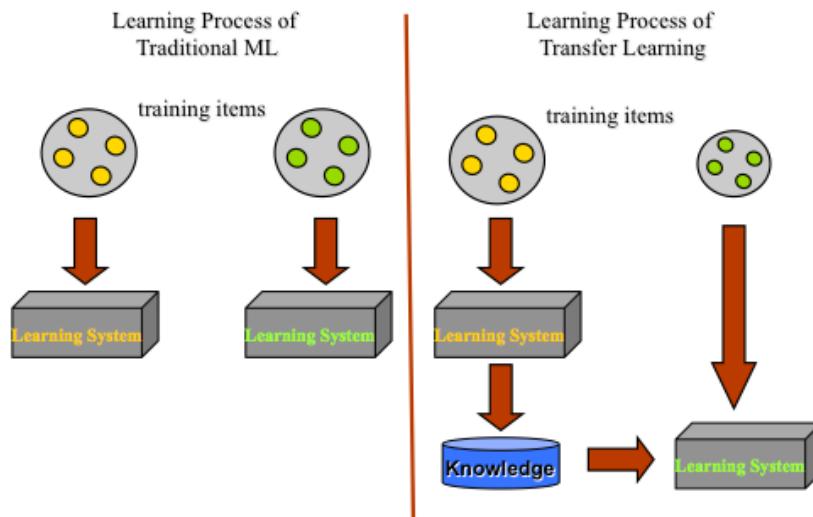


FIGURE 2.16: The left is traditional machine learning method. The right is transfer learning[9].

A domain can be represented

$$D = \{X, P(X)\} \quad (2.38)$$

where  $X$  is the feature space and  $P(X)$  is the marginal probability distribution.

In the transfer learning, there are two main challenges:

1. Which part of previous knowledge is useful to the new task?
2. How to represent the existing knowledge in the new model?

The first challenge arise, when we evaluate the relation between the previous knowledge and the current task. The correlated knowledge for the task can improve the performance of the model. Meanwhile, the unrelated and the negative correlated knowledge are useless, or even harmful for the task. After evaluating the useful knowledge, new learning algorithms should be developed to transfer the knowledge. This leads to the second challenge. Because knowledge can have different representations, the translation process must keep the accuracy and minimise the loss of knowledge.

# Chapter 3

## Methodology

### 3.1 Dataset

There are over 15 million labelled images in the ImageNet database which has 22,000 categories. The images were collected from the Internet and labelled manually. From 2010, the competition, named the ImageNet Large-Scale Visual REcognition Challenge(ILSVRC)[10], has been held annually. The competition uses a subset of dataset which contains 1000 images in 1000 categories. With 50,000 validation images and 150,000 testing images, there are over 1 million in total.



FIGURE 3.1: 2 Figures from the ImageNet[10].

The weather dataset [11] contains 10,000 images for two categories evenly, cloudy and sunny. They were collected from three sources, the Sun Dataset[29], the Labelme

Dataset[30] and the website, Flickr. They were classified manually and similar images were removed. No unambiguous images exist in the dataset.

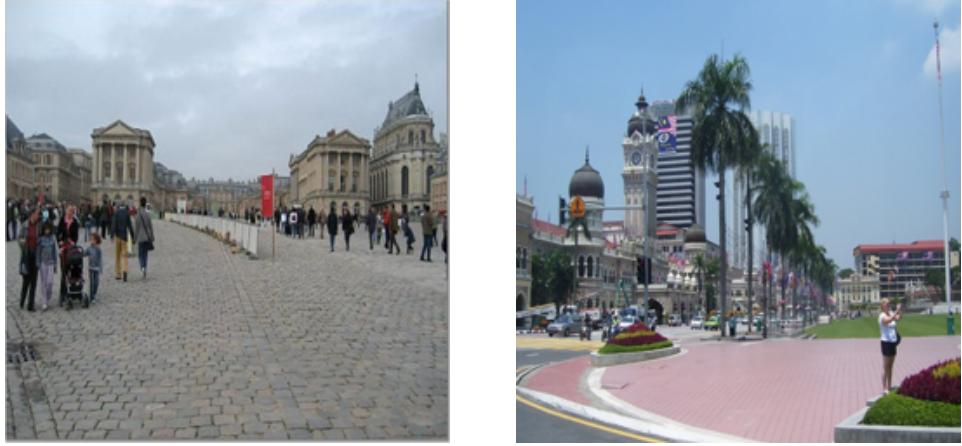


FIGURE 3.2: 2 Figures from the Weather Dataset[11].

In general, the two datasets are different. The ImageNet dataset is used for object classification and the weather dataset is used for scenes classification. The objects of the images are different too. Because the images, of the two datasets, are different size, we have to resize them to a fixed  $224 \times 224$  to feed into CNN.

## 3.2 Data Argumentation

The CNN architecture[13] has about 60 million parameters, and it is essential to avoid overfitting. One of methods is data argument. The dataset is artificially enlarged by cropping and horizontal reflecting images. To each  $256 \times 256$  image, the network extracts five  $224 \times 224$  patches from four corners and center and reflects them horizontally. Hence, there are 10 patches for 1 image in total. Figures 3.3 shows the method of data argument.

## 3.3 Spatial Pyramid Pooling (SPP)

The factors, which have effects on scene classification, can be at any spatial position in an image. This could be a problem to CNN, because it was built up to recognize the objects in the center of images. The SPP layer can help to solve the problem.

The SPP layer is deployed behind the fifth convolutional layer. A set of bins are set to discern different local information from the output of the fifth convolutional layer.

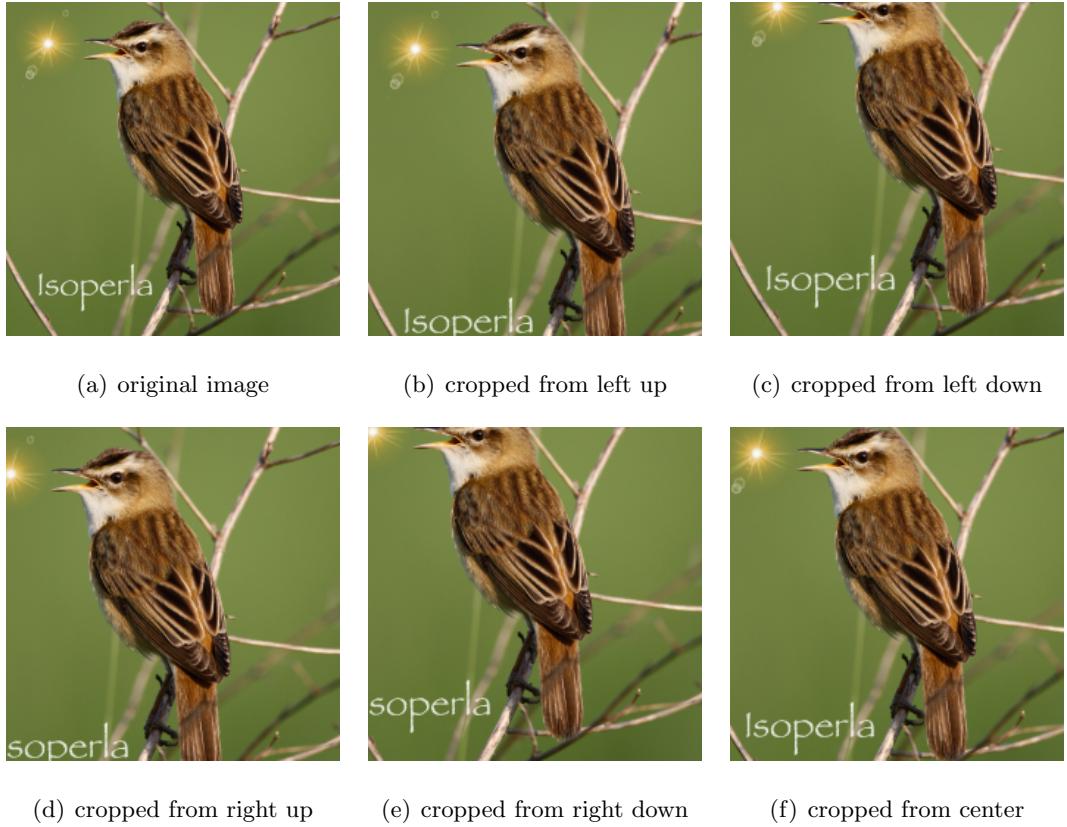


FIGURE 3.3: A set of cropped patches from original image

Assuming dimension of feature maps is  $a \times a$  and the bin size is  $n$ , each window size is  $\lceil a/n \rceil$  and stride size is  $\lfloor a/n \rfloor$  where  $\lceil \cdot \rceil$  and  $\lfloor \cdot \rfloor$  denote the ceiling and flooring operations. For the  $l$  level pyramid, there are  $l$  SPP layers. The layers will be concatenated into a fully connected layer. The bin sizes can be set to 1, 2, 3 and 6 for the SPP layers.

With the help of the SPP layer, most factors, in different scales, are taken into account.

### 3.4 Convolutional Neural Networks Architecture

Due to the significant performance of the AlexNet[13] which is a deep neural network, we train a model based on the network architecture. The architecture of the network has seven hidden adaptive layers, which are five convolutional layers and two fully connected layers. The network is very deep and has a huge number of parameters. It was implemented on two GPUs in the original experiment. The model is able to reach 62.5% accuracy rates with one prediction and 83% accuracy rates with five predictions in the competition.

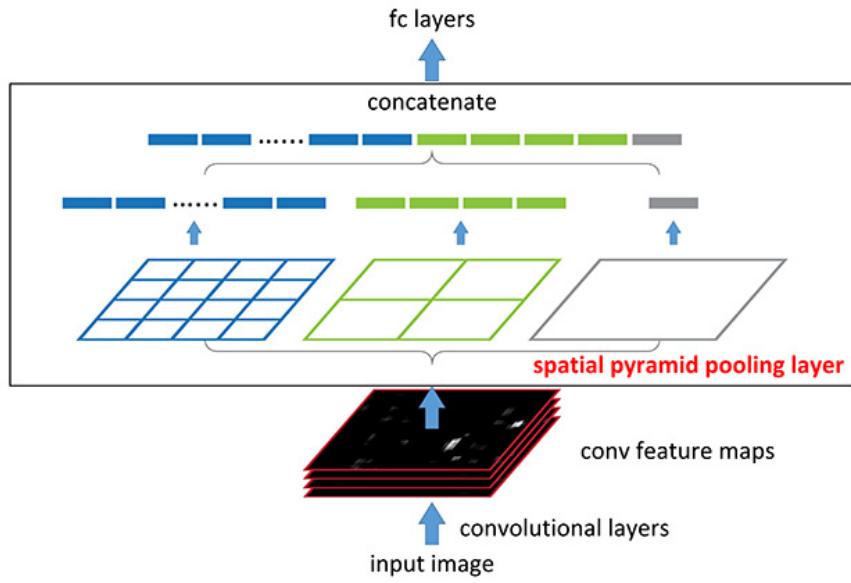


FIGURE 3.4: Diagram of the SPP layer[12]

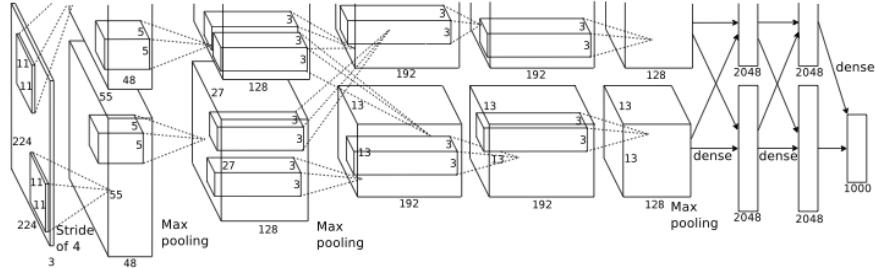


FIGURE 3.5: Architecture of the AlexNet[13]

The network uses the ReLU[31] as the activation function, which can achieve a faster learning speed than the *tanh* function does.

In the network, there are three types of layers and they play different roles in the model. The input data dimension is  $224 \times 224 \times 3$ . In the first convolutional layer, there are 96 kernels with size  $11 \times 11$ . The stride size is 4 and the outputs are 96 neurons. A following max pooling layer downsamples the spatial dimension. The second convolutional layer filters the output of the previous pooling layer with 256 kernels with size  $5 \times 5$ . The third one owns 384 kernels with size  $3 \times 3$ . The fourth convolutional layer has 384 kernels with size  $3 \times 3$  and the last convolutional layer has 256 kernels with size  $3 \times 3$ . After those convolutional layers, two fully connecting layers have 4096 neurons each. At the end of network, there is a softmax layer with 1000 outputs.

Layer Name	Layer Description
Input	$224 \times 224$ RGB image
CONV1	$11 \times 11$ conv, 96 ReLU neurons, stride 4
POOL1	$3 \times 3$ max pooling, stride 2
CONV2	$5 \times 5$ conv 256 ReLU neurons, stride 1
POOL2	$3 \times 3$ max pooling, stride 2
CONV3	$3 \times 3$ conv 384 ReLU neurons, stride 1
CONV4	$3 \times 3$ conv 384 ReLU neurons, stride 1
CONV5	$3 \times 3$ conv 256 ReLU neurons, stride 1
POOL5	$3 \times 3$ max pooling, stride 2
SPP	bin size 1,2,3,6
FC6	fully connect, ReLU 4096 neurons
FC7	fully connect, ReLU 4096 neurons
FC8	fully connect, ReLU 1000 neurons
SOFTMAX	1000 way softmax

TABLE 3.1: Architecture of the model

# Chapter 4

## Experiment

The experiment environment includes hardware, which is i7 CPU, 8G RAM and a GeForce GTX 770, and software, which is Ubuntu Linux 14.04 and Caffe[32], a deep learning software framework. We train the model in Caffe with implemented SPP layers.

### 4.1 Training Neural Networks

We trained model with the 1000-category ImageNet2012 dataset. We use the network architecture of the model [33] which achieved an excellent accuracy in the 2013 ImageNet Competition. We implemented SPP with CUDA C language and deployed it before the first fully connecting layer.

In the experiment, a subset of the ImageNet dataset, which contains 1.2 million labelled high-resolution images depicting 1000 object categories and 50,000 validation images, is used as training dataset. Most images of the dataset are multi-scale and some are grey. To feed them into the network conveniently, it is better to convert them with a constant dimensionality. The images are resized to  $256 \times 256$ . And for grey images, they are combined triple times to simulate RGB images. The model is trained on raw RGB values of pixels. The activation function is the ReLU, which ensures fast training of the neural network.

$$f(x) = \max(0, x) \tag{4.1}$$

The model 3.5 is trained with a descent optimisation method, SGD. The batch size is 128, and the momentum is 0.9. In each layer, the weights are initialised with a Gaussian distribution which has a 0 mean, and 0.01 standard deviation. The neuron bias, in the  $Conv_2$ ,  $Conv_4$ ,  $Conv_5$  and fully connecting layers, are initialized with value 1, while other bias is initialised with value 0.

The learning rates are set equally for all layers. They are initialised at 0.01 and decrease with the stepdown policy which means that they would drop by a factor of 10 after each 100,000 iteration. In total, the learning rate drops 3 times and the accuracy keeps stable after 370,000 iterations.

The training is regularised via techniques, dropout and weight decay. The dropout regularisation is implemented at the two fully connecting layers and the dropout ratio is 0.5. The neurons, which are dropped out, output zero and do not participate in the backpropagation process. Therefore, the neural network samples different architectures each time. It, significantly, decreases complex co-adaptations of neurons because a neuron does not depend on the existence of all other neurons. The weight decay,  $\epsilon$ , is set to 0.0005 which means the new weights are shrunk according to

$$w^{new} = w^{old}(1 - \epsilon) \quad (4.2)$$

after each update.

## 4.2 Fine-tuning Model

There are two challenges. First, the CNN model is trained to classify objects in images. However, the current task is to classify weather scenes. This can be solved by changing the output of the model from 1000 to 2. Second, the CNN architecture contains more than 60 million parameters which are too many for the weather classification dataset. The target dataset has only 10000 images in total. It is insufficient and the model is easy to overfit. It can be solved by training on the pre-trained CNN model. Because the learned model is close to optimism, it needs tiny adjustment only.

The fine-tune transfers weights of each layers from the previous model to the new model except for the last fully connected layer. The last layer is taken over by a new layer

which contains the same amount of neurons equally to the class number. The weights in the replaced layer are initialized with random values. One advantage of fine-tune is keeping minimum risk of overfitting. The other one is that weights reach optimal values quickly.

In the task, we want to classify sunny and cloudy images. The last layer of the network, *FC8*, is taken place by a new layer with two neurons according to two types of weather conditions. The weights of the pre-trained model are used to initialise the weights in the new neural network.

1000 images are kept out to evaluate the model. One iteration is a process of feeding a batch of images into the network. There are 10,000 iterations in the training process totally, which means that total training image number is  $128 \times 10000$ . One epoch is that the total training images are fed into the network once. Dividing the size of training images 9000, the number of epochs is 142. The initial base learning rate is 0.001 and the rate is divided by 10 every 10 epochs. Because the weights in *FC8* are randomly initialised which means they are not close to final optimization value, the learning rate of *FC8* is 10 times of the base learning rate in order to converge quickly.

### 4.3 Companion Experiments

To compare the performance of fine-tuned model, we did extra experiments with extracting feature methods. We used a pre-trained model with the AlexNet architecture and extracted features from the layer *FC7*. We trained a SVM classifier based on the features and classified the test samples.

### 4.4 Experimental Results

The results in table 4.1 show that the models, trained by the neural networks, have better performance than the extracting features method.

The fine-tuning process is very quick and the accuracy rate is over 90%. After 12 epoch, the accuracy rate exceeds 90%.

Methods	CNN+SVM	CNN+SPP+SVM	Finetune on CNN	Finetune on CNN + SPP
Accuracy	84.8%	82.1%	93.1%	93.98%

TABLE 4.1: The first test is extracting features from the pre-trained CNN model and training a SVM classifier. The second test is similar with the first except for extracting features from the CNN model with a SPP layer. The third test is fine-tuning the model with AlexNet architecture. The fourth test is fine-tuning the CNN model with a SPP layer

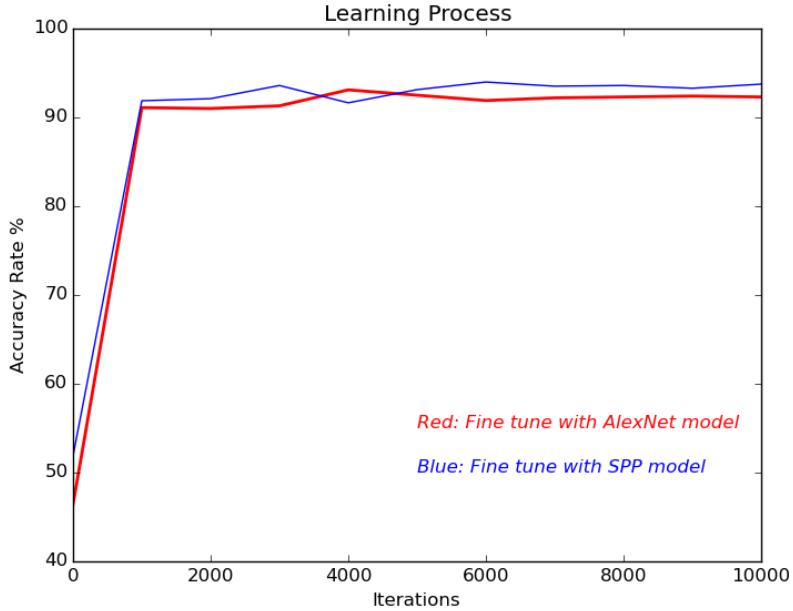


FIGURE 4.1: Learning Process

The learning process may have overfitting which definitely will damage model generalization capability. From figure 4.1, we find that there is no overfitting in the fine-tuning process. Meanwhile, the accuracy increases sharply and keeps stable.

To have a more detailed information of fine-tuning process, we investigate the training loss values. We plot the curve of first 200 iterations and loss value.

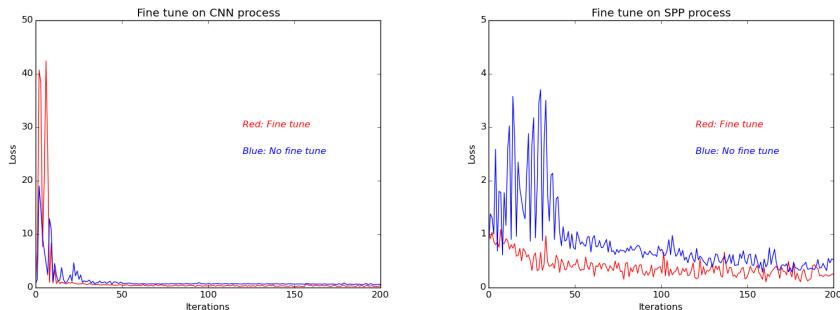


FIGURE 4.2: Finetune Process

From the figure 4.2, we can find that the fine-tuning procedure produces a smoother loss function curve and ceases at a higher accuracy rate. In the left figure, the loss value of the fine-tuning procedure is higher than the value of the no fine-tuning process at the beginning. It then shrinks sharply and keeps lower than the value of the no fine-tuning process. In the right figure, we can find that starting loss values are both less than those in the left figure. And the loss value of the fine-tuning process is less than the value in the no fine-tune process. We can conclude that generally the fine-tuning is an effective approach to reduce loss value and the fine-tuning on the model with the SPP layer can achieve better results than the original CNN model.

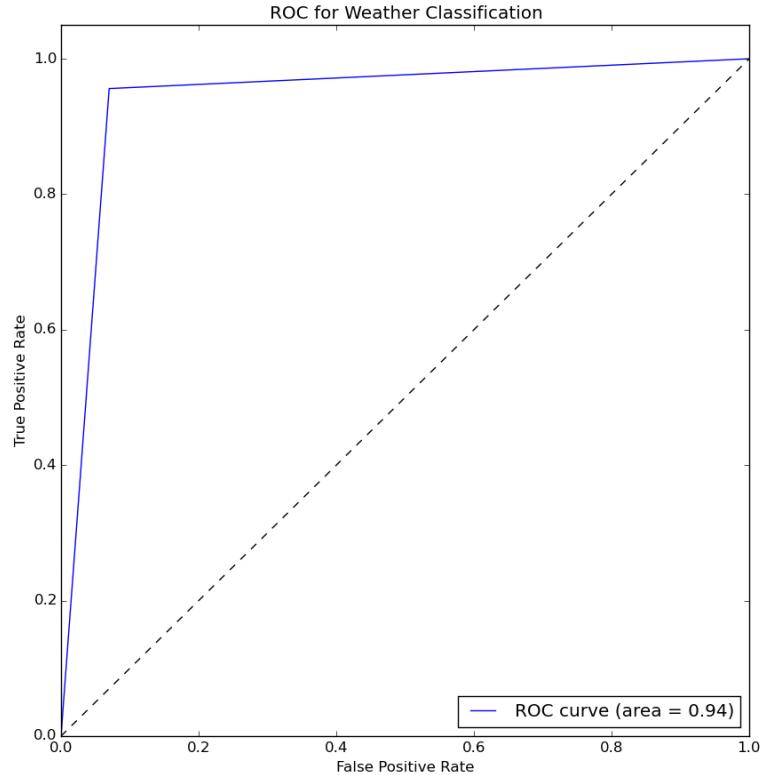


FIGURE 4.3: ROC Curve

## 4.5 Architecture Analysis

The convolutional neural networks have achieved excellent performance in object classification field, although a full understanding of the mechanism is ambiguous. In order to have some intuition about the CNN, we will do some analysis on the network.

The outputs of each layers have been treated as visual descriptors [34], and the vectors can present some information from the outputs of previous layers. The front layers encode low-level features, and the rear layers are able to capture high-level information. In other words, the image is retrieved heuristically when it passes the convolutional layers.

Fully connecting layers can be represented as a  $d$ -dimensional vector. The layer takes all outputs of the previous layer whose feature maps are multidimensional, and flats the feature maps into a  $d$ -dimensional vector. The outputs of the second fully connecting layer are feed into a classifier. Two types of classifiers can be used, which are the SVM and the softmax.

If a cloudy image is fed into the CNN, Figure 4.4 shows the feature maps of convolutional layers. Because the filter size is too many, only parts of the filters are plotted in images b-d.

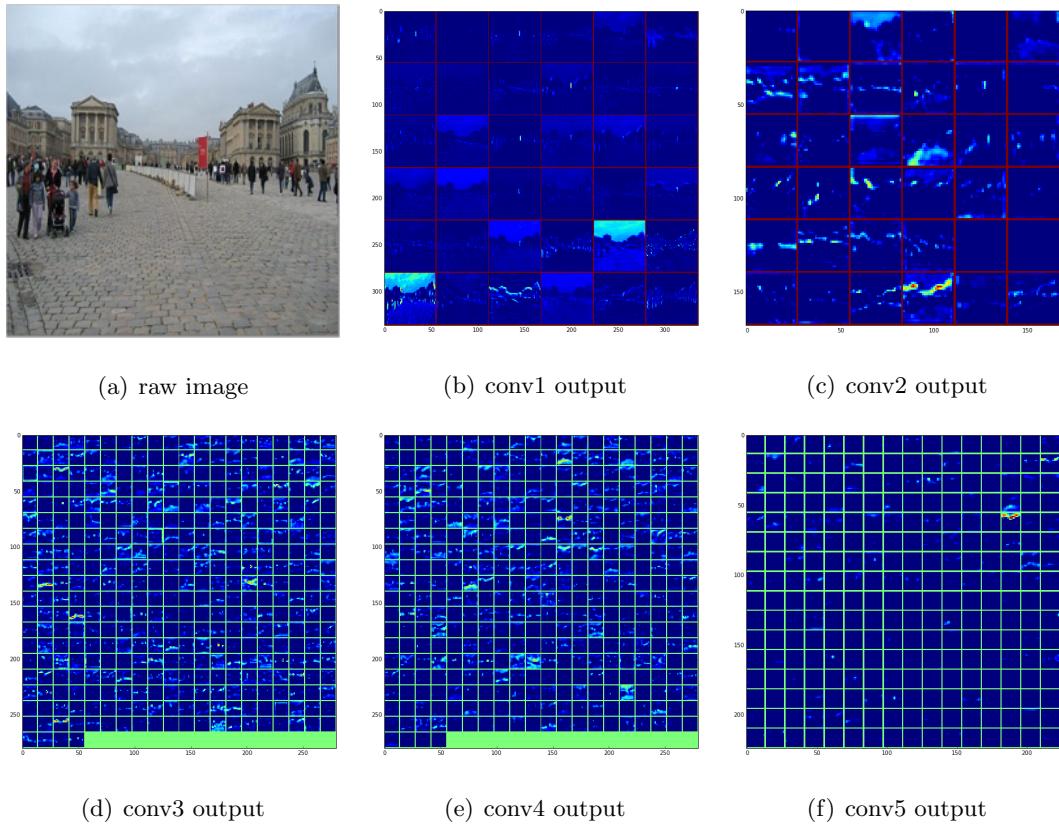


FIGURE 4.4: A cloudy image and the feature maps from convolutional layers

In figure 4.5, a sunny image is fed into CNN and the according outputs are plotted.

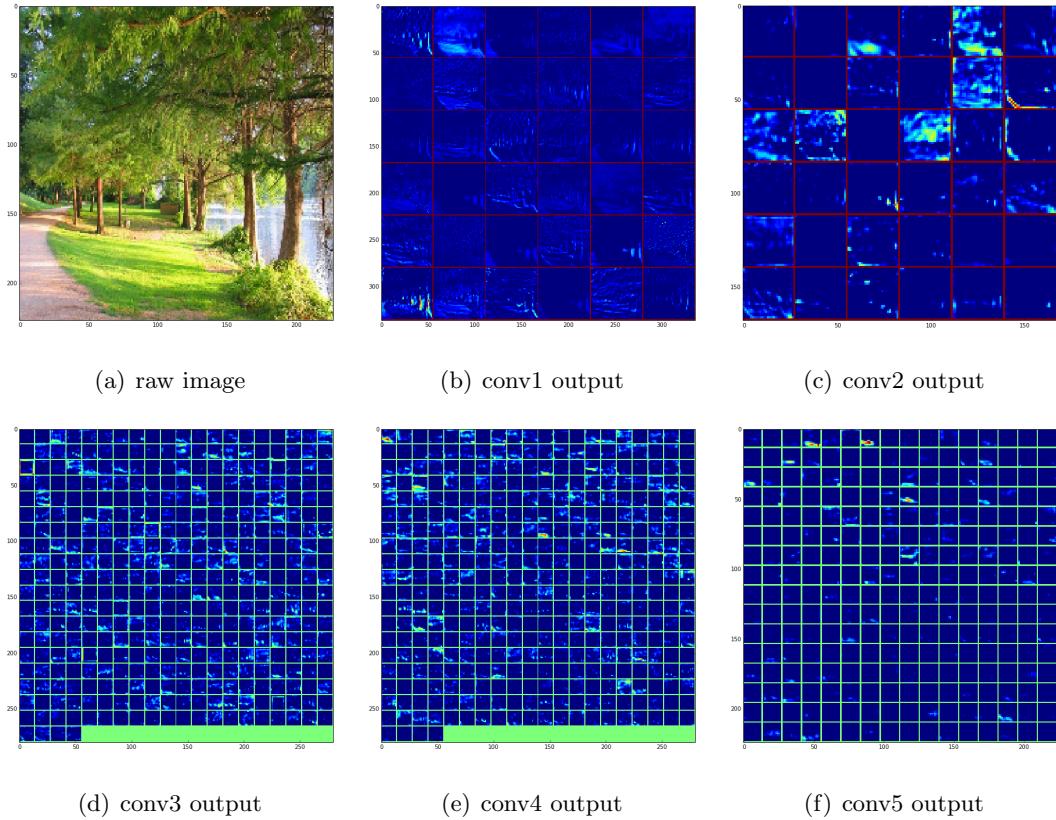


FIGURE 4.5: A sunny image and the feature maps from convolutional layers

The outputs of the *CONV1* and *CONV2* are still partly recognisable by people. The outputs of the *CONV3* and *CONV4* are quite unrecognisable from a people view. From the outputs of *CONV5* for the sunny image, the position of sun light is highlighted by several filters. However, the according filters show no signs about the cloudy image.

## 4.6 Effects of SPP

Because the outputs of the SPP cannot be represented as visual descriptors, the feature maps of the convolutional layers are compared. The outputs of the *CONV1* and *CONV2* show that more features are recognisable in the SPP model. It is probably that the SPP model has stronger representation capability than the CNN model.

In figure 4.7, the outputs of the layer *FC7* are displayed by the histogram method. Comparing histogram difference, the generated features of the SPP model is more divisible than those of the CNN model.

Image	Conv1	Conv2	Conv3	Conv4	Conv5
Image	Conv1	Conv2	Conv3	Conv4	Conv5

FIGURE 4.6: Visiualisation of feature maps from the CNN model and the SPP model. The upper images are from the CNN model and the lower images are from the fine-tuned model.

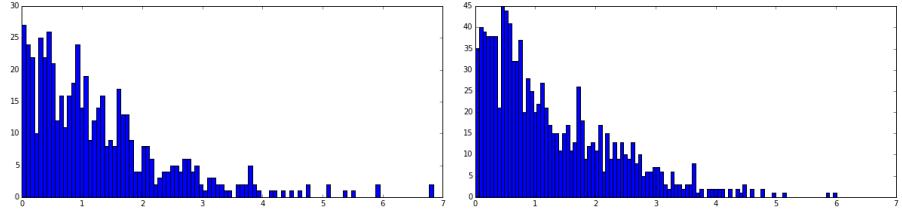


FIGURE 4.7: Histogram distribution of vectors from FC7. The left is from CNN model and the right is from the SPP model.

From the feature maps and histogram distributions, it is clear that the SPP model has strong representing capability and can generate divisible features to classifiers.

## 4.7 Error Results

The images 4.8 are misclassified. They are hard to judge sunny or cloudy, even for people.

## 4.8 Conclusion and Future Work

In this project, we present an approach to do weather classification through the convolutional neural network and transfer learning. The result shows the strong capacity of the convolutional neural network and the convenience of the transfer learning. Compared with the traditional method, which uses extracting features and training a classifier based

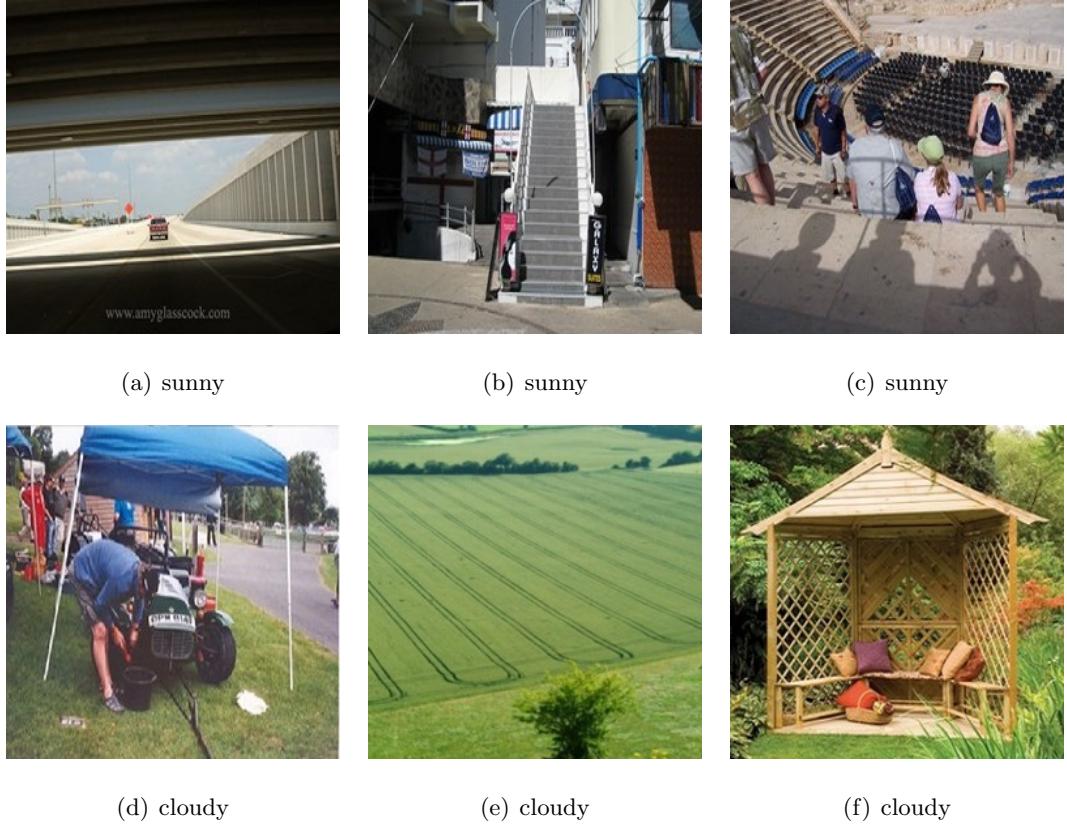


FIGURE 4.8: Misclassified images[11].

on the features, the CNN does not depend on specific feature detectors and achieve high accuracy.

In the future work, the full understanding of the CNN mechanism is a demanding work. And the work can be extended to multi-class weather classification and implemented in industry widely.

## **Part II**

# **Multilabel Learning**

# Chapter 5

## Introduction

### 5.1 Overview

In the literature on machine learning, multi-label classification is a variant of the classification problems where each sample has several labels. In general, the task of multi-label learning is to find a model that can map inputs  $\mathbf{x}$  to a set of binary vectors  $\mathbf{y}$ . It is different with multi-class classification in terms of the output label space.



FIGURE 5.1: Example Image

The difference between single-label classification and multi-label classification is the number of output labels. For figure 5.1, we can classify it as a picture of a beach in

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$Y_1$	$Y_2$	$Y_3$	$Y_4$
1	0.4	0.2	0	1	0	1	1	0
0	0.2	0.6	1	1	1	0	1	0
0	0.5	0.8	1	1	0	0	1	0
1	0.1	0.4	0	1	1	1	1	0
1	0.8	0.2	0	1	0	1	1	0
0	0.5	0.3	1	1	?	?	?	?

TABLE 5.1: Multilabel  $Y_1, \dots, Y_L \in 2^L$ 

single-label classification problem  $\in \{Yes, No\}$ . In multi-label classification literature, we can tag beach, sea, chairs, sand for the picture  $\in \{beach, sea, chairs, sand\}$ .

In general, there are two main approaches of tackling the multi-label classification problem. One method is to transfer a multi-label problem into a set of binary classification problems which can be handled by a set of binary classifiers. The other one applies algorithms to classify multi-label images directly.

Several problem transformation methods can be applied to multi-label classification. A baseline method, named the binary relevance method[35], trains one binary classifier for each label independently. Depending on the results of the classifiers, the combined model predicts all labels for a test sample. The method divides the problem into multiple binary works which has something in common with the one-vs-all method for multi-class classification. Problem transformation methods benefit from scalability and flexibility because single-label classifier can be implemented by classifiers. The SVM, Naive Bayes,  $k$  Nearest Neighbor have been used in the method[35].

There are some other transformation methods, for instance label powerset transformation. The method builds up one binary classifier for each label combination verified in the training dataset[36]. The random  $k$ -labelsets algorithm[37] utilises multi-label powerset classifier which is trained on a random sub dataset of the labels. At last, a voting scheme is used to predict test samples via an ensemble method.

## 5.2 Multi-Label Learning

Let  $X = R^d$  represent the domain of dataset and let  $Y = 1, 2, \dots, L$  be the finite set of labels. Given that we have a set of training dataset  $T = (x_1, Y_1), (x_2, Y_2), \dots, (x_l, Y_l)$  ( $x_i \in X, Y_i \subset Y$ ) which are extracted from an unknown distribution  $D$ . Our target of task

is to learn a multi-label classifier  $h : X \rightarrow 2^y$  via optimising specific evaluation metric. However, instead of learning a multi-label classifier, we will learn a function  $f$  while  $f(X) \rightarrow R^d$ . Supposing that a high performance classifier can output a closer subset for labels in  $Y_i$  than those missing or exceeding in  $Y_i$ , which means  $f(x_i, y_1) > f(x_i, y_2)$  if  $y_1 \in Y_i$  and  $y_2 \notin Y_i$ . We can transfer real valued function  $f(\cdot, \cdot)$  to a ranking function  $r(\cdot, \cdot)$  that maps the outputs of  $f(x_i, y_1)$  to any  $y \in Y$  if  $f(x_i, y_1) > f(x_i, y_2)$ . It is worth noting that the multi-label classifier  $h(\cdot)$  can be derived from the function  $f(\cdot, \cdot)$ , where  $h(x_i) = y | f(x_i, y) > t(x_i), y \in Y$ , and  $t(\cdot)$  is a threshold function.

Single-label and multi-class classification can be regarded as two degenerated variants of multi-label learning problem if each sample has only one single label. However, multi-label problem is much more difficult than traditional single-label problems because of high dimensional output space. For example, the number of label sets increases exponentially with increasing number of class labels. If there are 10 class labels for the dataset, there are  $2^{10}$  possible label sets maximum.

The challenge of huge combination of output labels is hard to overcome. One of methods is to investigate dependency among labels to reduce label space. For example, if an image labelled with *castle*, it would be highly labelled with *brick* and *mountain*. A movie which is labelled with *comedy* is unlikely related to a *documentary*. Therefore, successful exploitation of the label correlations is regarded as an effective approach to high accuracy multi-label learning process. There are three strategies, depending on the order of the label correlation, to find the relation between labels. They are first order strategy, second order strategy and high order strategy.

The first order strategy treats the label by label independently and ignores correlation between labels. It can be regarded as decomposing a multi-label learning problem into a set of binary classification problems based on each label. The method benefits from simple computation and high efficiency. However, accuracy could be suboptimal because of ignoring the correlation of labels.

The second order strategy considers pairwise relations between labels. For example, the interaction between any pair of labels, or the ranking between related and unrelated labels. The method can achieve good generalisation performance because the label correlations are investigated in some extent. In real world, there are higher order correlations than second order assumption in many applications.

The high order strategy investigates more than 2 order correlations among labels which can be the influences on each label or addressing connections among sub space of output labels. It is obvious that the high order strategy is more capable than previous two strategies on the cost of complexity and intensive computation.

# Chapter 6

## Background

Compared to general classification, multi-label classification has different evaluation metrics and learning algorithms.

### 6.1 Evaluation Metrics

In supervised learning, different metrics, like accuracy and area under the ROC curve, are used to evaluate the generalisation performance of a model. In the multi-label learning, evaluating performance is more complicated than single-label classification problems because of increasing number of labels simultaneously. Therefore, two main types of evaluation methods are implemented in multi-label learning, say example-based metrics[38] and label-metrics[37].

The two types of metrics evaluate outputs of classifiers from different perspectives. Given that  $S = (x_i, Y_i)$  is a test sample and  $h(\cdot)$  is the learned multi-label classifier. The concept of example-based metrics is achieve all class labels of each test sample, and then compute the mean value of test sets to evaluate generalisation performance. Compared with considering all class labels simultaneously, label-based metrics evaluate performance by treating each class label separately and computing macro/micro-averaged value of all class labels.

In a supervised classification problem, there are a ground truth output and a predicted output for a test sample. So the results of each test sample can be assigned to one of the four categories:

- True Positive (TP) - label is positive and prediction is also positive
- True Negative (TN) - label is negative and prediction is also negative
- False Positive (FP) - label is negative but prediction is positive
- False Negative (FN) - label is positive but prediction is negative

Here we define a set  $D$  of  $N$  examples and  $Y_i$  to be a family of ground truth label sets and  $P_i = h(x_i)$  to be a family of predicted label set. The set of all unique labels is

$$L = \bigcup_{i=0}^{N-1} L_i \quad (6.1)$$

While the definition of indicator function  $I_A$  on a set  $A$  is presented:

$$I_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise} \end{cases} \quad (6.2)$$

### 6.1.1 Example-based Metrics

**Hamming Loss** evaluates performance via counting the number of misclassification labels. The smaller value of hamming loss is, the better performance the model has.

$$\frac{1}{N \cdot |L|} \sum_{i=0}^{N-1} |L_i| + |P_i| - 2 |L_i \cap P_i| \quad (6.3)$$

**Subset Accuracy** evaluates the fraction of correctly predicted example while the predicted label set is identical to the ground truth label set. It is equivalent to traditional accuracy metric.

$$\frac{1}{N} \sum_{i=0}^{N-1} I_{\{L_i\}}(P_i) \quad (6.4)$$

**Precision** is defined:

$$\frac{1}{N} \sum_{i=0}^{N-1} \frac{|L_i \cap P_i|}{|P_i|} \quad (6.5)$$

**Recall** is defined:

$$\frac{1}{N} \sum_{i=0}^{N-1} \frac{|L_i \cap P_i|}{|L_i|} \quad (6.6)$$

**Accuracy** is defined as

$$\frac{1}{N} \sum_{i=0}^{N-1} \frac{|L_i \cap P_i|}{|L_i| + |P_i| - |L_i \cap P_i|} \quad (6.7)$$

**F1 Measure** is an integrated version combined by harmonic mean of **Precision** and **Recall**.

$$\frac{1}{N} \sum_{i=0}^{N-1} 2 \frac{|P_i \cap L_i|}{|P_i| \cdot |L_i|} \quad (6.8)$$

### 6.1.2 Label-based Metrics

**Macro Precision** (precision averaged across all labels) is defined as

$$PPV(\ell) = \frac{TP}{TP + FP} = \frac{\sum_{i=0}^{N-1} I_{P_i}(\ell) \cdot I_{L_i}(\ell)}{\sum_{i=0}^{N-1} I_{P_i}(\ell)} \quad (6.9)$$

**Macro Recall** (recall averaged across all labels) is defined as

$$TPR(\ell) = \frac{TP}{P} = \frac{\sum_{i=0}^{N-1} I_{P_i}(\ell) \cdot I_{L_i}(\ell)}{\sum_{i=0}^{N-1} I_{L_i}(\ell)} \quad (6.10)$$

**F1 Measure by label** is the harmonic mean between **Precision** and **Recall**.

$$F1(\ell) = 2 \cdot \left( \frac{PPV(\ell) \cdot TPR(\ell)}{PPV(\ell) + TPR(\ell)} \right) \quad (6.11)$$

**Micro Precision** (precision averaged over all example/label pairs) is defined as

$$\frac{TP}{TP + FP} = \frac{\sum_{i=0}^{N-1} |P_i \cap L_i|}{\sum_{i=0}^{N-1} |P_i \cap L_i| + \sum_{i=0}^{N-1} |P_i - L_i|} \quad (6.12)$$

**Micro Recall** (recall averaged over all the example/label pairs) is defined as

$$\frac{TP}{TP + FN} = \frac{\sum_{i=0}^{N-1} |P_i \cap L_i|}{\sum_{i=0}^{N-1} |P_i \cap L_i| + \sum_{i=0}^{N-1} |L_i - P_i|} \quad (6.13)$$

**Micro F1 Measure by label** is the harmonic mean between **Micro Precision** and **Micro Recall**.

$$2 \cdot \frac{TP}{2 \cdot TP + FP + FN} = 2 \cdot \frac{\sum_{i=0}^{N-1} |P_i \cap L_i|}{2 \cdot \sum_{i=0}^{N-1} |P_i \cap L_i| + \sum_{i=0}^{N-1} |L_i - P_i| + \sum_{i=0}^{N-1} |P_i - L_i|} \quad (6.14)$$

For the label-based metrics, the larger the metrics value means the higher generalisation performance.

With the previous metrics, there are diverse methods to evaluate model generalization performance. In most multi-label classification, learning algorithms optimise one of the metrics. To make evaluation fair and precise, learning algorithms should be tested on different metrics to evaluate performance.

Most metrics are non-convex and discrete, and most algorithms turn to optimise alternative multi-label metrics. Recently, some researcher study the consistency of multi-label learning[39].

## 6.2 Learning Algorithms

Algorithms play a key role in the literature on machine learning, and there is no exception in multi-label learning. The capability of representation is important to evaluate the performance of an algorithm. Moreover, some relating criterions can be used to measure performance. First, the broad spectrum should be considered. An algorithm should cover a range of algorithmic design strategies with unique characteristics. Second, it is reasonable to evaluate the impact which the algorithm poses on the multi-label learning settings. Last, computational complexity is a critical factor to evaluate if an algorithm is practical.

### 6.2.1 Problem Transformation Methods

#### 6.2.1.1 Binary Relevance (BR)

The Binary Relevance method is an elementary algorithm which decomposes a multi-label learning problem into a set of independent binary classification problems, and

each problem harmonises one label in the set  $L = y_1, y_2, \dots, y_q$ . The approach initially transforms the multi-label dataset into  $q$  binary datasets  $D_{y_i}(i = 1, 2, \dots, q)$ , where each  $D_{Y_i}$  includes all samples of the multi-label dataset and has a single binary label to instruct if the dataset has or has no attribute to the relevant label. For example, if a sample has a positive value means that the sample owns the correlation label, otherwise, it does not own it. After transforming the dataset, a set of  $q$  binary classifiers, say  $H_i(E)(i = 1, 2, \dots, q)$ , have been built up using the respective training dataset  $D_{y_i}$ .

$$H = \{C_{y_i}(x, y_i) \rightarrow y'_i \in \{0, 1\} | y \in L, i = 1, 2, \dots, q\} \quad (6.15)$$

To classify a test sample, the BR method outputs the collection of labels which are predicted with positive value by the independent binary classifiers.

The BR method combines computational efficiency and simple implementation. With a constant number of samples, the algorithm scales with size  $q$  of label set  $L$ . Supposing that each classifier has complexity  $f(|X|, |D|)$ , The method has complexity  $O(q \times f(|X|, |D|))$ . With a limit number of  $q$ , the method learns a model quickly and has an reasonable performance.

One of the disadvantages is the limitation to label relationship information. Unless all labels are independent, the method losses the correlations among labels.

### 6.2.1.2 Classifier Chains (CC)

To achieve a better performance, the CC method has been introduced in multi-label classification[35]. The method transforms multi-label learning problems into a chain of binary classifiers based on label dependence.

For a set of  $q$  labels  $L$ , the CC method learns  $q$  classifiers as BR does. In addition, it links all classifiers in feature space. Given that there are a set of samples  $D(x, y_i)(i = 1, 2, \dots, q)$  where  $y_i$  is a subset of labels  $L$  and  $x$  is the domain of dataset. The original dataset is transformed into a set of  $q$  datasets in which the  $j$ th example is based on the previous dataset.

$$H = \{C_{y_i}(x, y_1, y_2, \dots, y_{i-1}) \rightarrow y'_i \in \{0, 1\} | y \in L, i = 1, 2, \dots, q\} \quad (6.16)$$

Thus it forms a chain of binary classifiers  $C_1, C_2, \dots, C_q$ . Each classifier  $C_i$  learns and predicts the binary association of label  $y_i$  based on the dataset  $x$  and all prior binary relevance predictions  $y_j, j = 1, 2, \dots, i-1$ . The learning process starts from  $C_1$  and follows the chain sequentially. Therefore,  $C_1$  determines  $p(y_1|x)$ , and sequential classifiers, say  $C_2, \dots, C_q$ , determine  $p(y_i|x_i, y_1, \dots, y_{i-1})$ .

The CC method propagates label information through classifiers, and latter classifiers take account of previous predictions. This method can retrieve lost information among labels partly. At the same time, the method keeps advantages of BR, say computation efficiency and memory efficiency. The computational complexity of the method is close to the BR in terms of the number of labels and complexity of elemental classifiers  $C_i$ . As previous statement, the computational complexity of the BR is  $O(q \times f(|X|, |D|))$  where  $f(|X|, |D|)$  represents the elemental classifier. With extra computation introduced by previous labels, the computational complexity of the CC is  $O(q \times f(|X| + q, |D|))$ , while the computation will be unaffordable in case of  $q \gg |X|$ .

## 6.2.2 Algorithm Adaptation Methods

### 6.2.2.1 Multi-label k-Nearest Neighbour (ML-kNN)

The algorithm adapts k-nearest Neighbour techniques to propose multi-label data and utilise the MAP to make prediction through reasoning embodied labelling information among neighbours [40].

Given dataset  $X$  and its label set  $Y$ ,  $y$  represents output vector for  $x$  where  $i$ -th element  $y(i)$  is positive if  $i \in Y$ , otherwise it is negative. Suppose that  $N(x)$  is the set of neighbours of  $x$  in the training dataset, we can compute a membership counting vector which represents the number of neighbours of  $x$  owning the  $l$ -th label.

$$C_x(i) = \sum_{a \in N(x)} y_a(i), i \in Y \quad (6.17)$$

To test a new sample  $t$ , the algorithm sorts out its category in the training dataset by kNN  $N(t)$ . Let  $H_1^l$  denotes that  $t$  has the label  $l$  and  $H_0^l$  denotes that  $t$  do not have

the label  $l$ . If  $j$  samples have the label  $l$ , say  $E_j^l (j \in 0, 1, \dots, k)$ , the category vector  $y_t$  is determined by the MAP principle:

$$y_t(l) = \arg \max_{b \in \{0,1\}} P(H_b^l | E_{C_t(i)}^l) l \in Y \quad (6.18)$$

With the Bayesian rule, 6.18 can be represented as

$$y_t(l) = \arg \max_{b \in \{0,1\}} P(H_b^l) P(E_{C_t(i)}^l | H_b^l) l \in Y \quad (6.19)$$

As stated in equation 6.19, the output of a test sample  $t$  is  $y_t(l)$ . It can be computed through prior probability  $P(H_b^l) l \in Y, b \in \{0, 1\}$  and the posterior probability  $P(E_j^l | H_b^l) (j \in \{0, 1, \dots, k\})$ .

### 6.2.2.2 Collective Multi-label Classifier (CML)

Supposed that labels are highly interdependent in some scenarios, the CML explores multi-label conditional random field (CRF) classification models which learn parameters for each pair of labels directly[38].

In dataset  $(x, Y)$ , each sample has a corresponding random variables representation  $(x, y)$ , in which  $y = (y_1, y_2, \dots, y_q) y_i \in \{-1, 1\}$  is a binary label vector. If the sample contains  $j$ -th label, the  $j$ -th element of  $y$  is 1, otherwise -1. Then the aim is to learn the joint probability distribution  $p(x, y)$ .

The entropy of  $(x, y)$  is represented as  $H_p(x, y)$  and gives the distribution  $p(\cdot, \cdot)$  of  $(x, y)$ . The principle of maximum entropy can be achieved by maximising  $H_p(x, y)$ . The fact is expressed with constrains on the expectation of function  $f(x, y)$ . The expected value can be estimated from the training dataset

$$F_k = \frac{1}{m} \sum_{(x,y) \in D} f_k(x, y) \quad (6.20)$$

According to the normalization constraint on  $p(\cdot, \cdot)$ , the optimal solution can be represented:

$$p(y|x) = \frac{1}{Z(x)} \exp\left(\sum_{k \in K} \lambda_k \cdot f_k(x, y)\right) \quad (6.21)$$

Yielding to Gaussian distribution, parameters can be found in a convex log-posterior probability function. For a test sample  $x$ , the predicted label set follows to:

$$y_x = \arg \max_y p(y|x) \quad (6.22)$$

It is notable that the exact inference is only suitable for small label space via  $\arg \max$ , because it needs to reduce the search space of  $\arg \max$  significantly via the pruning strategy. The CML is a second-order approach.

# Chapter 7

## Methodology

### 7.1 Artificial Dataset

The dataset, named a colour detector, is generated artificially. Each image has three labels to represent colours, *red*, *green* and *blue*. If an image is a red hue only, it has a label for *red* while the label values of *green* and *blue* are negative. The representation of colour combination follows colour wheels. If an image has a hue close to *purple*, it has positive labels for *red* and *blue*, and negative label for *green*, and so on.

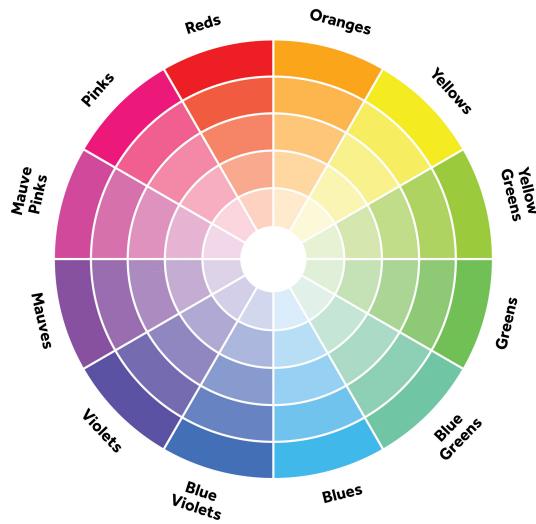


FIGURE 7.1: Colour Wheel Diagram[14]

For each sample  $x_i$ , it owns 3 labels  $y_0, y_1, y_2, y_j \in \{-1, 1\}$  which represent *red*, *green* and *blue* respectively.

### 7.1.1 Generating Images

The RGB and HSV coordinate systems represent a geometric shape in colour space. The distance among colours contains little meaning. However, corresponding distance makes some intuitive sense and enable conversion possible between RGB and HSV coordinates.

The dataset consists of images with size  $16 \times 16$ , therefore each image has 256 pixels. To generate an image, I generate a random floating point number  $h$  in the range  $[0.0, 1.0)$  and use it as value for a hue via formulation

$$H = h + r * 0.4 - 0.2(r \in [0.0, 1.0)) \quad (7.1)$$

where  $r$  is another random floating point number. 2 random floating point numbers are generated for Saturation(S) and Value(V) values. The HSV values are converted to the RGB values and timed 255 for each pixel. Following that, an image with 256 pixels is generated. Because a RGB image is converted to a HSV image, the label values are based on the previous value  $h$ .

The dataset contains 1000 images in total. 960 images are training samples and 40 are test samples.

## 7.2 Artificial Neural Networks (ANNs)

The Artificial Neural Network is an approach to learning a nonlinear function which can be used to map a sample to multi labels. The neurons in the first layer take a raw image while the neurons in the last layer produce outputs. Between the first and last layer, the middle layers are called hidden layers because they do not connect to external world directly. A 3 layer neural network can represent any bounded degree polynomial under certain conditions[41]. The weights of neural network are learned by algorithms deployed over training dataset. One of successful learning algorithms is the Backpropagation algorithm which updates weights by propagating errors caused by comparing prediction for each sample with actual target values.

Two factors will be modified to adapt a neural network to do multi-label classification. One is to design a new error function which fits the characteristics of multi-label samples

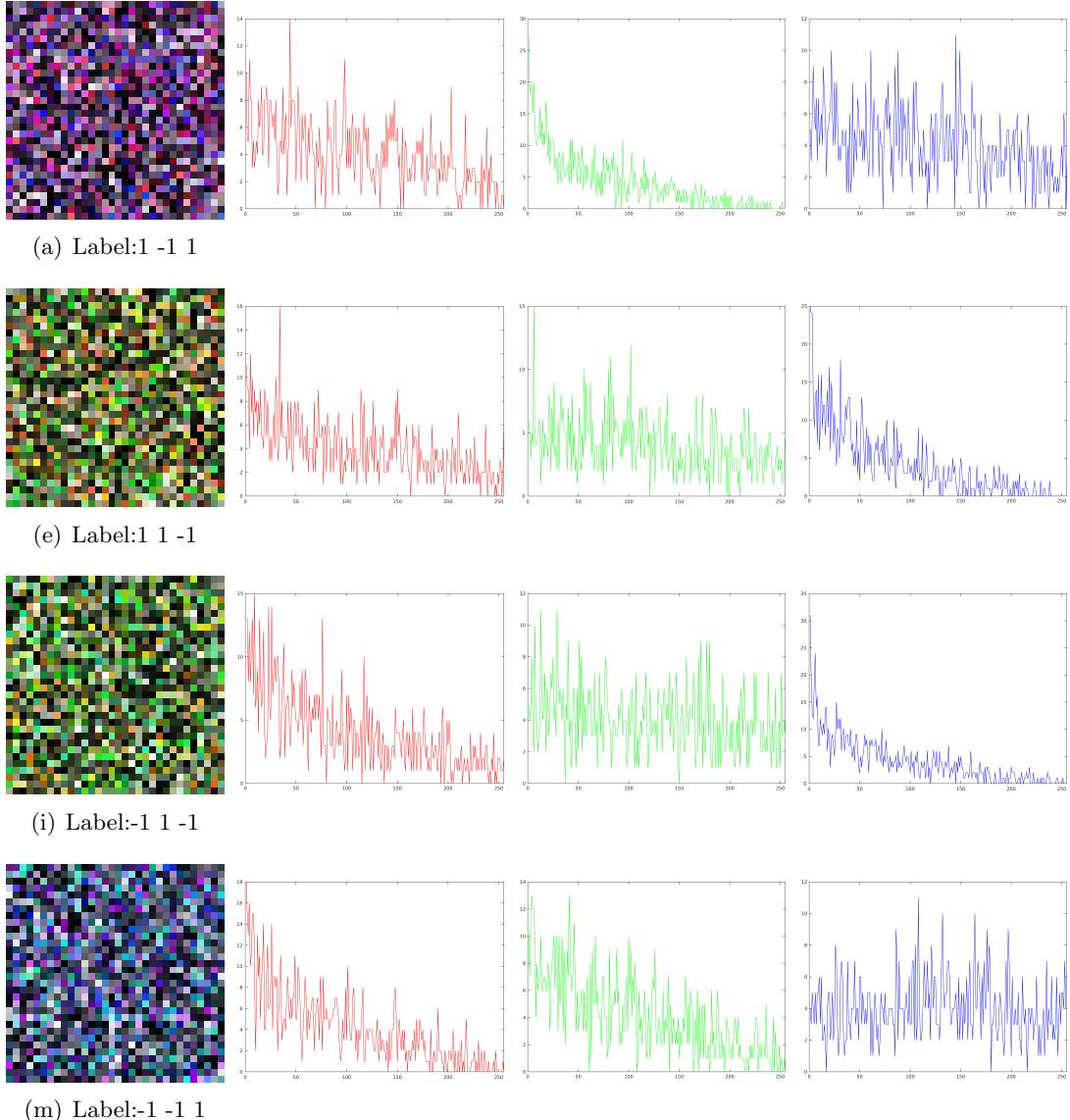


FIGURE 7.2: Multilabel samples and the RGB colour histograms. Three labels mean red, green and blue sequentially.

instead of single-label ones. The other one is to find a moderate metric according to the new designed error function.

### 7.2.1 Network Architecture

Define  $\mathcal{X} = \mathbb{R}^d$  as the sample space and  $\mathcal{Y} = 1, 2, \dots, Q$  as the set of output labels. The training dataset is composed of  $m$  multi-label samples, such as  $(x_1, Y_1), (x_2, Y_2), \dots, (x_m, Y_m)$ , while each sample  $x_i \in \mathcal{X}$  is represented as a  $d$ -dimensional feature vector and a set of  $q$  labels associate with the feature vector. A neural network, showed in figure 7.3, can be built up to learn a model .

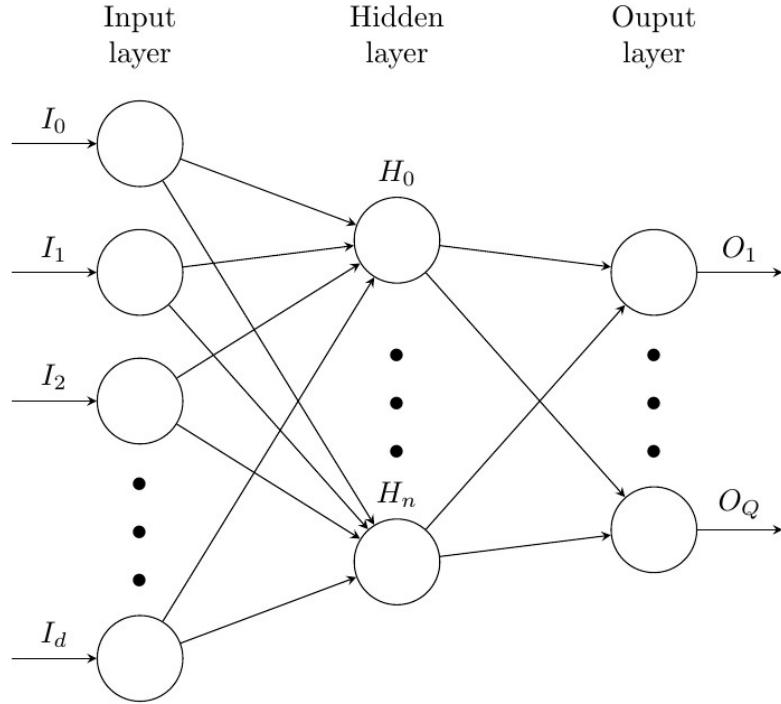


FIGURE 7.3: Network Topology For Multi-label Classification

The network has  $d$  input neurons which correspond to a  $d$ -dimensional feature vector, while the last  $Q$  neurons represent a combination of output labels. There is a hidden layer in figure 7.3 which owns  $n$  hidden neurons. The input layer is fully connecting with the hidden layer and the same connecting method is deployed between the hidden layer and the output layer. So there are  $d \times n$  weights ( $W_{ih}, 1 \leq i \leq d, 1 \leq h \leq n$ ) between the first two layers, and  $n \times q$  weights ( $W_{ho}, 1 \leq h \leq n, 1 \leq o \leq q$ ) between the latter two layers. The bias parameters are represented as  $I_0$  and  $H_0$ .

Because the task of multi-label learning is to predict labels of test samples, it needs to evaluate the global error of the model as:

$$E = \sum_{i=1}^m E_i \quad (7.2)$$

$E_i$  is the error on the sample  $x_i$  which can be defined as:

$$E_i = \sum_{j=1}^Q (c_j^i - d_j^i)^2 \quad (7.3)$$

where  $c_j^i = c_j(x_i)$  is the predicted  $j$ -th label by the model on sample  $x_i$ , and  $d_j^i$  is the actual  $j$ -th label of sample  $x_i$ . The actual label has value of either  $+1(j \in \mathcal{Y}_j)$  or

$-1(j \notin \mathcal{Y}_j)$ .

Various learning algorithms can be used to learn a model based on the training dataset. The backpropagation algorithm is used to learn from errors. However, the algorithm could be improper in multi-label learning because the error function 7.3 neglects the correlations among labels of a sample. In original BP algorithm, the error function 7.3 limits on individual label discrimination, whether a specific label  $j \in \mathcal{Y}$  belongs to the sample  $x_i$  or not. It should take into consideration that labels in  $Y_i$  are more important than those outside of  $Y_i$ . A new global error function is defined as:

$$E = \sum_{i=1}^m E_i = \sum_{i=1}^m \frac{1}{|Y_i||\hat{Y}_i|} \sum_{(k,l) \in Y_i \times \hat{Y}_i} \exp(-(c_k^i - c_l^i)) \quad (7.4)$$

In the error function 7.4, the  $i$ -th errors on the  $i$ -th sample  $(x_i, Y_i)$  are accumulated.  $\hat{Y}_i$  is complementary set of  $Y_i$  in  $\mathcal{Y}$  and  $|\cdot|$  computes the cardinality of a set. Specifically, the item,  $c_k^i - c_l^i$ , represents the difference between the output labels of the model on the label which belongs to sample  $x_i$  and the label which does not belong to the same sample. The error function 7.4 shows that bigger difference leads to better performance. Additionally, the negation of the difference is put into the exponential function to sharply penalize the  $i$ -th term if  $c_k^i$  is much smaller than  $c_l^i$ . The sum of  $i$ -th error term accumulates difference between outputs of any pair of labels which are the one belonging to the sample and the other one not belonging to it. The sum is normalized by the numbers of all pairs,  $|Y_i||\hat{Y}_i|$ . And then, the correlations between pair labels are computed. In other words, labels in  $Y_i$  should get larger output value than labels in  $\hat{Y}_i$ .

As previous statement, the error function 7.4 calculates the difference between output labels. The task of learning is minimising the error function 7.4 via enlarging output values of labels belonging to the training samples and diminishing the output values of the labels not belonging to it. If the training dataset can cover the distribution of the whole sample space, the model can learn it through minimising error function by feeding training samples.

### 7.2.2 Error Function

The basic goal of regression problems is to figure out the conditional distribution of the output labels with the input samples. It is common to use a sum-of-squares error

function.

The basic goal of classification problems is to figure out the posterior probabilities of class types with the input samples. Except for the sum-of-squares error function, there are some more approximate error functions which can be considered.

The central goal in training a neural network is to model the hidden generator of the samples instead of memorising the training samples. Therefore, the best prediction of an input sample can be found if the network can present a new value for the sample. Because the general and complete characterisation of the dataset is the probability density  $p(x, t)$ , it is available to decompose a joint probability density into the product of the conditional density of the labels, the input data and the density of the data,

$$p(x, t) = p(t|x)p(x) \quad (7.5)$$

where  $p(t|x)$  represents the probability density of  $t$  if  $x$  is a distinct label and  $p(x)$  represents the density of  $x$

$$p(x) = \int p(t, x)dt \quad (7.6)$$

Most error functions can be obtained from the idea of maximum likelihood. For training dataset,

$$L = \prod_n p(t^n|x^n)p(x^n) \quad (7.7)$$

where each sample is picked out randomly from the same distribution and their probabilities can be multiplied. The error function can be represented by minimizing the negative logarithm of the likelihood since the negative logarithm is a monotonic function.

$$E = -\ln L = -\sum_n \ln p(t^n|x^n) - \sum_n \ln p(x^n) \quad (7.8)$$

where  $E$  is notated as an error function. The task of learning is to model the conditional probability density  $p(t|x)$ . And the second term is independent with the parameters in neural networks. We can simplify equation 7.8 to

$$E = -\ln L = -\sum_n \ln p(t^n|x^n) + C \quad (7.9)$$

It is worth noting that error functions are dependent on different assumptions of the forms of the conditional distribution  $p(t|x)$ . In this classification task,  $t$  represents labels which act as class members or the prediction of the probabilities of class members.

### 7.2.3 Cross Entropy

The Mean Square Error is a common risk metric comparable to the predicted value of the squared error loss. It is easy to implement and its value is non-negative. However, it has the disadvantage of heavily weighting outliers[42].

Given that there are two discrete distributions  $p(x)$  and  $q(x)$  over the same variable  $x$ . The relative entropy, relating to the cross entropy, is a measurement of the distance between samples:

$$D_{pq}(p(x), q(x)) = \sum_x q(x) \ln \frac{q(x)}{p(x)} \quad (7.10)$$

where the relative entropy is not a true metric because equation 7.10 could not be symmetric in the interchange  $p \leftrightarrow q$  and probably not satisfy the triangle inequality.

Cross entropy loss, also named logistic regression loss, is an alternative measurement of a probability distribution. The measure is commonly used in neural networks. It can be used to evaluate the posterior probabilities of class membership.

Given that training a neuron, which has several input data,  $x_1, x_2, \dots$ , with weights  $w_1, w_2, \dots$ , and a bias,  $b$ , the outputs, for example two classes, can be represented as the weighted sum of the input data.

$$a = f(x) = \sum_i w_i x_i + b \quad (7.11)$$

Then the cross entropy loss function is

$$C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)] \quad (7.12)$$

where  $n$  is the number of training samples,  $a$  is the prediction result and  $y$  is actual result.

The cross entropy can act as a loss function because of two properties. First,  $C$  is non-negative, because the items of logarithms are in the range  $[0, 1]$ . Second, if the prediction

output is close to the actual value for all training samples, the loss value,  $C$ , will be close to 0. Given that  $y = 0$  and the prediction value  $a \approx 0$ , the first item in equation 7.12 is 0 while the second one is  $-\ln(1 - a) \approx 0$ . The similar situation occurs for conditions,  $y = 1$  and  $a \approx 1$ . Therefore, the value of loss function will be small, if the prediction value is close to the actual value.

In total, the cross entropy tends to 0, when the neuron acts better at predicting the output  $y$  for total training inputs  $x$ . The two properties are basic to a loss function. Although the quadratic loss function satisfy the properties, the cross entropy has the advantage on avoiding the issue of learning curve slowing down. To illustrate the advantage, the partial derivative of the cross entropy loss function can be done with respect to the weights. Applying the chain rule twice on equation 7.12

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_x \left( \frac{y}{f(x)} - \frac{(1-y)}{1-f(x)} \right) \frac{\partial f}{\partial w_j} \quad (7.13)$$

$$= -\frac{1}{n} \sum_x \left( \frac{y}{f(x)} - \frac{(1-y)}{1-f(x)} \right) f'(x)x_j. \quad (7.14)$$

where the second one can be represented as

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x \frac{f'(x)x_j}{f(x)(1-f(x))} (f(x) - y) \quad (7.15)$$

Because the defination of the sigmoid function is  $f(x) = 1/(1+e^{-x})$  and the derivative of sigmoid function is  $f'(x) = f(x)(1-f(x))$ . Then the items  $f'(x)$  and  $f(x) = 1/(1+e^{-x})$  cancel in the equation and it becomes

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j(f(x) - y) \quad (7.16)$$

The equation 7.16 shows that the rate of learning weights is controlled by  $f(x) - y$ . The larger the error is, the faster the neuron learns. In particular, the property avoids the learning slowdown because the derivative item  $f'(x)$  gets canceled out in the quadratic cost.

In the similar way, the bias can be computed by the partial derivative.

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum_x (f(x) - y) \quad (7.17)$$

### 7.2.4 Training and Testing

Following the training process, gradient descent is used to minimise the global error function with backpropagation.

To train a sample  $(x_i, Y_i)$ , in which  $x_i$  is the input data and  $Y_i$  is the associated labels, the predicted output labels computed by the neural network for is

$$c_k = f(netc_k + \theta_k) \quad (7.18)$$

where  $\theta_k$  is the bias neurons of  $k$  layer,  $f()$  is the activation function for the output neurons which is the  $tanh$  function 2.12 in this project.  $netc_k$  is the input data of the layer:

$$netc_k = \sum_{s=1}^M b_s w_{sk} \quad (7.19)$$

where  $w_{sk}$  is the weights for the layer  $s$  and layer  $k$ ,  $b_s$  is the input vector, and  $M$  is the number of neurons in the hidden layer.

As the  $tanh$  function is differentiable, the general error of the  $k$ -th output neuron can be defined:

$$d_k = -\frac{\partial E}{\partial netc_k} \quad (7.20)$$

combining with equation 7.18, we can get

$$d_k = -\frac{\partial E_i}{\partial c_j} \frac{\partial c_j}{\partial netc_k} = -\frac{\partial E_i}{\partial c_j} f'(netc_k + \theta_k) \quad (7.21)$$

With considering global error function 7.4

$$\frac{\partial E_i}{\partial c_j} = \begin{cases} -\frac{1}{|Y_i||\hat{Y}_i|} \sum_{l \in \hat{Y}_i} \exp(-(c_j - c_l)) & \text{if } j \in Y_i \\ \frac{1}{|Y_i||\hat{Y}_i|} \sum_{k \in Y_i} \exp(-(c_k - c_j)) & \text{if } j \in \hat{Y}_i \end{cases} \quad (7.22)$$

with derivation of the  $tanh$  function, and substituting it with equations 7.21 and 7.22 we can get

$$d_k = \begin{cases} \left( -\frac{1}{|Y_i||\hat{Y}_i|} \sum_{l \in \hat{Y}_i} \exp(-(c_j - c_l)) \right) (1 + c_j)(1 - c_j) & \text{if } j \in Y_i \\ \left( \frac{1}{|Y_i||\hat{Y}_i|} \sum_{k \in Y_i} \exp(-(c_k - c_j)) \right) (1 + c_j)(1 - c_j) & \text{if } j \in \hat{Y}_i \end{cases} \quad (7.23)$$

According to the previous method, we can define the general error of the  $s$ -th hidden neuron:

$$e_s = -\frac{\partial E_i}{\partial netb_s} \quad (7.24)$$

with  $b_s = f(netb_s + \lambda_s)$  and chain rule,

$$e_s = -\frac{\partial E_i}{\partial b_s} \frac{\partial b_s}{\partial netb_s} = -\left(\sum_{j=1}^Q \frac{\partial E_i}{\partial netc_j} \frac{\partial netc_j}{\partial b_s}\right) f'(netb_s + \lambda_s) \quad (7.25)$$

For  $d_j = -\frac{\partial E_i}{\partial netc_j}$  and  $netc_j = \sum_{s=1}^M b_s w_{sj}$ , then

$$e_s = \left(\sum_{j=1}^Q d_j \times \frac{\partial(\sum_{s=1}^M b_s w_{sj})}{\partial b_s}\right) f'(netb_s + \lambda_s) = \left(\sum_{j=1}^Q d_j w_{sj}\right) f'(netb_s + \lambda_s) \quad (7.26)$$

As  $f()$  is the  $\tanh$  function, we get

$$e_s = \left(\sum_{j=1}^Q d_j w_{sj}\right)(1 + b_s)(1 - b_s) \quad (7.27)$$

The SGD is used to approximate function.

$$\Delta w_{sj} = -\alpha \frac{\partial E_i}{\partial w_{sj}} = \alpha \frac{\partial E_i}{\partial netc_j} \frac{\partial netc_j}{\partial w_{sj}} = \alpha d_j b_s \quad (7.28)$$

$$\Delta v_{hs} = -\alpha \frac{\partial E_i}{\partial v_{hs}} = \alpha \frac{\partial E_i}{\partial netb_s} \frac{\partial netb_s}{\partial v_{hs}} = \alpha e_s a_h \quad (7.29)$$

while the bias is updated according to

$$\Delta \theta_j = \alpha d_j \quad \Delta \lambda_s = \alpha e_s \quad (7.30)$$

In previous equations,  $\alpha$  is the learning rate with its value in the range of [0.0 1.0].

In the training process, a learning algorithm has been set up with backpropagation. Moreover, training samples are fed into the neural network. After all training samples  $(x, Y)$  fed into network, weights and bias are updated through equations 7.28 and 7.30. The training samples are fed into the network iteratively while global error value decreases. Finally, the error value converges to a minimum value.

In the testing process, the network predicts a sample which has a set of actual labels  $c_j (j = 1, 2, \dots, Q)$  by ranking the labels. Because the output value of each label is in

range  $[-1.0, 1.0]$ , a threshold function  $t(x)$  is used to determine associated label set for the sample  $x$ . A large margin ranking system[43] is adopted to generalise the sets.  $t(x)$  is a linear function,  $t(x) = w^T \cdot c(x) + b$ , where  $c(x) = (c_1(x), c_2(x), \dots, c_Q(x))$  is a  $Q$ -dimensional vector which represents  $j$  output labels of the sample. For every training sample  $(x_i, Y_i)(1 \leq i \leq m)$ , the relation between  $c(x_i)$  and target value  $t(x_i)$  is:

$$t(x_i) = \arg \max_t (|\{k|k \in Y_i, c_k^i \leq t\}| + |\{l|l \in \hat{Y}_i, c_l^i \geq t\}|) \quad (7.31)$$

If there are several minimum values and optimal values are in a division, the middle value of the division is chosen. The task of learning the parameters of threshold function is to solve the matrix equation  $\Phi \cdot w' = t$ . The matrix  $\Phi$  has dimensions  $m \times (Q + 1)$  in which  $i$ -th vector is  $(c_1^i, c_2^i, \dots, c_Q^i, 1)$ , and  $w'$  is  $(Q + 1)$  dimensional vector  $(w, b)$  and  $t$  is the  $m$  dimensional vector  $(t(x_1), t(x_2), \dots, t(x_m))$ . Linear least squares is used to find the solution of the equation. Given a sample  $x$ , the network predicts output label vector  $c(x)$  and the threshold value for  $x$  is gotten by solving equation  $t(x) = w^T \cdot c(x) + b$ .

The computational complexity of evaluating derivatives of the error function is linear with the neuron numbers. Three main components are composed of computation, feed-forward process, backpropagation process and updating weights process. In the feed-forward process, to compute  $b_i$  and  $c_j$ , the computation cost is mainly on evaluating the sums and activation function. In the backpropagation process, the computational complexity of  $d_k$  and  $e_s$  is  $O(Q)$ . In the updating weights process, the overall computational complexity is  $O(W)$  where  $W$  is the total number of weights .

# Chapter 8

## Experiment

The experiment has the identical hardware and software environment as described in weather classification project.

### 8.1 Dataset

We generate 1000 raw RGB images with size  $16 \times 16$ . 40 images are reserved for testing and 960 images are used for training. Each image has 3 labels  $y_i \in (-1, 1)$  representing three attributes, red, green and blue. If the image has a colour attribute, the corresponding label value is 1, otherwise  $-1$ .

### 8.2 Details of Network

The neural network is built from scratch up. First, we need to determine how many layers needed for the network. In [44], Lippmann has proved that two hidden layers are able to create classification regions of arbitrary desired shape. However, Kolmogorov [45] showed that the superposition of continuous one-dimensional function can represent a continuous function with several variables. Then we set up a neural network with one fully connecting hidden layer between the input and output layer. Between two layers, we put a ReLU layer to apply the non-saturating activation function  $f(x) = \max(0, x)$  which helps the decision function increase the non-linear properties.

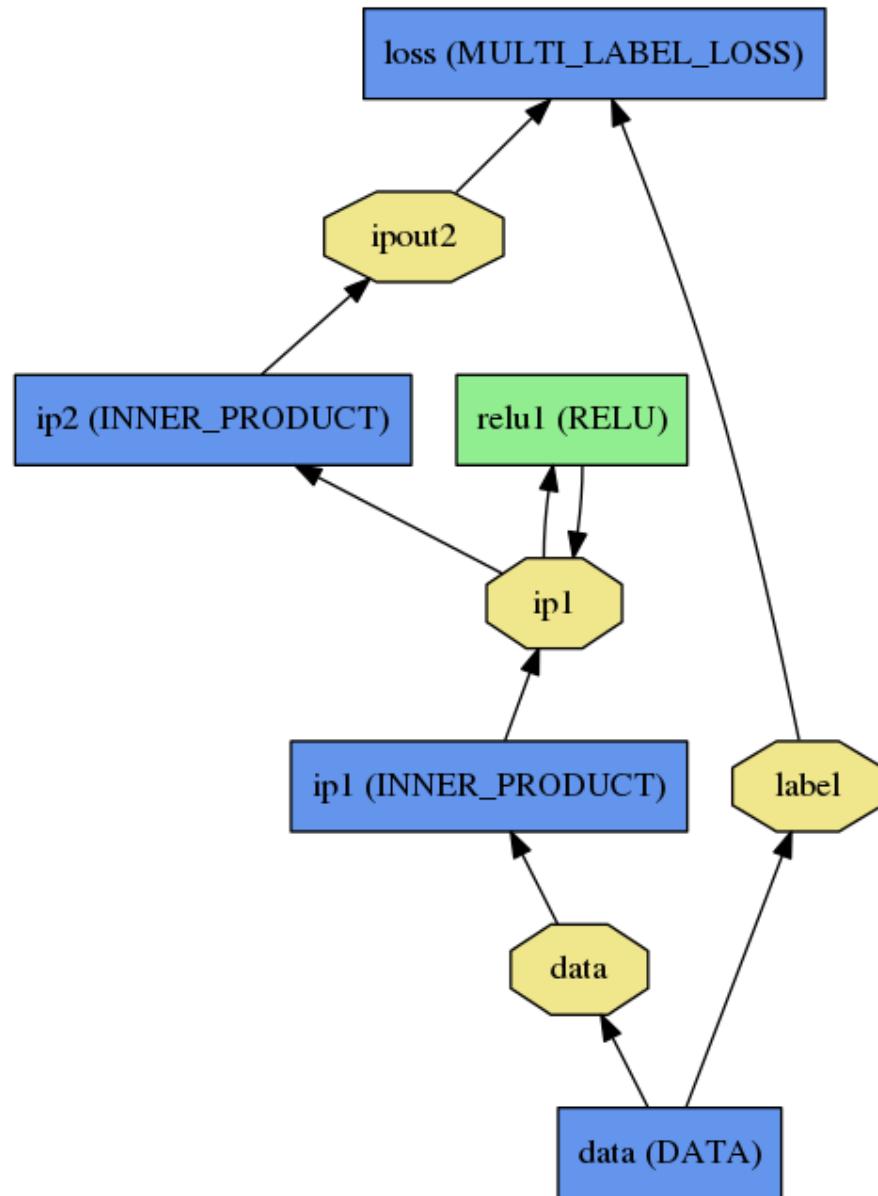


FIGURE 8.1: Network Topology

Following that, it needs to determine how many neurons needed in the hidden layer.

The number of neurons in the hidden layer depends mainly on

1. number of input and output neurons
2. number of training samples
3. layer connecting types
4. training algorithms
5. type of activation function

## 6. regularisation

It is hard to determine the best number of hidden neurons without evaluating several trained models and comparing the generalisation error rates among the models. Lacking hidden neurons will lead a high training error rate and poor generalisation performance because of underfitting and high bias. On the other hand, exceeding neurons will achieve a high training error rate and poor generalization performance because of overfitting and high variance. One of crucial considerations is to evaluate the hidden neuron effects on the bias/variance trade-off.

There are many rules of thumb to determine the number of hidden neurons [46]

1.  $\frac{2}{3}$  of the sum of the input neurons plus the output neurons.
2. less than twice the size of the input layer.
3. following geometric pyramid rule, between the number of input neurons and the output neurons.

It is time-consuming to find the optimal number of hidden neurons. We will start from a number of neurons which is small and increase the number gradually by evaluating the performance of the network.

The different layers are fully connecting because we want to map each patch to the labels of the image. The information of each colour distributes in different positions of the image. The values of the weights and bias are initialised randomly. For weights, the *xavier* algorithm is applied to determine the scale of initialisation depending on the number of input and output neurons. The bias is simply initialised as a constant 0. The batch size is 40 means that there are 40 images fed into the neural network each time.

In the training process, the base learning rate is 0.0001. The learning rate of weights is equal to base learning rate, while the learning rate of bias is two times of the base learning rate. The parameter momentum is 0.9 to smooth the weight updating across iterations so that the learning process will be fast. The parameter of weight decay is 0.0005.

The evaluation metric is label-metric. Each label will be counted separately and statistics will be calculated in total.

Neuron number	Sensitivity	Specificity	Harmonic Mean	Precision	F1 Score
4	0.9245	0.9552	0.9396	0.9423	0.9333
50	0.9245	0.9701	0.9468	0.9608	0.9423
100	0.9623	0.9701	0.9662	0.9623	0.9623
150	0.9057	0.9552	0.9298	0.9412	0.9231
200	0.9811	1.0000	0.9905	1.0000	0.9905
250	0.9434	1.0000	0.9709	1.0000	0.9709
300	0.9811	0.9701	0.9756	0.9630	0.9720
350	0.9434	0.9701	0.9566	0.9615	0.9524
400	0.8868	0.9701	0.9266	0.9592	0.9216
450	0.9245	0.9701	0.9468	0.9608	0.9423
500	0.9622	0.9701	0.9662	0.9623	0.9623

TABLE 8.1: Test results for different number of hidden neurons.

### 8.3 Results

The test results with different number of hidden neurons are showed in table 8.1 and figure 8.2. Results show that the model with 200 neurons in the hidden layer has the best performance.

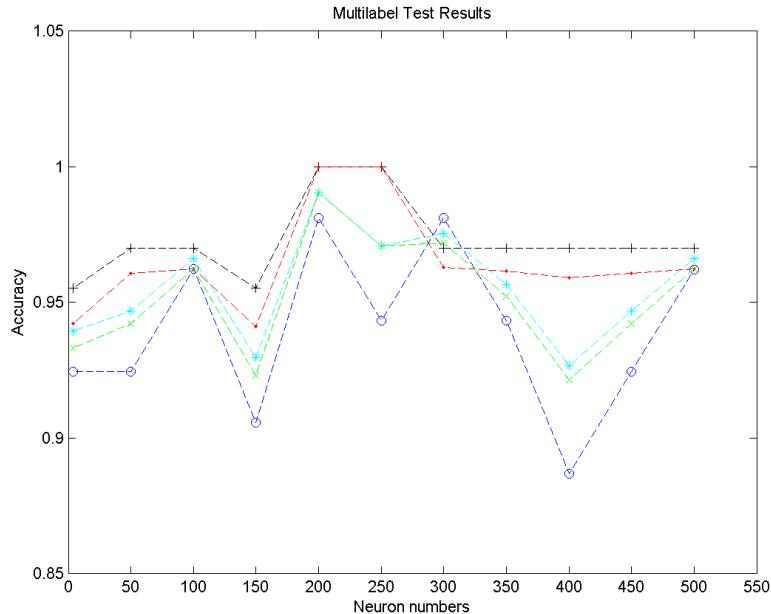


FIGURE 8.2: The test results for different number of hidden neurons. Blue circle for Sensitivity. Black plus for Specificity. Cyan star for Harmonic Mean. Red dot for Precision. Green x for F1 Score.

In figure 8.3, it shows that the learning speed is quick. After reaching high accuracy, the accuracy keeps stable.

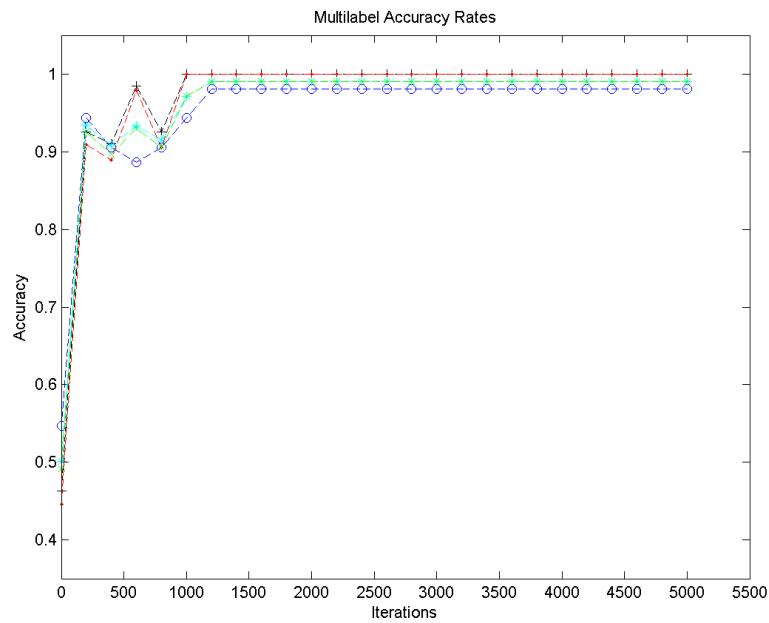


FIGURE 8.3: The learning speed for 200 neurons in the hidden layer. Blue circle for Sensitivity. Black plus for Specificity. Cyan star for Harmonic Mean. Red dot for Precision. Green x for F1 Score.

In figures 8.4 8.5, the ROC curve of test results shows that the average accuracy is high. In detail, the micro-accuracy for label 0(red) is the highest, followed by accuracies for label 1(green) and label 2(blue).

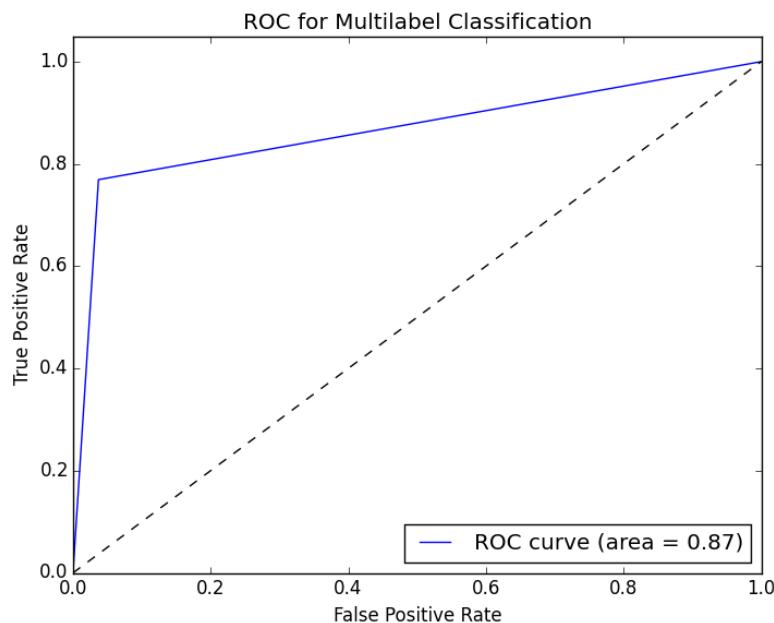


FIGURE 8.4: The ROC curve for 200 hidden neurons

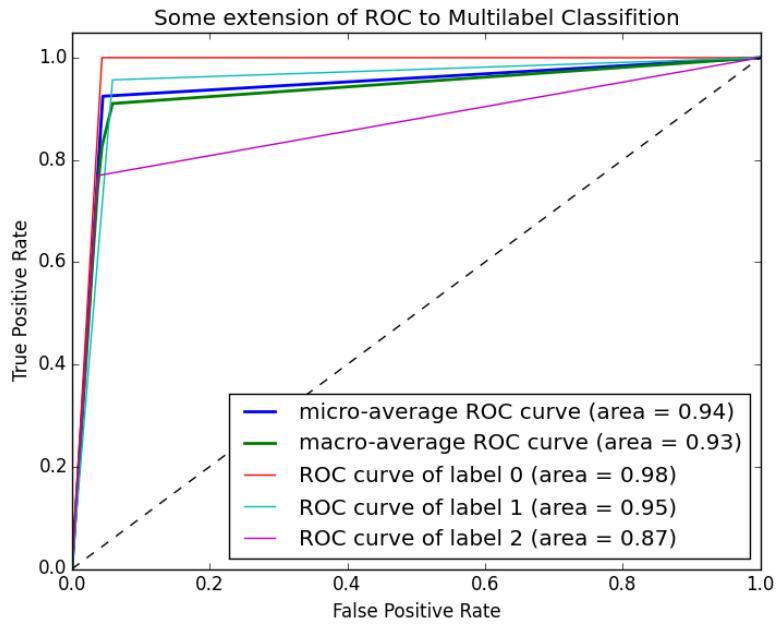


FIGURE 8.5: The ROC curve for 200 hidden neurons

## 8.4 Conclusion and Future Work

We have presented an approach to classifying multi-label images. With the increasing label numbers, the correlation between data and labels is complex. A Neural network can map data and labels efficiently. The updated loss function evaluates the model efficiently and leads optimisation quickly.

The experiment shows that the neural network is able to do complex multi-label classification perfectly. In the experiment, classifying the artificially images shows the high generalisation performance. It needs to update the network architecture for practical datasets in the future.

# Bibliography

- [1] Machine learning: Introduction to the artificial neural network. <http://www.durofy.com/machine-learning-introduction-to-the-artificial-neural-network/>, 2014. Accessed: 2015-12-30.
- [2] A gentle introduction to artificial neural networks. <https://theclevermachine.wordpress.com/tag/regression/>, 2014. Accessed: 2016-1-10.
- [3] Convnetjs demo: toy 2d classification with 2-layer neural network. <http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>, 2014. Accessed: 2016-1-10.
- [4] Khursiah Zainal-Mokhtar and Junita Mohamad-Saleh. An oil fraction neural sensor developed using electrical capacitance tomography sensor data. *Sensors*, 13(9):11385–11406, 2013.
- [5] Big data and hadoop- drivers of future. <https://affineanalytics.wordpress.com/>, 2013. Accessed: 2016-1-15.
- [6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [7] Cs231n convolutional neural networks for visual recognition. <http://cs231n.github.io/convolutional-networks/>, 2015. Accessed: 2015-11-15.
- [8] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

- [9] A survey on transfer learning. <http://www.slideshare.net/azuring/a-survey-on-transfer-learning>, 2012. Accessed: 2015-11-5.
- [10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.
- [11] Cewu Lu, Di Lin, Jiaya Jia, and Chi-Keung Tang. Two-class weather classification. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *The 13th European Conference on Computer Vision (ECCV)*, 2014.
- [13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [14] Epilog laser systems. <http://www.yourhomeok.com/color-wheel-schemes>, 2016. Accessed: 2015-11-5.
- [15] Christopher M. Bishop. *Neural networks for pattern recognition*. Clarendon press Oxford, 1995.
- [16] Martin Roser and Frank Moosmann. Classification of weather situations on single color images. *Intelligent Vehicles Symposium, IEEE*, 2008.
- [17] Navid Serrano, Andreas Savakis, and Jiebo Luo. A computationally efficient approach to indoor/outdoor scene classification. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4, pages 146–149. IEEE, 2002.
- [18] Demir Gokalp and Selim Aksoy. Scene classification using bag-of-regions representations. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.
- [19] Martin Szummer and Rosalind W Picard. Indoor-outdoor image classification. In *Content-Based Access of Image and Video Database, 1998. Proceedings., 1998 IEEE International Workshop on*, pages 42–51. IEEE, 1998.

- [20] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1):2–23, 2009.
- [21] Aditya Vailaya, HongJiang Zhang, Changjiang Yang, Feng-I Liu, and Anil K Jain. Automatic image orientation detection. *Image Processing, IEEE Transactions on*, 11(7):746–755, 2002.
- [22] Matthew R Boutell, Jiebo Luo, Xipeng Shen, and Christopher M Brown. Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771, 2004.
- [23] Xunshi Yan, Yupin Luo, and Xiaoming Zheng. Weather recognition based on images captured by vision system in vehicle. In *Advances in Neural Networks-ISNN 2009*, pages 390–398. Springer, 2009.
- [24] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [25] Herbert Robbins and David Siegmund. A convergence theorem for non negative almost supermartingales and some applications. In *Herbert Robbins Selected Papers*, pages 111–135. Springer, 1985.
- [26] Geoffrey E Hinton. Learning translation invariant recognition in a massively parallel networks. In *PARLE Parallel Architectures and Languages Europe*, pages 1–13. Springer, 1987.
- [27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [28] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.
- [29] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173, 2008.
- [30] Jianxiong Xiao, James Hays, Krista Ehinger, Aude Oliva, Antonio Torralba, et al. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision*

- and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.
- [31] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.
- [32] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.
- [33] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL <http://arxiv.org/abs/1311.2901>.
- [34] Ali S Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.
- [35] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.
- [36] Grigoris Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *Dept. of Informatics, Aristotle University of Thessaloniki, Greece*, 2006.
- [37] Grigoris Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Machine learning: ECML 2007*, pages 406–417. Springer, 2007.
- [38] Nadia Ghamrawi and Andrew McCallum. Collective multi-label classification. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 195–200. ACM, 2005.
- [39] Wei Gao and Zhi-Hua Zhou. On the consistency of multi-label learning. *Artificial Intelligence*, 199:22–44, 2013.
- [40] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

- [41] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *Information Theory, IEEE Transactions on*, 39(3):930–945, 1993.
- [42] Sergio Bermejo and Joan Cabestany. Oriented principal component analysis for large margin classifiers. *Neural Networks*, 14(10):1447–1461, 2001.
- [43] André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Advances in neural information processing systems*, pages 681–687, 2001.
- [44] Richard P Lippmann. An introduction to computing with neural nets. *ASSP Magazine, IEEE*, 4(2):4–22, 1987.
- [45] Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Amer. Math. Soc. Transl*, 28:55–59, 1963.
- [46] Jeff Heaton. *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.