University of Adelaide

Thesis

by

Junjie Zhang

A thesis submitted in partial fulfillment for the
degree of Master

in the

Chunhua Shen
School of Computer Science

January 2016

Declaration of Authorship

I, Junjie Zhang, declare that this thesis titled, 'Weather Classification' 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 this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University 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 paper, we present a novel approach to classify cloudy and sunny weather. Inspired by recent study of deep multi-layer neural network and spatial pyramid polling, generate a model based on imagenet dataset. Starting with parameters trained from more than 1 million images, we fine-tune the parameters. Experiment demonstrates that our classifier can achieve the state of the art accuracy.

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

Abbreviations

LAH **L**ist **A**bbreviations **H**ere

Physical Constants

Speed of Light $c = 2.997\ 924\ 58 \times 10^8\ \mathrm{ms}^{-\mathrm{S}}$ (exact)

Symbols

a	distance	m
P	power	W (Js^{-1})
ω	angular frequency	$rads^{-1}$

For/Dedicated to/To my...

Weather Classification

Chapter 1

Introduction

1.1 Overview

Computer is one of the most significant inventions in history. It provides huge power in data processing field, like scientific computation. Also, computer system can aid human being in daily life, for example driverless vehicles. However, human brain still has compelling advantage in some fields, like 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 daily life for a long period and human brain has developed an advanced neural and cognitive system for such tasks.

Weather classification is one of the most important pattern recognition tasks which relates our work and lives. There are several major kinds of weather conditions, like sunny, cloudy, rainy. And people can classify them easily through eyes. However, it is not a easy job for machines, especially in computer vision literacy.

In this thesis, I describe an approach to this problem of weather classification based upon the new big trend in machine learning, and more precisely, deep learning. It differs with feature detectors method that extracts features manually and training a model to do classification.

Compared to shadow learning which includes decision trees, SVM and naive bayes, deep learning passes input data through several non-linearities functions to generate features and does classification based on the features. It generates a mapping and finds a optimist solution.

1.2 Statistical Pattern Recognization

In the statistical approach, the pattern is represented in terms of d dimensional feature vector and each element of the vector describes some subjects of the example. In brief, three components are essential to do statistical pattern recognization. First is a representation of the model. Second is an objective function used to evaluating the accuracy of the model. Third is an optimization method for learning a model with minimum errors.

1.3 Artificial Neural Networks

Artificial neural networks(ANNs) 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 computer vision.

The single-layer perceptron has shown great success for a simple model. For increasing complex models and applications, multi-layer perceptron has exhibited the power of features learning. With hardware developing At the same time, the demanding of more efficient optimizing and model evoluation methods is needed for BIG DATA.

Recent development in ANNs approaches have giantly advanced the performance of state-of-the-art visual recognition systems. With implementing deep convolutional neural networks, it achieves top accuracy in ImageNet Challenge. The model has been used in related fields and succeeds in competition.

1.4 Weather Classification

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

There are some obstacles in front of weather classification. First, because inputting pixels are independent and high dimension, say a 500x500 RPG image means 750000 pixels, computation is huge. Second, some simple middle level image characters are difficult to machines, like light, shading, sun shine. It is still not easy to detect these characters with high accuracy. Thirdly, there are no decisive features, for example brightness and

lightness, to classify scene. For example, sun shine can be both found in sunny and cloudy weather. Last but not least, outdoor images are various background.

Chapter 2

Background

2.1 Single-Layer Networks

Artificial neurons were introduced as information processing devices more than fifty years ago[1]. Following the early work, perceptrons were deployed in layers to do pattern recognition job. A lot of resource was invested in researching 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).

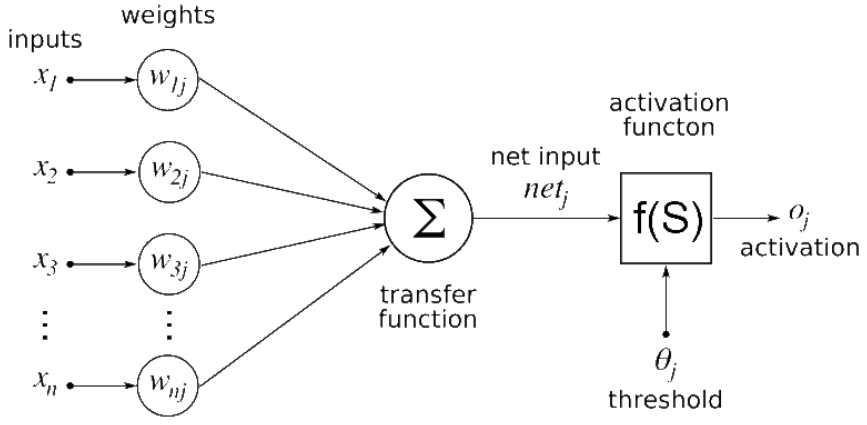


FIGURE 2.1: Diagram of a perceptron.

Each output is updated according to the equation:

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

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

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

and it can be plotted out

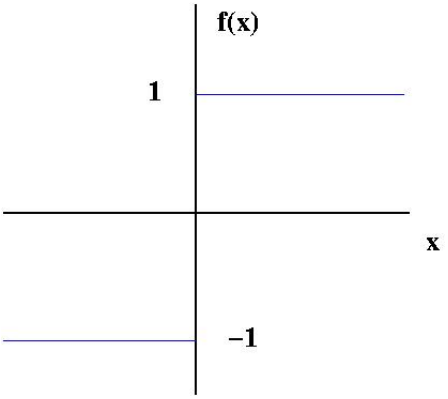


FIGURE 2.2: Threshold function

Besides threshold function, there are servel deterministic action functions.

- Linear function

$$f(h) = h \tag{2.3}$$

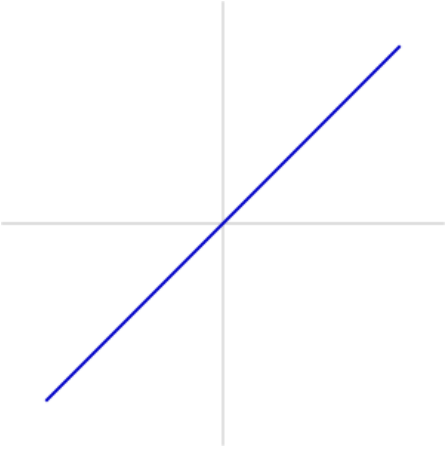


FIGURE 2.3: Linear function

- Sigmoid function

$$f(h) = \frac{1}{1 + e^{-h}} \tag{2.4}$$

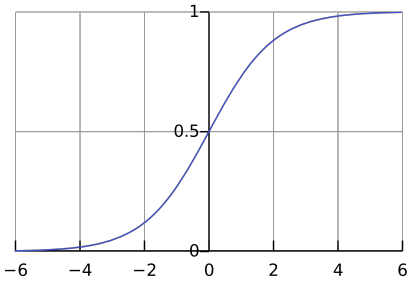


FIGURE 2.4: Sigmoid function

- tanh function)

$$f(h) = tanh(h) \tag{2.5}$$

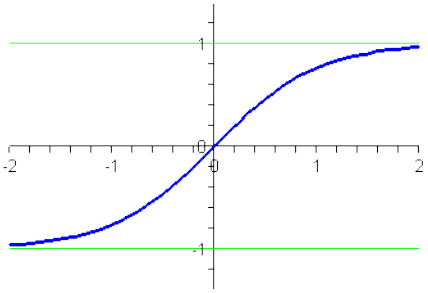


FIGURE 2.5: Tanh function

We can also represent 2.2 in vector notation, as in

$$y = f(h) = f(\mathbf{w} \cdot \mathbf{x}) \tag{2.6}$$

where $\mathbf{w}$ and $\mathbf{x}$ can be regarded as $nx1$ dimensional column vectors, and n is the number of input data. The term $w \cdot x$ in 2.5 constructs a $(n-1)$-dimension hyperplane which passes the origin. The hyperplane can be shifted by adding an parameter to 2.1, for example

$$y = f(h) = f(\mathbf{w} \cdot \mathbf{x} + \mathbf{b}) \tag{2.7}$$

We can have the same effect by putting a constant value 1 and increase 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:

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

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

$$y = f(\sum_{i=0}^{n} w_i x_i) = 1 \quad x \in ClassA$$
$$y = f(\sum_{i=0}^{n} w_i x_i) = 0 \quad x \in ClassB$$

Single layer neural network classifier is simple to implement, while it can only support a linear decision boundary. We can build a pretty simple neural network to acquire intuition behind mathematical theory. The network has no bias and one neuron which means it has one weight, say $w1$. And we implement a logistic sigmoid activation function on the multiply of input data and weight $w1$. Therefore, the network can map the input data a_0 onto an output a_{out} based on the function

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

where $f()$ is a 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**??**.

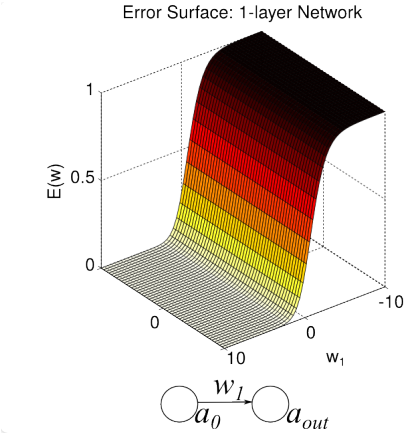


FIGURE 2.6: Error surface for a single layer neural network. Source: Internet

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 dataset.

FIGURE 2.7: Two types of dataset. Left one can be separated by a single layer neural network. Right one cannot be separated by a single neural network.

2.2 Multi-Layer Networks

Single layer networks have some important limitation in terms of the representing range of functions. We are seeking to learn the nonlinearity as the linear discriminant.To improve the representation capability, we can stack layers networks. This is the approach of multilayer neural networks. Multilayer neural networks implement linear discriminants via mapping input data to a nonlinear space. They can use fairly simple algorithms to learn form of the nonlinearity from training data.

In the thesis, we limit multi-layer networks in subset of feedforward networks. As feedforward networks can provide a general mechanism for representing nonlinear functional mapping between a set of input data and a set of labels. The 2.7 is a feedforward network having two layers of adaptive weights.

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

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

In the 4.2, outer activation function could be different with the inner one.

There are some choices for activation functions, sigmoid and tanh are related and can provide high capability with continuous input data. Logistic activation function sigmoid can be represented as

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

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

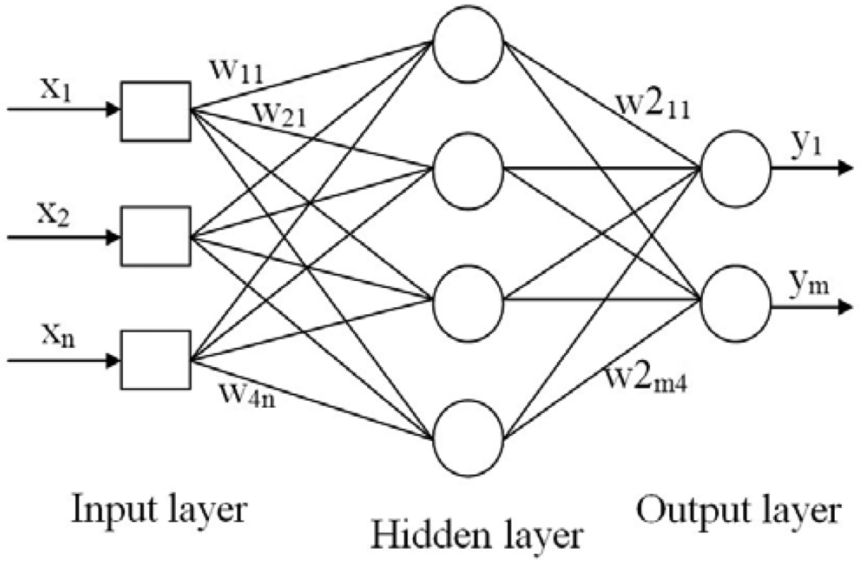


FIGURE 2.8: Diagram of a feedford neural networks.

equivalent activation function tanh which can be represented as

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

The three layer neural network is capable of approximating any function with enough hidden units which means the networks with two layers of weights and sigmoid nonlinearities can provide any accuracy in classification problems.

Again, we can use a simple example to illustrate the power of multilayer neural network. In this example, we have one input, one hidden neuron and one output. There is no bias in 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

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

where $f()$ is logistic function. With varying w_1 and w_2, the error surface can be represented And the samples which cannot be separated by single neural network can be done by multilayer neural network.

2.3 Backpropagation

Multi-layer neural networks can represent mapping from input data to output classes. How to learn a suitable mapping from training dataset? And there is no explicit mapping

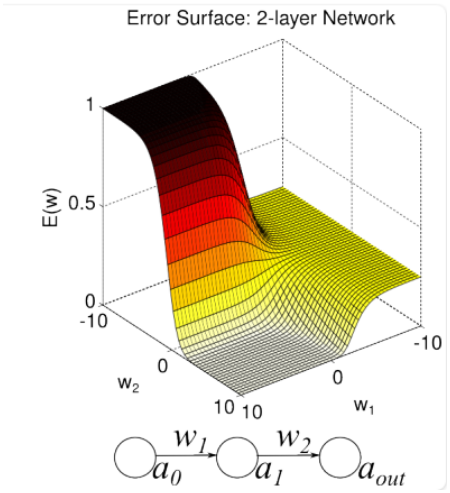


FIGURE 2.9: Error surface for a multilayer neural network. Source: Internet

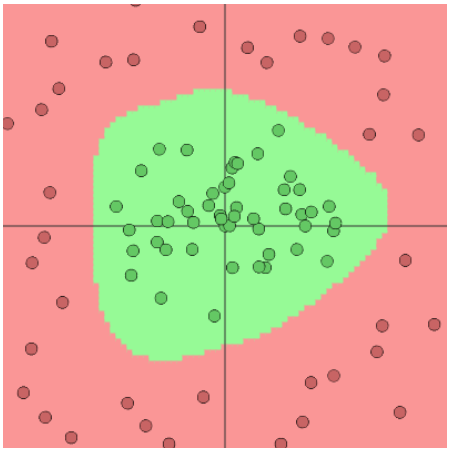


FIGURE 2.10: Source: Internet

between output and hidden units. This will be resolved by a popular learning algorithm-backpropagation.

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

Backpropagation can be applied to find the derivatives of an error function related to the weights and biases 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 optimization schemes can be implemented to compute the adjustment of weights based on derivatives. After putting data forward propagation through the network, we can get the output result. It updates weight changes based on grandient descent. Suppose the network has i inputs, h hidden units and k outputs.

The update equation can be represented as

$$w(j+1) = w(j) + \Delta w(j) \tag{2.14}$$

where $\Delta w(j)$ defined as

$$\Delta w(j) = -\eta \frac{\partial E}{\partial w} \tag{2.15}$$

For the hidden to output layer weights

$$\Delta w(jk) = -\eta \frac{\partial E}{\partial w_{jk}} = -\eta \delta_k y_j \tag{2.16}$$

where

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

For the hidden layer weights

$$\Delta w(ij) = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \delta_j y_i \tag{2.17}$$

where

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

The δ_j of a hidden unit is based on the δ_ks of output units to which it links. To minimise the error function$E$ with gradient descent, it needs the propagation of errors backwards.

2.4 Softmax classifier

Softmax function, also named normalized exponential, is a generalization of 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 normalizer of categorical probability distribution, it can be used in probabilistic multiclass classification methods.

The softmax function derives from log linear models and interprets the weights in terms of convenient odds ratios. We can constrain the layer input to the output neurons to be positive and divide by the sum.

The softmax layer begins the same way as 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 layer number, k is input data number and j is output neuron number. Then it implements a softmax function to the z_j^L and the activation a_j^L

of the j output neuron is

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.18}$$

The equation 2.17 implies that the output values are all positive and the sum of all values $\sum_k a_k$ is 1.

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

Suppose to predict an unseen sample x, we will use hypothesis to estimate the probability $P(y = k|x)$ for each value $k = 1, \ldots, K$. For example, we want to compute the probability of the class label on each of K different possible values. Then the neural network will output a K dimensional vectors which represent K estimated probabilities.

$$h_W(x) = \begin{bmatrix} P(y = 1|x; W) \\ P(y = 2|x; W) \\ \vdots \\ P(y = K|x; W) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(W^{(j)\top} x)} \begin{bmatrix} \exp(W^{(1)\top} x) \\ \exp(W^{(2)\top} x) \\ \vdots \\ \exp(W^{(K)\top} x) \end{bmatrix} \tag{2.19}$$

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

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

$$H(p, q) = -\sum_x p(x) \log q(x) \tag{2.20}$$

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

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

$$P(y_i \mid x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \tag{2.21}$$

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

2.4.1 Practical issues

For numerical view, the exponentiation computation is easy overflow. Thus, the output of softmax function is not numeric stable through computing $e^{f_{y_i}}$ and $\sum_j e^{f_{y_j}}$ in straight way. It is important to implement with a normalization trick. It is mathematically equivalent if multiplying a constant C both with top and bottom of the fraction.

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+\log C}}{\sum_j e^{f_j+\log C}} \tag{2.22}$$

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

2.4.2 Error function

The error function is used to evaluate the performance of the 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 as

$$L(W) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} I\left\{y^{(i)} = k\right\} \log \frac{\exp(W^{(k)\top}x^{(i)})}{\sum_{j=1}^{K}\exp(W^{(j)\top}x^{(i)})}\right] \tag{2.23}$$

where this generates the logistic regression error function

$$L(W) = -\left[\sum_{i=1}^{m}(1-y^{(i)})\log(1-h_W(x^{(i)})) + y^{(i)}\log h_W(x^{(i)})\right] \tag{2.24}$$

$$= -\left[\sum_{i=1}^{m}\sum_{k=0}^{1} 1\left\{y^{(i)} = k\right\}\log P(y^{(i)} = k|x^{(i)}; W)\right] \tag{2.25}$$

Similar to the logistic regression error function, the softmax error function sum over the predicted different K values of the class value. In the softmax regression, the posterior probability distribution can be represented as

$$P(y^{(i)} = k|x^{(i)}; W) = \frac{\exp(W^{(k)\top}x^{(i)})}{\sum_{j=1}^{K}\exp(W^{(j)\top}x^{(i)})} \tag{2.26}$$

It is not easy to solve equation 2.23. Usually an optimization algorithm can help to approximate the optimal value. Taking derivative with respect to weights, we can get

the entire gradient

$$\nabla_{W^{(k)}} L(W) = -\sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k|x^{(i)}; W) \right) \right] \tag{2.27}$$

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

2.5 Training protocols

In supervised learning, we have training dataset which is 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 training dataset and update weights every time depending on output of neural networks. In batch training, we use some samples and pass them through network, then update weights. In online training method, each sample of training dataset is used once and weights are updated each time. We usually call one time of passing all training dataset through neural networks one epoch.

2.6 Stochastic Gradient Descent

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

$$L(f_w) = \sum L(y, f_w(x)) \tag{2.28}$$

Gradient descent is a method which starts from a random point, then moves to a nearby point that is downhill repeatly. It converges on a minimum possible loss.

Stochastic gradient descent is a subtype of gradient descent. It only considers a single training point and move to nearby point based on

$$w = w - \eta \Delta L(w) = w - \eta \sum_{i=1}^{n} \Delta L_i(w) \tag{2.29}$$

where η is the learning rate and $L_i(w)$ is the value of the loss function at the i^{th} sample. Although stochastic gradient descent does not guarantee convergence, it is fast.

2.7 Convolutional Neural Networks

Convolutional Neural Networks[2] are widely applied in image data and they achieved top rank in image classification competetion[3]. Convolutional neural networks have three types of layers, convolutional layer, pooling layer and fully connected layer. At the output of the network, a softmax layer is used to do classification job. Convolutional layer is used to detect invariants in locat receptive fields. Pooling layer is downsampling feature maps. Fully connected layer makes the output of each unit an activation of dot product of all inputs.

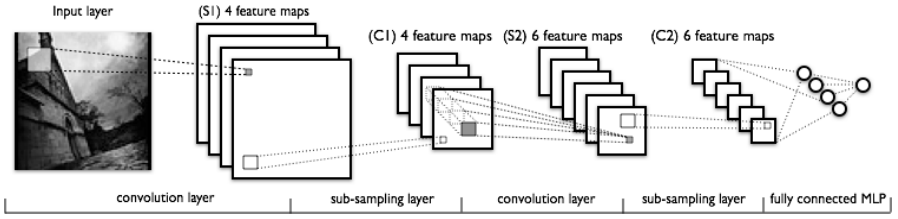


FIGURE 2.11: Diagram of LeNet.

2.8 Spatial Pyramid Match

Spatial pyramid match[4] is used to classify high-level semantic attributes, based on low-level features. The method subdivides a 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 images.

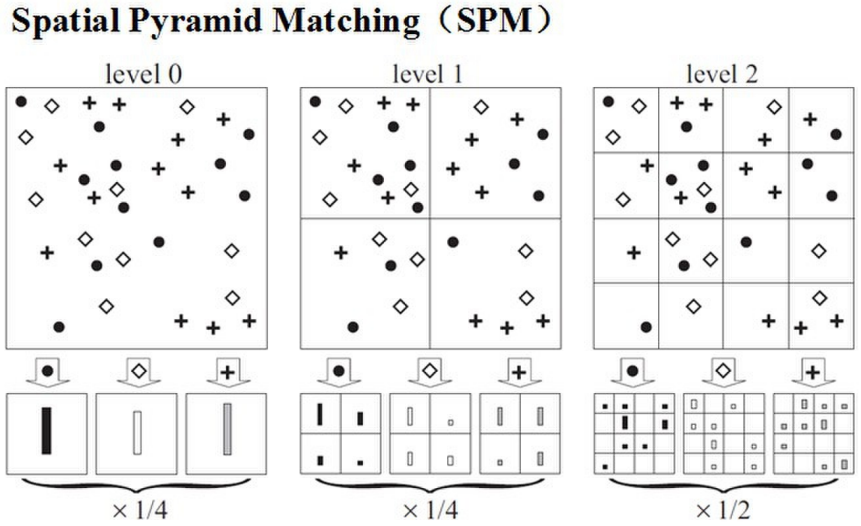


FIGURE 2.12: Diagram of Spatial Pyramid Match.

2.9 Transfer Learning

In machine learning literature, transfer learning[5] focuses on storing knowledge from one domain and applying it to a related problem. It has two benefits. One is saving time to build a model from scratch up. Another is saving effort to collect training data.

A domain with two components, a feature space and a marginal probability distribution, can be represented

$$D = \{\chi, P(X)\} \tag{2.30}$$

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

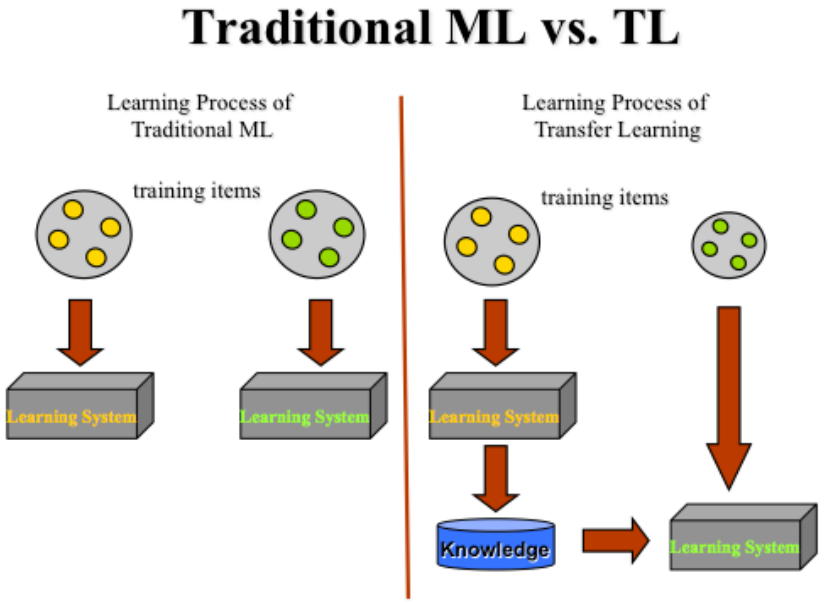


FIGURE 2.13: Diagram of Transfer Learning.

Chapter 3

Methodology

3.1 Dataset

There are over 15 million labeled images in ImageNet database belonging to about 22,000 categories. The images were collected from the Internet and labeled manually. From 2010, an annual competition named the ImageNet Large-Scale Visual REcognition Challenge (ILSVRC) has been held. The competition uses a subset of dataset which is 1000 images in 1000 categories. Totally, there are over 1 million training images and 50,000 validation images and 150,000 images for testing.

FIGURE 3.1: 2 Figures from ImageNet

For weather dataset, it contains 10,000 images for two categories evenly, cloudy and sunny. They are from three sources, Sun Dataset[6], Labelme Dataset[7] and Flickr. They were classified manually and similar images are removed. There are no unambiguous images.

FIGURE 3.2: 2 Figures from Weather Dataset

The two datasets are different. ImageNet dataset is for object classification and weather dataset is for scenes classification. Because the two datasets consists of different resolution images, we have to resize them to a fixed resolution of $256x256$ for constant dimension input for neural networks. To train the model with ImageNet images, we resize shorter side of images to length 256 and then cropped out a $256x256$ image from center.

3.2 Data Argument

The model has about 60 million neurons, and it is essential to avoid overfitting. One of methods is to do data argument. Dataset is artificially enlarged via image transformations and horizontal reflections. For each $256x256$ image, the network extracts five $224x224$ patches from four corners and center and reflects them horizontally. Hence, there are 10 patches for 1 image in total.

3.3 Spatial Pyramid Pooling

To improve the capability of generalization, a spatial pyramid pooling layer is deployed behind the fifth convolutional layer. A set of bin sizes are set to discern different size local information from output of the fifth convolutional layer. A sliding window pooling is implemented on the feature maps after the fifth convolutional layer. Assuming dimension of feature map is axa and bin size is n, each window size is $\lceil a/n \rceil$ and stride size is $\lfloor a/n \rfloor$ where $\lceil\rceil$ and $\lfloor\rfloor$ denote ceiling and flooring operations. For l level pyramid, there are l spatial pyramid pooling layers, The the l layers will be concatenated into a fully connected layer. The bin sizes are 1, 2, 3 and 6.

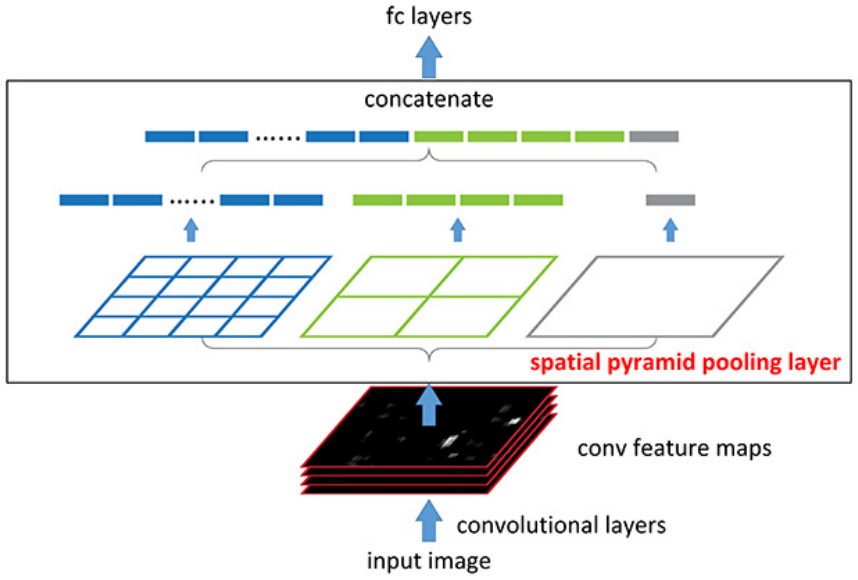


FIGURE 3.3: Diagram of Spatial Pyramid Pooling layer

With implementing spatial pyramid pooling layers in convolutional neural networks, all contribution of different scales are considered.

3.4 Convolutional Neural Networks Architecture

Due to the significant performance of AlexNet [3] deep convolutional neural networks, we train a model based on it. The architecture of the network has seven hidden adaptive layers-five convolutional and two fully connected layers. The network is very deep and

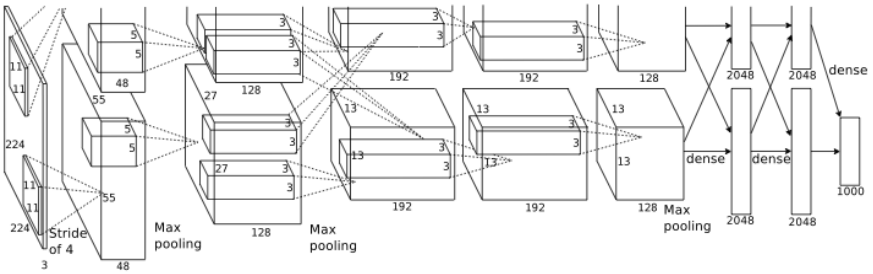


FIGURE 3.4: Architecture of AlexNet

the early layers are split over on two GPUs. It is able to give 62.5% accuracy rates with one prediction and 83% accuracy rates with five predictions.

The network replaces each neuron's outputs nonlinearity function f from $f(x) = tanh(x)$ or $f(x) = (1 + e^{-x})^{(} - 1)$ to Rectified Linear Units(RELUs)[8] which can accelerate learning speed several times faster than $tanh$ function.

In the network, there are three types of layers and they play different roles in the model. The input data dimension is $224x224x3$ which means the raw pixel value stored in a width 224, height 224 and with three color channels R,G,B matrix. In the first convolutional layer, there are 98 kernel of size $11x11x3$ with stride size 4 and outputs are 96 neurons. A max pooling layer downsamples the spatial dimensions. The second convolutional layer filters output of previous pooling layer with kernel size $5x5x48$. The third convolutiona layer owns 384 kernels of $3x3x256$ and receives outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3x3x192$ and the last convolutional layer has 256 kernels of size $3x3x192$. Each fully connected layers have 4096 units. At the end of network, there is a softmax layer with 1000 outputs.

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

TABLE 3.1: Parameters of model

Chapter 4

Experiment

We set up experiment environment on a desktop which has hardware i7 CPU, 8G RAM and a GeForce GTX 770 and software operation system Ubuntu Linux 14.04. We train models in Caffe[9] which is a open source framework and develop spatial pyramid pooling layer in the architecture.

4.1 Training Neural Network

We trained model with the 1000-category ImageNet2012 on deep learning framework CaffeNet [9] on a We use the base network architecture of fast(smaller) model [10] which achieved an excellent result in 2013 ImageNet Competition. We implement SPP with CUDA C language and deploy them before the first fully-connected layer.

In experiment, a subset of ImageNet dataset, which contains 1.2 million labelled high-resolution images depicting 1000 object categories for training and 50,000 validation images, is used as training dataset. ImageNet contains of multi-scale and some grey images which are converted to a constant input dimensionality. Then images are resized to $256x256$. And for grey images, they are combined it triple to minic a RGB image. Model is trained on the raw RGB values of pixels. Activation function is chosen Rectified Linear Units(ReLUs), which allows for faster and effective training of deep neural networks.

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

The network model **??** is trained with stochastic gradient descent as backpropagation learning rule, with a batch size of 128, and momentum of 0.9. The weights are initialized

from a 0 mean Gaussian distribution with standard deviation 0.01 in each layer. The neuron biases, in the $Conv_2$, $Conv_4$, $Conv_5$ and fully connected layers, are initialized with the constant 1, while biases in other layers are initialized with constant 0.

The learning rates are set equally for all layers. It was initialized at 0.01 and decreases with stepdown policy which means that it would drop by a factor of 10 after 100000 iteration. In total, the learning rate drops 3 times and accuracy stops after 370K iterations.

The training was regularised by dropout and weight decay. Dropout regularisation is implemented in the first two fully-connected layers and dropout ratio is set to 0.5. The neurons which are dropped out output zero and do not participate in backpropagation. Therefore, the neural network samples a different architecture each time. It greatly decreases complex co-adaptations of neurons because a neuron cannot depend on the existence of distinct other neurons. For weight decay ϵ, it is set to 0.0005 which means , after each weight update, the new weight is shrunk according to

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

4.2 Fine-tuning Model

As mentioned previously, the CNN network is pre-trained with aim of classifying objects in images. However, the target task is to classify weather scene and the pre-trained model contains more than 60 million parameters which are too high for training target model from scratch up, because the target dataset has only 10000 images in total. With the excellent accuracy achieved from previous works about transfer learning, it is a good approach to fine-tune previous model. The previous model does a job to classify 1000 categories and the target model does for two class classification.

Fine-tuning transfers weights of each layers in previous model to new one except the last fully connected layer. The last layer is taken over by a new one which contains the same amount of neurons as class number in the dataset, say two, and is initialized with random weights. One advantage of fine-tune is that it highly reduces the probability of overfitting while training with small dataset. The other advantage is the existing weights are close to optimal values and the gradient descent algorithm can converge quickly.

In the problem, we want to classify sunny and cloudy images, then the last layer of the previous architecture, $fc8$, is taken place by a layer with two neurons, one for sunny and one for cloudy. The model, trained using ILSVRC2012 dataset, is used to initialize most weights in the neural network for the fine-tuning experiment. At the same time,

1000 images are used to evaluate model. The model is fine-tuned in 10000 iterations which means that total training images are fed into CNN $\frac{128x10000}{9000} = 142$ times. 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 initialized 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 faster.

To minimise risk of overfitting, data augmentation is introduced in to experiment. Different version of patches are generated from each image via simple transformations, for example flipping and cropping. Some practical methods have been implemented and increase accuracy [3]. We do this job by increasing the spatial generalization capability of the model by cropping 5 different patches from four corners and center of images, and flipping them. Totally, we generate 10 different images with the same label.

4.3 Companion Experimental

To compare the performance of fine-tuned model, we do some a experiment with other method. We use a pre-trained model with AlexNet architecture and extract features from fc7. Then we apply SVM on the features and learn a classifier to do weather classification.

4.4 Experimental Result

The results in table4.1 shows that supervised pre-training model have excellent generalization capability.

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

TABLE 4.1: CNN means with original trained model, SPP means model deployed with spatial pyramid pooling layer

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

No single image dataset can totally capture variation in natural images, so even ImageNet is biased in some fields. Then pre-training may be make the CNN to overfit and then damage model generalization capability. To learning the process of ImageNet pre-training, we look into the effect of pre-training period on generalization performance

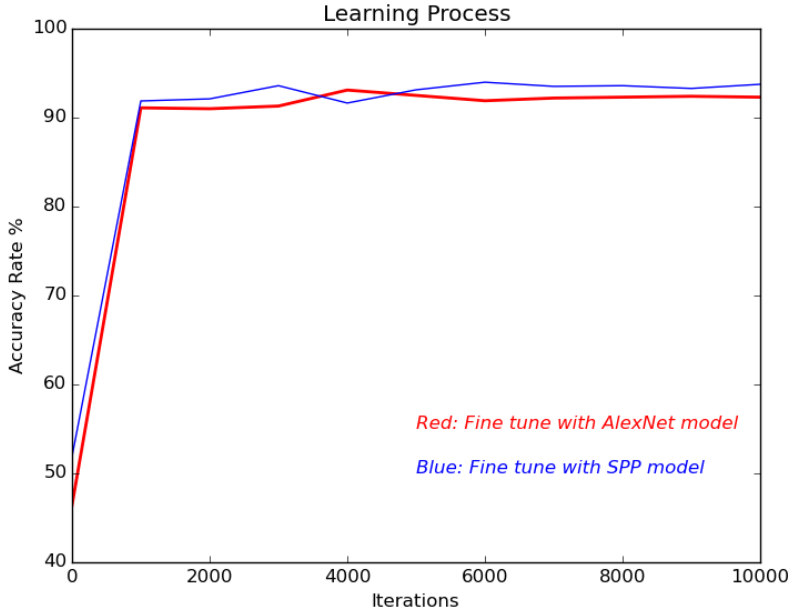


FIGURE 4.1: Finetune Process

with and without fine-tuning. In figure 4.1, we can find that longer pre-training improves performance.

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

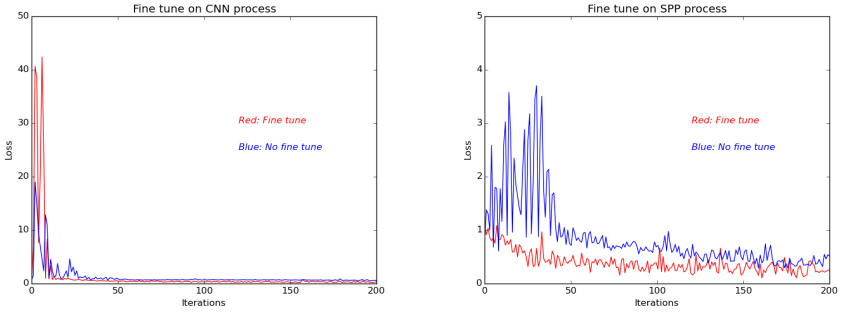


FIGURE 4.2: Finetune Process

From figure 4.2, we can find that the fine-tuning procedure produces a more smooth loss function curve and ceases with a better loss. In the left figure, the loss value of fine tune procedure is higher than no fine tune process at the beginning, then it shrinks sharply and keeps lower. In the other figure, we can find that starting loss value are both less than in the left figure. And the loss values of fine tune are less than no fine tune process. We can conclude that generally fine tune is a effective approach to reduce loss value and fine tune on SPP model can achieve a better result than CNN model.

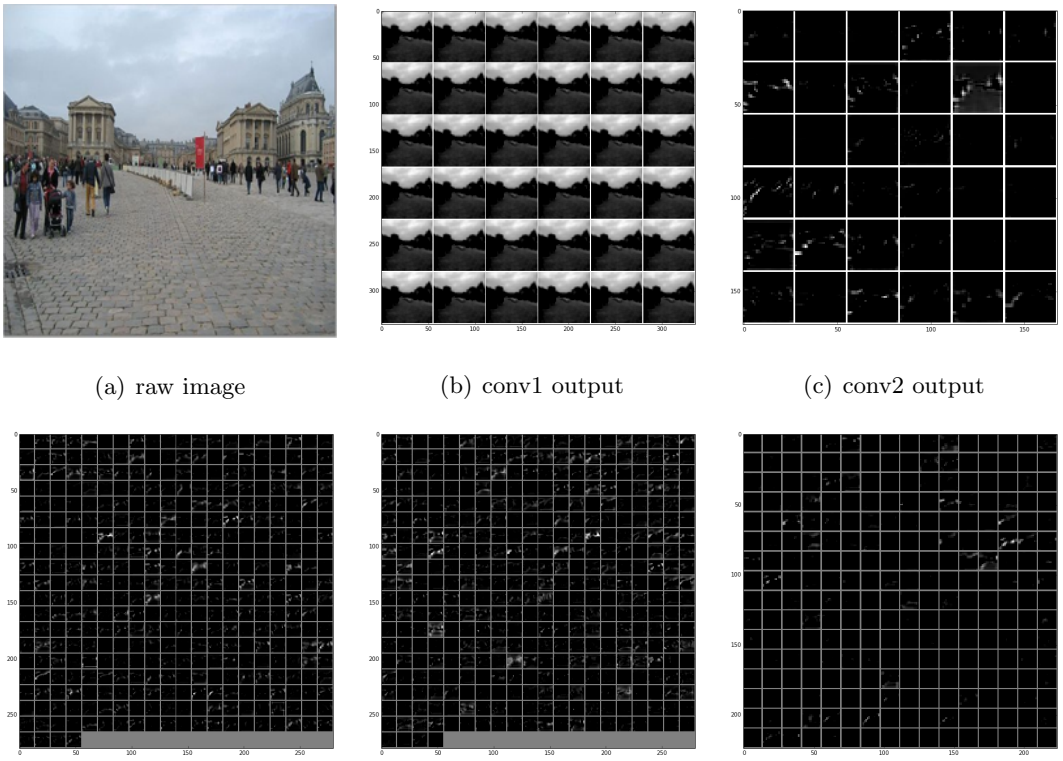
4.5 Architecture Analysis

Although CNNs have excellent performance in computer vision field, there is still a long way to understand insight working mechanism. In order to have more insight about the reason why deep layers can increase performance of visual sentiment classification, we will do some analysis on fine-tuned network.

The outputs of each layers have been treated as visual descriptors [11], while the vectors are composed by outputs of neurons in previous layer. While the lower layers encode low-level features, top layers have been used to capture high-level information. In other words, we can train classifiers basing on features extracted from each layer.

In fully-connected layer, units can be represented as d dimensional vectors, and no further manipulation is needed. The layer takes all outputs of previous layer ,for example CONV, POOL, NORM,whose feature maps are multidimensional, and flats the feature maps into d dimensional vectors. The outputs of fully connected layer are feed into a classifier. Two types of linear classifiers are used. They are linear Support Vector Machine and Softmax.

We have a look for the feature maps of each convolutional layers a cloudy image.



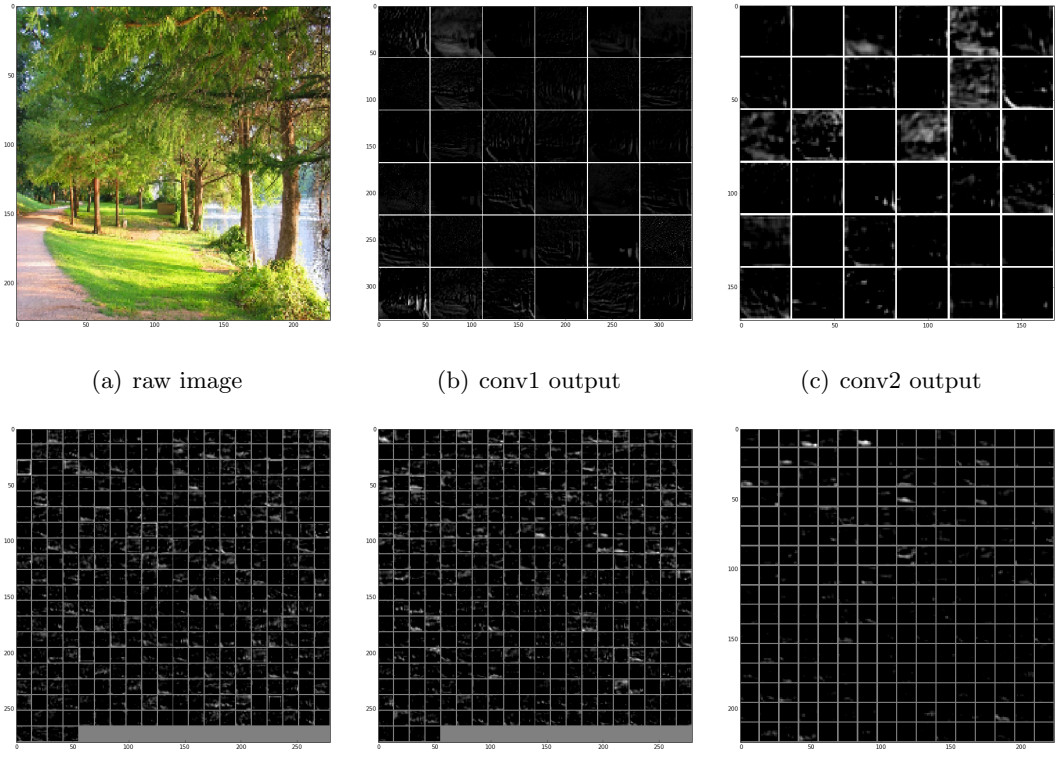
(a) raw image (b) conv1 output (c) conv2 output

(d) conv3 output (e) conv4 output (f) conv5 output

FIGURE 4.3: A cloudy image and outputs of convolutional layers

In Figure 4.3, a cloudy image is fed into CNN and b-d are outputs of different convolutional layers. For the sake of image size, there are only parts of filters are plotted in images b-d.

Compared to the previous cloudy image, we can have a look for similar outputs of a sunny image.



(a) raw image (b) conv1 output (c) conv2 output

(d) conv3 output (e) conv4 output (f) conv5 output

FIGURE 4.4: A sunny image and outputs of convolutional layers

The convolutional layers output the same number filters as Figure 4.3. Neural activations in fully connected layers can act as d-dimensional vectors, where d is the number of neurons. Not like the earlier layers, for instance CONV and POOLING whose feature maps are multimensional. In figure 3.4, we can get that the feature maps of CONV1 are $96x55x55$ dimension and the feature maps of CONV5 are $256x13x13$ dimension. The feature maps of CONV5 are downsampled by POOL5 and flatten into d-dimensional vectors and used as features for classifier. We can use two types of classifiers. One is linear support vector machine and the other is SoftMax.

4.6 Transfer Learning

Because we do not have sufficient size of dataset, comparing to ImangNet, we use a pretrained model based on ImageNet dataset and then use the CNN either as a fixed feature extractor or an initialization for our task. After fine tuning, weights have been updated for fitting new functions. Accordingly, the feature maps have some difference.

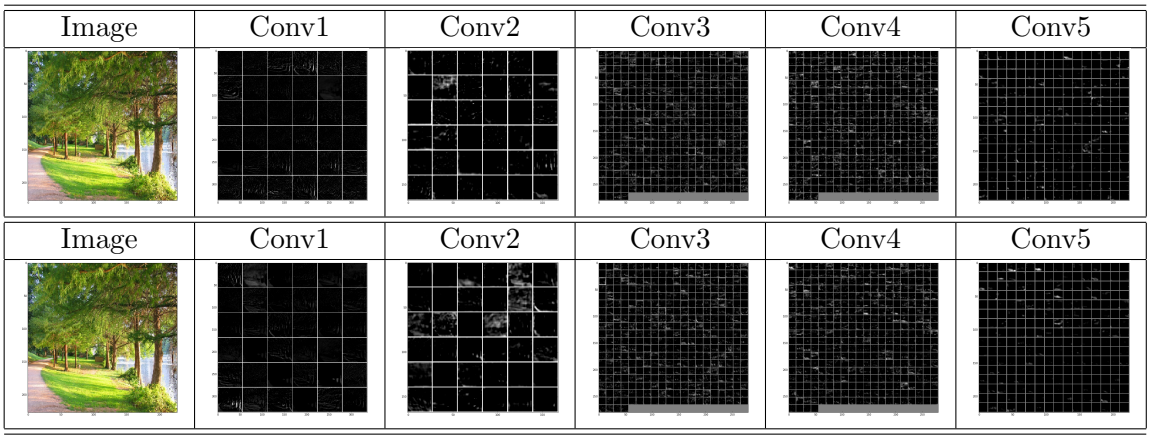


FIGURE 4.5: Visiualization of feature maps outputs between original CNN model and CNN-SPP model. The upper images are from original model and the lower images are from fine tuned model

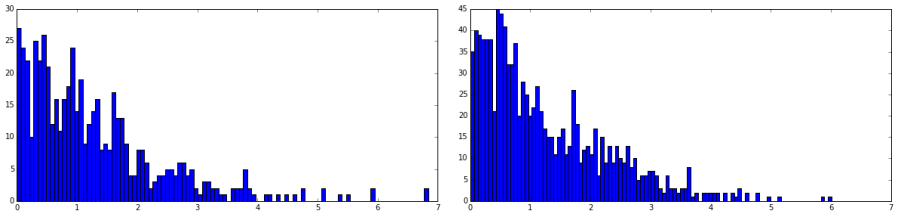


FIGURE 4.6: Histogram distribution of vectors from FC7. Left one is from CNN model and right one is from finetuned model

From the feature maps and histogram distribution maps, We can find that the outputs are slightly different. The fine tuned model generates smoother histogram distribution.

Chapter 5

Introduction

5.1 Overview

In machine learning literature, multi-label classification is a variant of the classification problem in which each instance has several labels. In general, the task of multi-label learning is to find a model that can maps inputs $\mathbf{x}$ to binary vector $\mathbf{y}$. While it is different with multi-class classfication in terms of the later one scalars outputs in classification problem.

FIGURE 5.1: Example Image

The difference between single-label classification and multi-label classification is the outputs of datasets. In Figure 5.1, we can classify it as a picture of a beach in classification literature $\in \{Yes, No\}$. In multi-label literature, we can tag sea, beach, chairs, sky, cloud for the picture $\in \{beach, sea, chairs, sand\}$.

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 $Y_1, ..., Y_L \in 2^L$
??

In general, there exists two main approaches of tackling the multi-label classification problem. One transfers 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 complete multi-label classification.

Several problem transformation methods can be applied to multi-label classification. A baseline method, named the binary relevance method[12], trains one binary classifier for each label independently. Depending on the results of the classifiers, the combined model predicts all labels for an unseen 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 benefits from scalability and flexibility because single-label classifier can be implemented to job. Support Vector Machine, Naive Bayes, k Nearest Neighbor have been used in the method[12].

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[13]. The random k-labelsets algorithm[14] utilizes multiple label powerset classifier which is trained on a random sub dataset of the labels. A voting scheme is used to predict unseen samples via a ensemble method.

Multilayer neural network can be trained to learn a model from multi-label examples. We will generate a 3 labels dataset and use a 2 layer network to do classification.

5.2 Multi-Label Learning

Let $X = R^d$ represent the domain of dataset and let $Y = 1, 2, .., L$ be the finite set of labels. Given that we have a set of training dataset $T = (x_1, Y_1), (x_2, Y_2), ..., (x_l, Y_l)(x_i \in X, Y_i \subset Y)$ which are extracted from an unknown distribution D. Our target of task is to learn a multi-label classifier $h : X \to 2^y$ via optimizing specific evaluation metric. However, instead of learning a multi-label classifier, we will learn a function f while $f(X) \to R^d$. 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 $y_1 \in Yi$ and $y_2 \notin Yi$. 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 dataset, there are 2^20 possible label sets maximun.

The challenge of huge combination of output labels is hard to overcome. One of methods is to investigate dependency among labels. 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 categories, based on the order of correlation of labels, for existing strategies to find the relation between labels. They are first order strategy, second order strategy and high-order strategy.

The first order strategy treats 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 correlations of labels.

The second order strategy considers pairwise relations between labels, for example, interaction between any pair of labels, or the ranking between related labels and unrelated labels. The method can help to achieve good generalization 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 high order strategy has the strongest model capabilities than previous two strategies on the cost of complexity and intensive computation.

5.3 Evaluation Metrics

In supervised learning settings, different metrics, like accuracy and area under the ROC curve, are used to evaluate the generalization performance of learning system. In multi-label learning, evaluating performance is more complicated than single-label problems with the increasing number of labels simultaneously. Therefore, two main types of evaluation methods are implemented in multi-label learning, say example-based metrics[15] and label-metrics[14].

The two types of metrics evaluate the outputs of classifier from different perspective. Given that $S = (x_i, Y_i)$ is the test instance and $h(\cdot)$ is the learned multi-label classifier. Example-based metrics achieve all class labels of each test example, and then compute the mean value of test set to evaluate generalization performance. Compared to 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 have a ground truth output and a predicted output of test instance. So the results of each test instance can be 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 instances 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

$$L = \bigcup_{i=0}^{N-1} L_i \tag{5.1}$$

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

$$I_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

5.3.1 Example-based Metrics

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

$$\frac{1}{N \cdot |L|} \sum_{i=0}^{N-1} |L_i| + |P_i| - 2\,|L_i \cap P_i| \tag{5.3}$$

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

$$\frac{1}{N} \sum_{i=0}^{N-1} I_{\{L_i\}}(P_i) \tag{5.4}$$

Precision is defined as

$$\frac{1}{N} \sum_{i=0}^{N-1} \frac{|L_i \cap P_i|}{|P_i|} \tag{5.5}$$

Recall is defined as

$$\frac{1}{N} \sum_{i=0}^{N-1} \frac{|L_i \cap P_i|}{|L_i|} \tag{5.6}$$

Accuracy is defined as

$$\frac{1}{N} \sum_{i=0}^{N-1} \frac{|L_i \cap P_i|}{|L_i| + |P_i| - |L_i \cap P_i|} \tag{5.7}$$

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

$$\frac{1}{N} \sum_{i=0}^{N-1} 2\frac{|P_i \cap L_i|}{|P_i| \cdot |L_i|} \tag{5.8}$$

5.3.2 Label-based Metrics

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

$$PPV(\ell) = \frac{TP}{TP + FP} = \frac{\sum_{i=0}^{N-1} I_{P_i}(\ell) \cdot I_{L_i}(\ell)}{\sum_{i=0}^{N-1} I_{P_i}(\ell)} \tag{5.9}$$

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

$$TPR(\ell) = \frac{TP}{P} = \frac{\sum_{i=0}^{N-1} I_{P_i}(\ell) \cdot I_{L_i}(\ell)}{\sum_{i=0}^{N-1} I_{L_i}(\ell)} \tag{5.10}$$

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

$$F1(\ell) = 2 \cdot \left(\frac{PPV(\ell) \cdot TPR(\ell)}{PPV(\ell) + TPR(\ell)} \right) \tag{5.11}$$

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

$$\frac{TP}{TP + FP} = \frac{\sum_{i=0}^{N-1} |P_i \cap L_i|}{\sum_{i=0}^{N-1} |P_i \cap L_i| + \sum_{i=0}^{N-1} |P_i - L_i|} \tag{5.12}$$

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

$$\frac{TP}{TP + FN} = \frac{\sum_{i=0}^{N-1} |P_i \cap L_i|}{\sum_{i=0}^{N-1} |P_i \cap L_i| + \sum_{i=0}^{N-1} |L_i - P_i|} \tag{5.13}$$

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

$$2 \cdot \frac{TP}{2 \cdot TP + FP + FN} = 2 \cdot \frac{\sum_{i=0}^{N-1} |P_i \cap L_i|}{2 \cdot \sum_{i=0}^{N-1} |P_i \cap L_i| + \sum_{i=0}^{N-1} |L_i - P_i| + \sum_{i=0}^{N-1} |P_i - L_i|} \tag{5.14}$$

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

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

Most multi-label metrics are non-convex and discrete, most algorithms turn to optimize alternative multi-label metrics. Recently, some researcher study the consistency of multi-label learning[16]. For example, it is ad hot to figure out if the loss of classifier converges to the Bayes loss while increasing training set size.

5.4 Learning Algorithms

Algorithms play key role in the literature of machine learning, so there is no exception in multi-label learning. The capability of representation is an important criterion to evaluate the performance of a algorithm. Otherwise, there are some related criteria to be consider. Firstly, broad spectrum is considered. If the algorithm can cover an range of algorithmic design strategies with unique characteristics. Secondly, it is reasonable to

evaluate the impact which the algorithm poses on the multi-label learning settings. Last, the algorithm complexity is a critical factor if the algorithm can be used in practice.

5.4.1 Problem Transformation Methods

5.4.1.1 Binary Relevance

Binary Relevance is a elementary algorithm which decomposes multi-label learning problem into a set of independent binary classification problems in which each problem harmonizes a label of the q labels in the set $L = y_1, y_2, ..., y_q$. The approach initially transforms multi-label dataset into q binary datasets $D_{y_i}(i = 1, 2, ..., q)$, 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 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 dataset, a set of q binary classifiers, say $H_i(E)(i = 1, 2, ..., q)$, has been built up using the respective training dataset D_{y_i}.

$$H = \{C_{y_i}(x, y_i) \rightarrow y_i' \in \{0, 1\} | y \in L, i = 1, 2, ..., q\} \tag{5.15}$$

To classify a new multi-label sample, Binary Relevance outputs the collection of labels which are predicted with positive value by the independent binary classifiers.

Binary Relevance 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 $f(|X|, |D|)$, Binary Relevance has complexity $O(qxf(|X|, |D|))$. With limit number of q, Binary Relevance can be regarded as a simple and effective method.

One of the disadvantages of Binary Relevance is the limitation of label relationship information. Given that all labels are independent, Binary Relevance discards the information among labels.

5.4.1.2 Classifier Chains

To improve the performance of Binary Relevance, classifier chains has been introduced in multi-label classification literature[12]. The algorithm transform multi-label learning problems into a chain of binary classifiers based on label dependence.

For a set of q labels L, Classifier Chains model learns q classifiers as Binary Relevance does. However, it links all classifiers through feature space. Given that there are a set of samples $D(x, y_i)(i = 1, 2, ..., q)$ where y_i is a subset of labels L and x is domain of dataset. We transform the dataset into q datasets in which the j-th sample is based on datasets and previous labels

$$H = \{C_{y_i}(x, y_1, y_2, ..., y_{i-1}) \rightarrow y_i' \in \{0, 1\} | y \in L, i = 1, 2, ..., q\} \tag{5.16}$$

Thus it forms a train of binary classifier $C_1, C_2, ..., C_q$. 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 $y_j, j = 1, 2, ..., i - 1$. The learning process starts from C_1 and follows the china step by step. Therefore, C_1 determines $p(y_1|x)$ and each classifier, say $C_2, ..., C_q$, determines $p(y_i|x_i, y_1, ..., y_{i-1})$.

The Classifier Chains method propagates label information through classifiers, and post classifier takes into account previous predictions. This can overcome issues of ignorant label correlation in some degree. At the same time, the method keep benefits of Binary Relevance including computation efficiency and memory efficiency. The computational complexity of Classifier Chains can be close to Binary Relevance depending on the size of labels and complexity of elemental classifiers C_i. As stated in previous, the complexity of Binary Relevance is $O(qxf(|X|, |D|))$ where $f(|X|, |D|))$ is the complexity of elemental classifier. With extra computation introduced by previous labels, the complexity of Classifier Chains is $O(qxf(|X| + q, |D|))$, while the complexity will be worse in case $q \gg |X|$.

5.4.2 Algorithm Adaptation Methods

5.4.2.1 Multi-label k-Nearest Neighbor(ML-kNN)

The algorithm adapts k-nearest neighbor techniques to propose multi-label data and utilize maximum a posteriori (MAP) to make prediction through reasoning embodied labelling information among neighbour [17].

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 neighbour of x in training set, we can compute a membership counting vector which represents the number of neighbours of x owning l-th label.

$$C_x(i) = \sum_{a \in N(x)} y_a(i), i \in Y \tag{5.17}$$

To test a new sample t, the algorithm sorts out its category in training dataset by kNN $N(t)$. Let H_1^l denote that t has label l and H_0^l denote that t do not have label l. There are j samples have label l, say $E_j^l(j \in 0, 1, ..., k)$. Then the category vector y_t is determined via the MAP principle:

$$y_t(l) = \arg\max_{b \in \{0,1\}} P(H_b^l | E^l_{C_t(i)}) l \in Y \tag{5.18}$$

With the Bayesian rule, 5.18 can be represented as

$$y_t(l) = \arg\max_{b \in \{0,1\}} P(H_b^l) P(E^l_{C_t(i)} | H_b^l) l \in Y \tag{5.19}$$

As stated in 5.19, the output of test sample t is $y_t(l)$. It can be computed via prior probability $P(H_b^l) l \in Y, b \in \{0, 1\}$ and the posterior probability $P(E_j^l | H_b^l)(j \in \{0, 1, ..., k\})$.

5.4.2.2 Collective Multi-label Classifier(CML)

Supposed that labels are highly interdependent in some scenarios, CML explores multi-label conditional random field (CRF) classification models which learn parameters for each pair of labels directly[15].

In dataset (x, Y), each sample has a corresponding random variables representation (x, y) in which $y = (y_1, y_2, ..., y_q) y_i \in \{-1, 1\}$ is binary label vector. If the sample contains j-th label, the j-th element of y is 1, otherwise -1. Then the aim of multi-label learning 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 maximizing $H_p(x, y)$. The fact is expressed with constrains on expectation of function $f(x, y)$. The expected value can be estimated from training dataset

$$F_k = \frac{1}{m} \sum_{(x,y) \in D} f_k(x, y) \tag{5.20}$$

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

$$p(y|x) = \frac{1}{Z(x)} exp(\sum_{k \in K} \lambda_k \cdot f_k(x, y)) \tag{5.21}$$

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:

$$y_x = \arg\max_y p(y|x) \tag{5.22}$$

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 pruning strategies. CML is a second-order approach which considers correlations between every label pair constrains in k_2.

Chapter 6

Multi-lable Classification Methodology

6.1 Artificial Dataset

To demonstrate the approach, the dataset, named color detector, is created artificially. To each image, it has three labels for representing colour, *red*, *green* and *blue*. If an image is red hue, it has a label for *red* and 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 6.1: Colour Wheel Diagram

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.

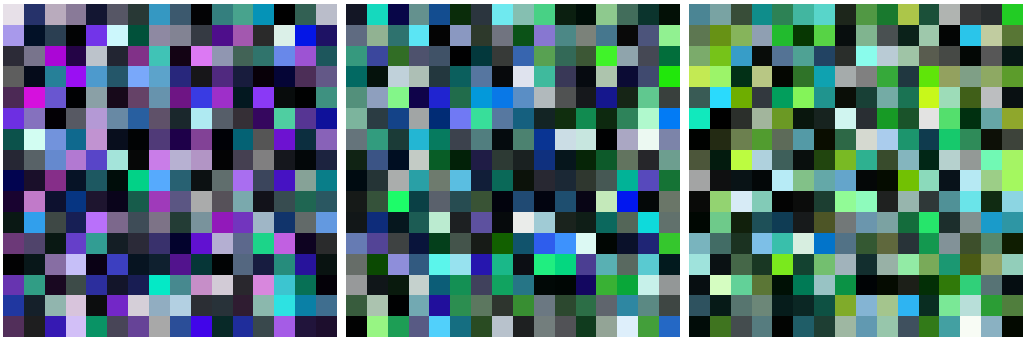
6.1.1 Image Generation

The RGB and HSV coordinate systems represent a geometric shape in the color space. Therefore, the distance among colors contains little meaning. However, corresponding distances make some intuitive sense and enable conversion possible between RGB coordinates and HSV coordinates.

To generate dataset artificially, we create $16x16$ size images and each image has 256 pixels. For each pixel, we generate a random floating point number h in the range $[0.0, 1.0)$ and use it as value for Hue via formulation.

$$H = h + r * 0.4 - 0.2(r \in [0.0, 1.0)) \tag{6.1}$$

Where r is another random floating point number. Then we generate 2 random floating point numbers as Saturation(S) and Value(V) values. We transform HSV values to RGB values and time 255 for each pixel. Repeating the steps, we can get a 256 pixels image. Because we get a RGB image via converting a HSV image, we compute RGB label value basing on previous h.



(a) L:-1 -1 1 (b) L:-1 1 1 (c) L:-1 1 -1

FIGURE 6.2: Multi-label Samples. Three labels mean red, green and blue seperately.

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

6.2 Artificial Neural Networks

Artificial Neural networks are a good way to learn a nonlinear function which can be used to map samples to multi labels. At the first layer of a neural networks, neurons

take raw dataset while the neurons in the last layer produce outputs. Among the first and the last layers, the middle layers are called hidden layers because they do not connect to external world. The 3-layer neural networks can represent any bounded degree polynomial under certain conditions[18]. The weights of neural networks 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 network's prediction for each sample with actual target value.

To adapt classical neural networks, which handle simple-label classification, to classify multi-label samples, we need to modify two factors. One is to design a new error function which is fitting the characteristics of multi-label samples instead of single-label ones. The other is to find a moderate metric according to new designed error function.

6.2.1 Network Architecture

Define $\mathcal{X} = \mathbb{R}^d$ as the sample space and $\mathcal{Y} = 1, 2, ..., Q$ as the set of output labels. Training dataset is composed of m multi-label samples, such as $(x_1, Y_1), ((x_2, Y_2), ..., ((x_m, Y_m)$, 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 sample. A neural network can be constructed as figure6.3 to learn a model from training samples.

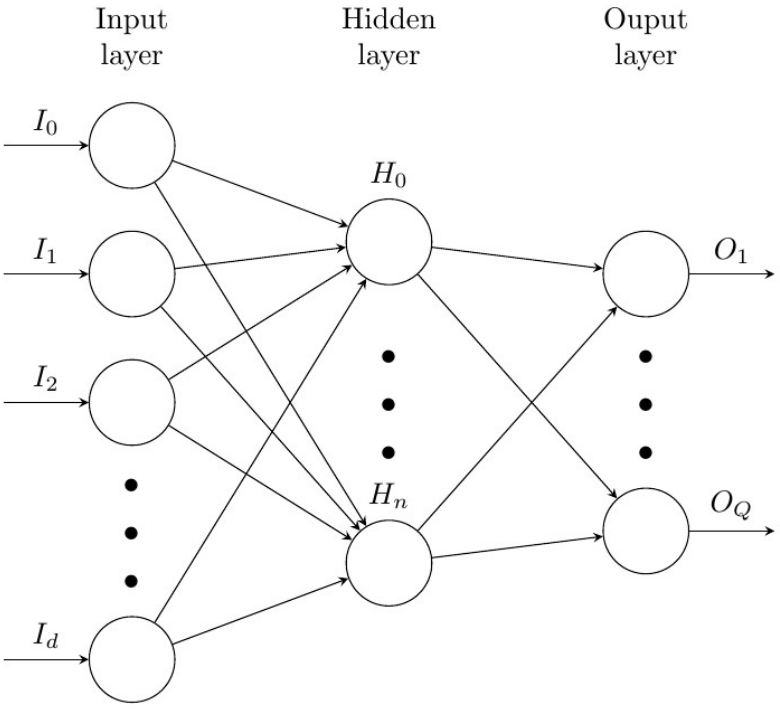


FIGURE 6.3: Network Topology For Multi-lable classification

The network has d input neurons which corresponds to a d-dimensional feature vector, while last Q neurons represent a combination of output labels. There is a hidden layer in 6.3 which owns n hidden neurons. Input layer is fully connected with hidden layer and the same connecting method between hidden layer and output layer. So there are dxn weights $(W_{ih}, 1 \leq i \leq d, 1 \leq h \leq n)$ between input layer and hidden layer and nxq weights $(W_{ho}, 1 \leq h \leq n, 1 \leq o \leq q)$ between hidden layer and output layer. 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 model as:

$$E = \sum_{i=1}^{m} E_i \tag{6.2}$$

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

$$E_i = \sum_{j=1}^{Q} (c_j^i - d_j^i)^2 \tag{6.3}$$

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

Various learning algorithms can be used to achieve a model based on training dataset. Backpropagation(BP) algorithm is deployed to learn from errors. However, original BP algorithm could be improper in multi-label learning because the error function6.3 neglects the correlations among labels of a sample. In original BP algorithm, the error function6.3 limits on individual label discrimination, such that a specific label $j \in mathcalY$ belongs to sample x_i or not. It should be taken into consider that labels in Y_i is more important than those outside of Y_i. A new global error function is defined as:

$$E = \sum_{i=1}^{m} E_i = \sum_{i=1}^{m} \frac{1}{|Y_i||\hat{Y}_i|} \sum_{(k,l) \in Y_i \times \hat{Y}_i} exp(-(c_k^i - c_l^i)) \tag{6.4}$$

In error function6.4, i-th errors on the i-th multi-label training sample (x_i, Y_i) have been accumulated. $\hat{Y}_i$ is complementary set of Y_i in $\mathcal{Y}$ and $|\cdot|$ computes the cardinality of a set. Specifically, $c_k^i - c_l^i$ represents the difference between the output labels of network on a label which belongs to sample x_i and a label which does not belong to the same sample. The error function6.4 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 of which one

belongs to the sample and the other does not belong to it. The sum is normalized by the numbers of all pairs, $|Y_i||\hat{Y}_i|$. Then, the correlations between pair labels are computed. In the other words, labels in Y_i should get larger output value than labels in $\hat{Y}_i$.

As previous statements, error function6.4 calculates the difference between output labels of which some belong to a sample and others do not belong to it. The task of learning is minimizing error function6.4 via enlarging output values of labels belonging to the training samples and diminishing output values of labels not belonging to it. If training dataset can cover the distribution of whole sample space, neural network model can learn it through minimizing error function by feeding training dataset.

6.2.2 Error Function

The basic goal of regression problems is to figure out the conditional distribution of the output labels, given 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, given the input samples. Except for sum-of-squares error function, there are some more approximate error functions which can be considered.

The central goal in training neural network is to model the hidden generator of the samples instead of memorizing 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 characterization of the generator 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 input data,

$$p(x, t) = p(t|x)p(x) \tag{6.5}$$

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

$$p(x) = \int p(t, x)dt \tag{6.6}$$

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

$$L = \prod_n \prod p(t^n|x^n)p(x^n) \tag{6.7}$$

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.

$$E = -lnL = -\sum_{n} lnp(t^n|x^n) - \sum_{n} lnp(x^n) \tag{6.8}$$

where E is notated as error function. The task of feed-forward neural network 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 6.8 to

$$E = -lnL = -\sum_{n} lnp(t^n|x^n) + C \tag{6.9}$$

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.

6.2.3 Cross Entropy

The Mean Square Error is a common risk metric comparable to the predicted value of the squared error loss. It is simple to implementation and non-negative. However, it has the disadvantage of heavily weighting outliers[19].

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

$$D_{pq}(p(x), q(x)) = \sum_{x} q(x) ln \frac{q(x)}{p(x)} \tag{6.10}$$

where the relative entropy is not a true metric because 6.10 could not be symmetric in the interchange $p \leftrightarrow q$ and probably not satisfy the triangle inequality.

Cross entropy loss, also called logistic regression loss, is an alternative method of estimating probability distributions. 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, $x_1, x_2, ...$, with weights $w_1, w_2, ...$, and a bias, b, the output, for example two classes, can be represented as the weighted sum of the input data.

$$a = f(x) = \sum_{i} w_i x_i + b \tag{6.11}$$

Then the cross entropy cost function is

$$C = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)] \tag{6.12}$$

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

The cross entropy can act as a cost function because of two properties. The first one is that C is non-negative, because the items of logarithms are in the range $[0, 1]$ and there is a minus sign at the beginning of the sum. The second one, if the prediction output is close to the actual value for all training data, the loss value, C, will be close to 0. Given that $y = 0$ and prediction value $a \approx 0$, the first item in 6.12 is 0 while the second item is $-\ln(1-a) \approx 0$. The similar situation occurs for situation, $y = 1$ and $a \approx 1$. Therefore, the value of cost function will be small given that 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. They are basic properties for a cost function. Although the quadratic cost 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 cost function can be computed with the respect to the weights. Applying the chain rule twice on 6.12

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_{x}\left(\frac{y}{f(x)} - \frac{(1-y)}{1-f(x)}\right)\frac{\partial f}{\partial w_j} \tag{6.13}$$

$$= -\frac{1}{n}\sum_{x}\left(\frac{y}{f(x)} - \frac{(1-y)}{1-f(x)}\right)f'(x)x_j. \tag{6.14}$$

where the second equation can be represented as

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_{x}\frac{f'(x)x_j}{f(x)(1-f(x))}(f(z)-y) \tag{6.15}$$

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

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_{x}x_j(f(x)-y) \tag{6.16}$$

The equation shows that rate of learning weights is controlled by $f(x) - y$. The larger the error, 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.

$$\frac{\partial C}{\partial b} = \frac{1}{n}\sum_{x}(f(x) - y) \tag{6.17}$$

6.2.4 Training and Testing

Following previous process, gradient descent is used to minimize the global error function with backpropagation.

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

$$c_k = f(netc_k + \theta_k) \tag{6.18}$$

where θ_k is the bias units of k layer, $f()$ is the activation function of the output neurons which is a *tanh* function2.11 in this project. $netc_k$ is the input data of the layer:

$$netc_k = \sum_{s=1}^{M} b_s w_{sk} \tag{6.19}$$

where w_{sk} is the weights connecting layer s and k, while M is the number of neurons in the hidden layer.

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

$$d_k = -\frac{\partial E}{\partial netc_k} \tag{6.20}$$

combining with 6.18, we can get

$$d_k = -\frac{\partial E_i}{\partial c_j}\frac{\partial c_j}{\partial netc_k} = -\frac{\partial E_i}{\partial c_j} f'(netc_k + \theta_k) \tag{6.21}$$

With considering global error function 6.4

$$\frac{\partial E_i}{\partial c_j} = \begin{cases} -\frac{1}{|Y_i||\hat{Y}_i|}\sum_{l\in\hat{Y}_i} exp(-(c_j - c_l)) & \text{if } j \in Y_i \\ \frac{1}{|Y_i||\hat{Y}_i|}\sum_{k\in Y_i} exp(-(c_k - c_j)) & \text{if } j \in \hat{Y}_i \end{cases} \tag{6.22}$$

with derivation of *tanh*, and substituting it with 6.21 and 6.22 we can get

$$d_k = \begin{cases} \big(-\frac{1}{|Y_i||\hat{Y}_i|}\sum_{l\in\hat{Y}_i} exp(-(c_j - c_l))\big)(1 + c_j)(1 - c_j) & \text{if } j \in Y_i \\ \big(\frac{1}{|Y_i||\hat{Y}_i|}\sum_{k\in Y_i} exp(-(c_k - c_j))\big)(1 + c_j)(1 - c_j) & \text{if } j \in \hat{Y}_i \end{cases} \tag{6.23}$$

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

$$e_s = -\frac{\partial E_i}{\partial netb_s} \tag{6.24}$$

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

$$e_s = -\frac{\partial E_i}{\partial b_s}\frac{\partial b_s}{\partial netb_s} = -\Big(\sum_{j=1}^{Q}\frac{\partial E_i}{\partial netc_j}\frac{\partial netc_j}{\partial b_s}\Big) f'(netb_s + \lambda_s) \tag{6.25}$$

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

$$e_s = \Big(\sum_{j=1}^{Q} d_j \times \frac{\partial(\sum_{s=1}^{M} b_s w_{sj})}{\partial b_s}\Big) f'(netb_s + \lambda_s) = \Big(\sum_{j=1}^{Q} d_j w_{sj}\Big) f'(netb_s + \lambda_s) \tag{6.26}$$

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

$$e_s = \Big(\sum_{j=1}^{Q} d_j w_{sj}\Big)(1 + b_s)(1 - b_s) \tag{6.27}$$

Stochastic gradient descent(SGD) method is used to approximate function.

$$\Delta w_{sj} = -\alpha\frac{\partial E_i}{\partial w_{sj}} = \alpha\frac{\partial E_i}{\partial netc_j}\frac{\partial netc_j}{\partial w_{sj}} = \alpha d_j b_s \tag{6.28}$$

$$\Delta v_{hs} = -\alpha\frac{\partial E_i}{\partial v_{hs}} = \alpha\frac{\partial E_i}{\partial netb_s}\frac{\partial netb_s}{\partial v_{hs}} = \alpha e_s a_h \tag{6.29}$$

while bias are updated according to

$$\Delta\theta_j = \alpha d_j \;\; \Delta\lambda_s = \alpha e_s \tag{6.30}$$

in previous Equations, α is learning rate in the range of [0.0 1.0].

Based on previous process, a learning algorithm has been set up with backpropagation. More ever, training samples are fed into neural network in each epoch. After all training samples $((x, Y)$ fed into network, weights and bias are updated through equations6.28 and 6.30. The training samples are fed into the network iteratively while global error value decreases. Finally, the error value converge to a minimum value.

In testing process, network predicts a sample which has a set of actual labels $c_j(j = 1, 2, ..., Q)$ by ranking the labels. Because the output values 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[20] is adopted to generalize the sets. $t(x)$

is a linear function, $t(x) = w^T \cdot c(x) + b$, where $c(x) = (c_1(x), c_2(x), ..., c_Q(x))$ is a Q-dimensional vector which represent j output labels of a sample. Learning the threshold function $t(x)$ follows the next steps. 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:

$$t(x_i) = \arg\max_t(|\{k|k \in Y_i, c_k^i \leq t\}| + |\{l|l \in \hat{Y}_i, c_l^i \geq t\}|) \tag{6.31}$$

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 Φ has dimensions $m \times (Q+1)$ in which i-th vector is $(c_1^i, c_2^2, ..., c_Q^i, 1)$, w' is $(Q+1)$ dimensional vector (w, b) and t is the m dimensional vector $(t(x_1), t(x_2), ..., t(x_m))$. 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 computation complexity of evaluating derivatives of the error function is linearly with neuron size of network. Three main components are composed of computation, feedforward process, backpropagation process and updating weights process. In feedforward process to compute b_i and c_j, the computation cost is mainly on evaluation the sums and activation function. In backpropagation process(6.23and6.27), the computation complexity of d_k and e_s is $O(Q)$. In updating weights process, the overall computational cost is $O(W)$ where W is the total number of weights .

6.3 Experiment

We use the identical hardware and software environment as described in weather classification project.

6.3.1 Dataset

We generate 1000 raw RGB images with size $16x16$ automatically. 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 the color attribute, the corresponding label is 1, otherwise -1.

6.3.2 Details of network

We build a network architecture from scratch up. First, we need do determine how many layers needs to construct a multilayer network. In [21], Lippmann has proved that two hidden layers are able to create classification regions of arbitrary desired shape. However, Kolmogorov [22] 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 connected hidden layer between input and output layers. Between two fully connected layers, we put a ReLU layer to apply the non-saturating

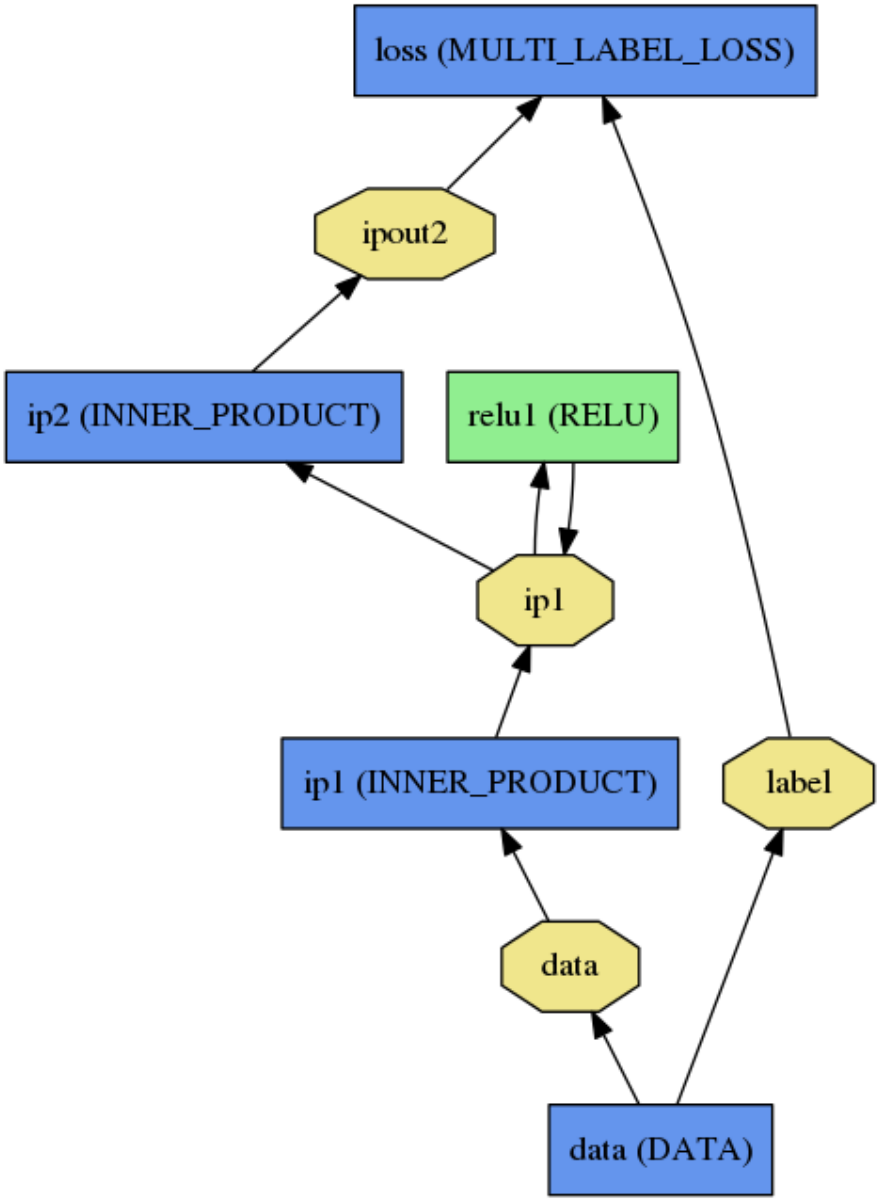


FIGURE 6.4: Network topology for multilable classification.

activation function $f(x) = max(0, x)$ which can help the decision function increase the non-linear properties.

Then we need to determine how many neurons need 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. network architecture
4. training algorithms
5. type of activation function
6. regularization

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

There are many rules of thumb to determine the number of hidden neurons [23] such as

1. The number should between the size of input layer and output layer.
2. The number should be $\frac{2}{3}$ of the sum of the input neurons plus the output neurons.
3. The number should be less than twice the size of the input layer.
4. Based on geometric pyramid rule, the number should 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 connecting method between different layers is fully connecting because we want to find which color label owned by each image in this case. The information of each color distributes evenly in each patch of images. The value of the weights and bias is initialized randomly. For weights, *xavier* algorithm is applied to determine the scale of initialization depending on the number of input and output neurons. The bias is simply

Neuron number	Sensitivity	Specificity	Harmonic Mean	Precission	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 6.1: Test results for different number of hidden neurons.

initialized as constant, default as 0. The batch size is set to 40 means that there are 40 images feeding into neural network each time.

In training process, the base learning rate is set to 0.0001. The learning rate of weights is equal to base learning rate while the learning rate of bias double the base learning rate. We set momentum with 0.9 to smooth the weight updating across iterations so that the learning process will be stable and fast. Weight decay is 0.0005.

Label-metric is used as evaluation metric. Each label will be counted separately and statistics will be calculated in total. The definitions are described in chapter 5.

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

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901.

[11] 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.

[12] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

[13] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *Dept. of Informatics, Aristotle University of Thessaloniki, Greece*, 2006.

[14] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Machine learning: ECML 2007*, pages 406–417. Springer, 2007.

[15] 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.

[16] Wei Gao and Zhi-Hua Zhou. On the consistency of multi-label learning. *Artificial Intelligence*, 199:22–44, 2013.

[17] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

[18] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *Information Theory, IEEE Transactions on*, 39(3):930–945, 1993.

[19] Sergio Bermejo and Joan Cabestany. Oriented principal component analysis for large margin classifiers. *Neural Networks*, 14(10):1447–1461, 2001.

[20] André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Advances in neural information processing systems*, pages 681–687, 2001.

[21] Richard P Lippmann. An introduction to computing with neural nets. *ASSP Magazine, IEEE*, 4(2):4–22, 1987.

[22] 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.

[23] Jeff Heaton. *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.