CHALMERS
UNIVERSITY OF TECHNOLOGY

Deep Learning for Human Fall Classification with Application to E-Healthcare

Master thesis in Biomedical Engineering Programme

WENJING CHEN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
EX101/2017

Master's thesis 2017: EX101/2017

Deep Learning for Human Fall Classification with Application to E-Healthcare

WENJING CHEN

Supervisor and Examiner: Prof. Irene Y.H. Gu

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2017

Deep Learning for Human Fall Classification with Application to E-Healthcare
WENJING CHEN

Supervisor and Examiner: Irene Gu, Department of Electrical Engineering

Master's Thesis 2017: EX101/2017
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46-(0)31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Deep Learning for Human Fall Classification with Application to E-Healthcare
WENJING CHEN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

With the increasing of computation power and amount of data, deep learning has achieved a great success in computer vision. Human activity recognition is one of the most important tasks in computer vision. Especially the classification of falling down and not falling down is of high importance in e-healthcare and assisted-living.

This thesis focuses on using deep learning for video classification on human falls. In the first part of the thesis, image classification using CNN is done on traffic sign images to study the network learning process. In the second part, which is the main part of the thesis, video classification is performed with three different network structures, namely, single-frame 2DCNN, multi-frame 2DCNN and RNN. Here single-frame 2DCNN is used as a baseline, while multi-frame 2DCNN and RNN are experimented for utilizing temporal information.

Results from this thesis showed a high test accuracy 98.83% on the GTSRB traffic sign dataset, and 85.42% on the multiple camera fall dataset.

Keywords: image classification, machine learning, convolutional neural network (CNN), video classification, recurrent neural network (RNN).

Acknowledgements

I want to thank Irene Y.H. Gu for her supervision and the department for providing the resources on doing this thesis. Special thanks to Chenjie Ge, for the help on processing multiple camera dataset and also for valuable discussion about deep learning. Finally, I owe my deepest gratitude to my family for supporting my study here in Sweden and their unconditional love all these years.

Wenjng Chen, Gothenburg, October 2017

Contents

1

Introduction

1.1 Background

With the development of high-performance computing system like Graphics Processing Units (GPU) and enormous available public data, deep learning has achieved a great success in the field of computer vision. It shows great power in image classification tasks using deep convolutional neural networks [1]. Implementing deep learning in other area of computer vision has drawn lots of attention. Activity classification is one of the fundamental tasks of computer vision. It has received tremendous interest recent years, largely due to its wide range real-world environment applications including video surveillance, behavior analysis, etc. This thesis focuses on using deep learning method to classify human activity using public available video datasets, especially human falling down activity. Besides, traffic sign classification is used as an example for image classification to test the deep learning method.

1.2 Related work

For traffic sign classification, the highest classification accuracy recorded by GTSRB official website is 99.46% using committee of convolutional neural networks (CNN) in 2012 [2]. In 2016, a new record of 99.67% was achieved by using hierarchical CNN [3]. The human performance on this dataset is 98.84%, which has been surpassed by CNN in 2012. As a result, there isn't much research work on this dataset in recent years.

For human action classification, lots of works have been done using deep learning in past few years. Unlike image classification, action video classification has not yet gained extraordinary performance compared with human. Work based on deep learning for video classification can be divided into mainly three categories: (1) two stream convolutional neural network, (2) 3D convolutional neural network, (3) recurrent neural network.

Two stream convolutional neural network. Two stream CNN uses both the raw frames and corresponding calculated optical flow images as input to 2D CNN. This method is first proposed by Simonyan et al. [4] in 2014. They achieved around 88.0% and 59.4% respectively on the most widely used datasets UCF101 and HMDB51. Based on this work [4], Feichtenhofer et al. [5] improved the accuracy to 92.5% and 65.4% by using CNN instead of softmax layer to fuse the spatial and temporal network in 2016. In the same year, Limin et al. reached 94.2% and 69.4% by adding RGB difference and warped optical flow on the original two stream CNN. Diba et al. [6] achieved by far the highest accuracy on both HMDB51 and UCF101 datasets, respectively 95.6% and 71.1%, using a bilinear model to aggregate the output of last convolutional layers of pre-trained networks.

3D convolutional neural network. The 3D CNN is first introduced by Ji et al. [7] in 2013. Karparthy et al. [8] used different kinds of method for time information fusion on large-scale dataset sports 1M in 2014 and acheived 63.3% on the UCF101 by transferring the trained model. Tran et al. [9] tried to learn spatiotemporal features with 3D CNN and got 85.2% accuracy on UCF101. Based on the works above, it seems that the trend is to incorporate optical flow snippets to achieve higher accuracy.

Recurrent neural network. RNN is known for dealing with temporal sequential information. Lots of studies has been done for action classification using RNN. Due to the enormous data size of videos, CNNs are always used before RNN to get a higher abstraction of the input video. Donahue et al. [10] proposed Long-term Recurrent Convolutional Networks (LRCNs) with 2D CNN for visual feature extraction and multi-layer LSTMs to fuse termporal information. They achieved 82.92% accuracy on UCF101. Ballas et al. proposed a new recurrent unit called Gated Recurrent Unit Recurrent Networks (GRUs) to replace LSTMs and introduced a new network architecture that applies an recurrent unit on the intermediate convolutional layers. They achieved 85.7% accuracy on the UCF101 dataset.

2

Theory

2.1 Deep learning

Deep learning is a new branch of machine learning based on using deep artificial neural networks to learn a specific pattern for mapping input to output. Artificial neural network is primarily inspired by modeling the neurons of the real neural networks inside our brain. Figure 2.1 shows the structure of a real neuron. The dendrites of the neuron are connected to three axons of former neurons. When these axons are activated, they release neurotransmitters to activate the dendrites. If the intensity of the activation is strong enough, an electrical impulse will be generated and travel along the axon to the next neuron.

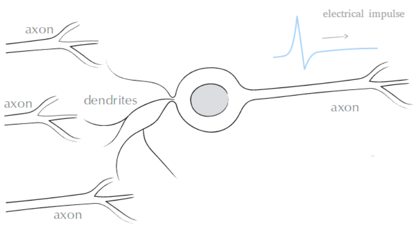


Figure 2.1: A real neuron

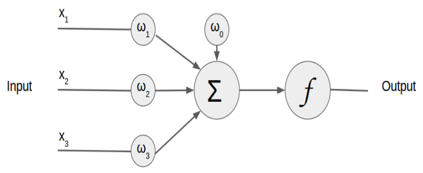


Figure 2.2: A simple model of a neuron

Figure 2.2 shows a simple mathematical model of the neuron. There are three inputs, x_1, x_2, x_3, simulating the axons from former neurons. The inputs are multiplied by different weights simulating dendrites with different strength of influence. The weighted inputs and a bias ω_0 are then added together to be sent to the activation function f. The details of activation functions will be discussed in section 2.4. It is important to note that this mathematical model is a coarse model and practically used deep learning method has diverged from imitating real neural systems.

In deep learning field, neuron usually refers to the mathematical model of a biological neuron and artificial neural network is simply referred as neural network. The same naming convention is used in this thesis. Neural network is composed of a large number of neurons with a layered structure. The layer is a fully-connected layer, where neurons within adjacent layers are fully connected and neurons within a layer are disconnected. An example of an artificial neural network is demonstrated below 2.3.

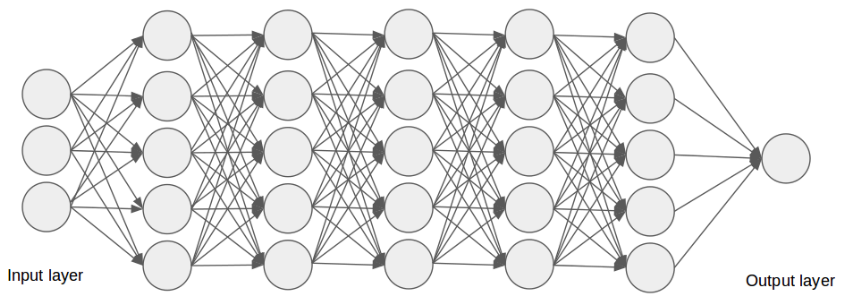


Figure 2.3: A 6-layer neural network with three inputs, five hidden layers of 5 neurons each and one output layer

The concept of artificial neural network have been existing for more than 40 years. With the introduction of GPU and enormous public data, artificial neural network has become an effective tool for modeling especially with deep layers. Deep learning has a huge impact on computer vision. Nowadays, image classification done by machines even outperforms humans in some scenarios via the use of deep learning.

2.2 Deep learning libraries

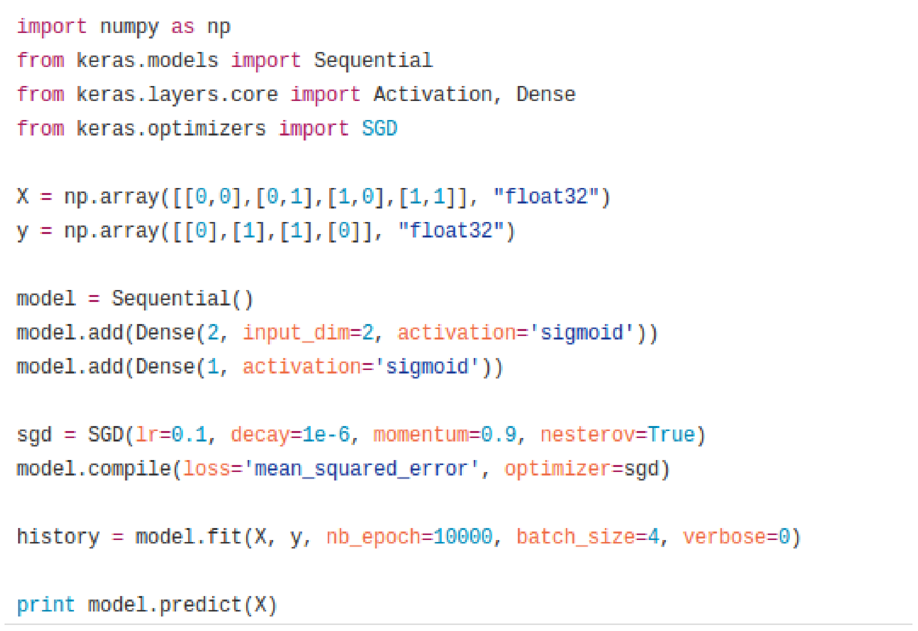
Due to extraordinary performance of deep learning, many works has been done for developing deep learning. They can be roughly divided into to direction. One focuses on exploring new algorithms for deep learning, including building new network structure, find the mathematical support for deep learning method. And another group of people pay attention to develop the open-source tools for implementing deep learning. There are five most famous deep learning libraries, namely, Theano, Caffe, Tensorflow, Torch and MXNet.

Theano

Theano is one of the oldest deep learning libraries. It was built by Yoshua Bengio's group at University of Montreal. The programming language for Theano is Python. It uses symbolic computation. It was preferred at first because its advantages for computing derivatives which is a main concern in deep learning algorithm for back propagation, which will be introduced in later section. Most old deep learning tutorial is based on Theano. As a result, most people who are interested in deep learning will get a start with Theano.

There are two famous high-level wrappers built on Theano, one is lasagne and one is Keras. Lasagne is a veteran library and has been widely used in past time. It has good documentation for beginners who want to enter deep learning field. The combination of Theano and lasagne is quite popular before. So the Lasagne model zoo has lots of pretrained common architectures. However, as time goes by, more and more frameworks appeared, and they lag behind now. Theano is a stable library and still capable for deep learning. But because of its low support for multi-GPU, the speed for training network on Theano will be comparablely slow.

Keras is a really popular library for now and is highly recommended for beginners. It is a very high-level library that based on Theano or Tensorflow. It only requires fairly few lines of code to build a neural network. And the code is very clear to follow. Figure 2.4 shows a simple example in keras for training a neural network and appling the trained model for test.



```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Activation, Dense
from keras.optimizers import SGD

X = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
y = np.array([[0],[1],[1],[0]], "float32")

model = Sequential()
model.add(Dense(2, input_dim=2, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))

sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)

history = model.fit(X, y, nb_epoch=10000, batch_size=4, verbose=0)

print model.predict(X)
```

Figure 2.4: A simple deep learning code using Keras

Caffe

Caffe is built by U.C. Berkeley. The source code is written in C++ with support for Python and MATLAB bindings. Normally Python and MATLAB is sufficient for programming, but the knowldege of C++ and CUDA is needed for new change. It is considered as one of the most veteran frameworks and focuses mainly on computer vision. So it is preferred in computer vision research area. As a result, it has many trained network resources and lots of released codes from papers in computer vision use Caffe as deep learning library.

Torch

Torch is developed by NYU and IDIAP. It is used by Facebook Research which makes it well-known. However, the programming language used in Torch is Lua, which is a less common language compared with Python. In the beginning of 2017, pyTorch is released as a complementary deep learning frameworks for Python users. The advantage of using lua is that LuaJIT is an easy and fast scripting language and has easy interface to C. Torch is embeddable, with ports to iOS, Android and FPGA backends, which means it is easy to implement in other platforms other than computer.

TensorFlow

TensorFlow comes from googel. It uses computation graphs which makes it similar to Theano. Unlike Theano, tensorFlow has support for multi-GPU and multi-node training. TensorFlow supports Python and C++. TensorFlow itself is very low level which makes it hard to use. Keras is also possible for TensorFlow and its default back environment is TensorFlow. TensorFlow also has good visualization using TensorBoard.

MXNet

MXNet is one of the most languages-supported libraries. It has support for Python, R, C++, Julia, Scala, Matlab and Javascript. It also has auto-differentiation property as Theano. Amazon web services use MXNet as their deep learning framework because of its enormous horizaontal scaling capabilities. It is considered as a developing framework with large potential.

Apart from the aforementioned four deep learning frameworks, there are also many other less popular frameworks. Some of them has stopped updating, for example, pylearn2 and decaf. Some of them are less popular but are still on the developing way, for example, pyTorch and Cognitive Toolkit. Because of fast information propagation and high demand for implementing deep learning, the deep learning frameworks are changing rapidly. The best strategy for choosing the suitable framework is to try one at first and keep an eye on others.

2.3 Training neural networks

Training a neural networks can be divided into three major parts: setting up the network architecture, setting up the data and the loss, learning and evaluation. The network architecture will be discussed detailedly in next chapter. This chapter is going to focus on the last two parts.

2.3.1 Setting up the data

Data is used to train a neural network. The data for training a neural network is always labeled or partly labeled. For example, to train a neural network for classifying images of dogs and cats, the image should be labeled about whether it contains a dog or a cat.

Dataset split

The original whole dataset should be split into three sets: training, validation, and test set. For most public available datasets, the test set is usually predefined. This act ensures an identical test environment for comparing trained models' performances. Ideally the test set should be used only once to avoid overestimating the performances. It is used at the end when the model training is finished. Since the test set can't be used to tune parameters of the model, a good choice is to create a fake test set instead. This is done randomly splitting the original training dataset into training set and validation set. Usually 70-90% of the training dataset is assigned to training set. This split ratio depends on the quantity of the data and the number of parameters to be trained. Usually, more data and more parameter will lead to a lower split ratio.

Data preprocessing

There are three commonly used preprocessing methods. Assume the data X is of size $N * D$ where N is the number of data and D is the dimensionality, and the preprocessed data is X'. x_i is the value of the i-th data in one unspecified dimension and ${x_i}'$ is the correspondingly preprocessed one.

Mean subtraction is the easiest and most commonly used one. First the mean value of the dataset in each dimension will be calculated and then it will be subtracted across every individual in the corresponding dimension. After the operation, the data will be zero-centered. The mean subtraction can be described by the equation below:

$${x_i}' = x_i - \frac{\sum x_i}{N}, i = 1, 2, ..., N \tag{2.1}$$

Normalization is to adjust the range of different dimensions into a same scale. There are two common normalization ways. One makes the preprocessed data have zero mean and unit variance. This normalization method can be described by

$$x_i{}' = \frac{x_i - \overline{x}}{s}, \overline{x} = \frac{\sum x_i}{N}, s^2 = \frac{1}{N-1} \sum (x_i - \overline{x})^2, i = 1, 2, ..., N \tag{2.2}$$

The standard deviation is a little bit computationally expensive. In some cases, the standard deviation can be roughly approximated by

$$s \approx \frac{X_{max} - X_{min}}{2} \tag{2.3}$$

X_{max} and X_{min} here means the maximum and minimum value in each dimension. Another method is to linearly bring all values of each dimension into the range -1 to 1, wrote as

$$x_i{}' = \frac{x_i - X_{min}}{X_{max} - X_{min}} - 1 \tag{2.4}$$

The last preprocessing method is Principal Component Analysis (PCA) and whitening. PCA is to convert possibly correlated original data into linearly uncorrelated data. This is done by rotate the data into the eigenbasis of the data covariance matrix after it has been zero-centered. In this way, the covariance matrix becomes diagonal. Whitening is to normalize the scale of each dimension by dividing the eigenvalues based on the result of PCA. After whitening, the covariance matrix becomes an identity matrix.

Due to complexity and effectiveness, PCA and whitening is not commonly used as a preprocessing method for training neural networks. It will be considered when the data is high dimensional and need a dimensionality reduction. Mean subtraction and normalization is much more commonly preferred, especially mean subtraction.

Data augmentation

Data is one of the most important part of deep learning. In some cases, scarcity of data can impede the learning process. And this leads to the concept data augmentation. There are two simple way to augment the dataset, one is by adding noise and another is to apply label-preserving transformations. In practice, transformation method is more common. Popular transformation methods include rotating, flipping, clipping, PCA dimensionality reduction, etc.

In summary, setting up the data is to split the data into three parts and then preprocess the three set independently.

2.3.2 Setting up the loss

After setting up the data, the data be sent through a predefined network and the output of the network will be compared with the true labels to generate the loss.

The loss is an average of individual loss which can be represented by $L = \frac{1}{N}\sum_i L_i$ where N is the number of training data. In this section, the form of L_i will be discussed. Here we assume that y_i the true label for one training data and $y_i{}'$ is the prediction from the network.

L2 norm and L1 norm

One of the easiest way to measure the difference between true value and predicted value is direct subtraction. To avoid the effect of sign, the result of direct subtraction is followed with square or absolute operation. The loss function using the square is called L2 norm, defined as

$$L_i = (y_i - y_i{}')^2 \tag{2.5}$$

The loss function using the absolute operation is called L1 norm, defined as

$$L_i = |y_i - y_i{}'| \tag{2.6}$$

Multiclass support vector machine and hinge loss

Support vector machine (SVM) is a classification method which maximizes the margin between different classes. The multiclass SVM loss is designed to ensure that the output score of the network for a correct class is higher than scores for the incorrect classes at a predefined margin δ. Here function $f(x_i, W)$ is used to represent the network, where W represents the parameters to be learned and x_i is the input. $t_j = f(x_i, W)_j$ is the score for the j-th class. The multiclass SVM loss can be formulated by

$$L_i = \sum_{j \neq y_i} max(0, t_j - t_{y_i} + \delta) \tag{2.7}$$

The function $max(0, -)$ is called hinge loss. So the multiclass SVM loss is also called multiclass hinge loss.

Softmax classifier and cross-entropy loss

Softmax classifier is a generalization of a binary logistic regression classifier. The softmax function is formulated as

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.8}$$

It squashes a K-dimensional vector $\mathbf{z}$ with arbitrary value into a K-dimensional vector $\delta(\mathbf{z})$ with values range in (0,1) and sum to 1. In this way, the output from softmax function can be interpreted as normalized class probabilities.

Cross-entropy is a terminology comes from information theory. The entropy of a distribution P measures the average minimum number of bits needed to encode data from P with encoding scheme based on P. The cross-entropy of a distribution p with respect to distribution Q measures the average minimum number of bits needed to encode data from p with encoding scheme based on q, which is defined as

$$H(p,q) = -\sum_x p(x)\log q(x) \tag{2.9}$$

For cross-entropy loss, the distribution p is the true distribution and the distribution q is the estimated distribution. In classification case, p can be formulated as $[0,...,1,...,0]$ which contains a single one at the y_i-th position, and q is the estimated class probabilities vector from the softmax function $q = e^{f_{y_i}} / \sum_j e^{f_j}$. Thereby cross-entropy loss can be formed by

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right), p_{y_i} = 1 \tag{2.10}$$

Regularization

Regularization is a process that introduces additional information to compensate conditions neglected by the loss function. This can be achieved by directly adding a weighted regularization penalty $R(W)$ to form a general loss function

$$L = \frac{1}{N}\sum_i L_i + \lambda R(\omega) \tag{2.11}$$

where λ is a term indicating the importance of the regularization, called regularization strength. Usually regularization is used to prevent overfitting.

There are four common regularization methods. The first and the most common form of regularization is L2 regularization, defined as

$$R(\omega) = \sum_k \omega_k^2 \tag{2.12}$$

The expression above suppress the magnitude of all parameters. The use of L2 regularization will penalize peaky weight vectors and tend to diffuse weight vectors, which intuitively promote the use of all inputs rather than part of its input.

The second form of regularization is L1 regularization, defined as

$$R(\omega) = \sum_k |\omega_k| \tag{2.13}$$

The use of L1 regularization will result in a sparsely distributed weight vectors, which means that only a few important inputs are used and some noisy inputs are neglected. The L1 regularization can be combined with L2 regularization in the form of $\lambda_1|\omega| + \lambda_2\omega^2$.

The third form of regularization is max norm regularization. It is not performed as adding regularization to the loss function as normal. In fact, it enforces a constraint of the magnitude of the weight vectors immediately after each gradient descent update. The constraint is to limit the norm of weight vector at each layer to be upper bounded by a fixed constant c, as shown in $||\omega||^2 \leq c$. The value of c is usually set on orders of 3 or 4.

The fourth form of regularization is the most commonly used one called dropout [11]. It is an extremely effective way to avoid overfitting. The scheme is simply by disabling some random neurons in one layer with a fixed portion during training. Disable a neuron means that the activation of this neuron set to zero during forward pass and the weight is not updated during backpropogation. Note that dropout is not applied during test, so a scaling after dropout at train time is important to make sure the output from train time and test time has the same scale.

Apart from the regularization methods mentioned above, there are some other less popular methods. Some methods regularize the bias parameters. In practice, bias regularization is not powerful which likely due to comparable fewer bias terms compared to all the weights. Few methods regularize different layers to different amounts. In practice, L2 regularization and dropout are the most commonly used regularization methods.

2.3.3 Learning the weights

The loss function describes how far the prediction is from the true label. So the best model parameters should minimize the loss function. Here for neural network the parameters are called weights.

Weight initialization

Before the optimization of the weights, the weights need to be initialized. If the data is properly normalized, it is reasonable to assume that the distribution of weights will be centered at zero. It is intuitive to set all weights to zero. However it is not applicable in practice. The update of weights is depended on their derivatives, which will be discussed later. If all the weights have same value, they will also get same derivatives and never diverge. Therefore, it is better to initialize the weights with small random numbers. Then the initialized weights will be close to zero but wouldn't have same update.

There are three commonly used weight initialization methods. Assume m is the number of neurons in previous layer and n is the number of neurons in next layer. The first method is to draw random number from a distribution with mean zero and standard deviation $\sqrt{1/m}$ [12]. Because the variance of randomly initialized neuron grows with the number of inputs. m here is to rescale the variance to one. The second method is the same process with a slight different standard deviation $\sqrt{1/(m+n)}$ [13]. It adds the consideration of backpropagation and uses the average as a compromise. This method is commonly referred as Glorot initialization. The third method is also the same with standard deviation $\sqrt{2/m}$ [14]. The third initialization method is specific for ReLU neurons.

Gradient descent

Mathematically, gradient represents the direction of the steepest ascend. To decrease the loss function, it is intuitive to move the weights towards the negative direction of the gradient. There are two ways to calculate the gradient. One is to calculate the numerical gradient using centered difference formula $[f(x+h)-f(x-h)]/2h$. h is set to a small value, for example, 1e-5. Another method is to compute analytic gradient using Calculus. The first method is easy to perform but computationally inefficient. The numerical gradient is approximate and h needs to be small enough to get a relatively accurate result. The analytic gradient is based on mathematical formula derived by human which means it has exact value but error-prone. In practice, analytic gradient is used for gradient calculation, and numerical gradient is only used to compare with the analytic result to make sure the gradient is correct.

The update of parameters in the negative direction of the gradient is called gradient descent, described by

$$\theta^{(k+1)} = \theta^{(k)} - \mu\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta L_i \tag{2.14}$$

θ here represents the parameters to be updated and k represents the number of version. μ is the step size which is also called learning rate. i means the index of a training data. Depend on the value of n, there are three gradient descent methods. The first one is batch gradient descent which calculates the gradients based on the sum loss of all the training data. The second one is mini-batch gradient descent which calculates the gradients over batches of the training data. The third one is stochastic gradient descent which only use a single training sample for gradient descent.

Computing the analytic gradients is an arduous task for neural networks especially deep neural networks. That is why the terminology backpropagation is introduced. Backpropagation is a way to compute gradients expression by using chain rule recursively. Assume the neural network can be interpreted as a complex function, backpropagation provides a way to easily calculate the gradients without an symbolically full gradient function using stacked computation. Neuron from 2.2 is used as an example to illustrate this concept, shown in 2.5. Here sigmoid $f(t) = 1/(1+e^{-t})$ is chosen as the activation function. The green equations above the line represent the forward propagation and the red equations below the line represent the back propagation.

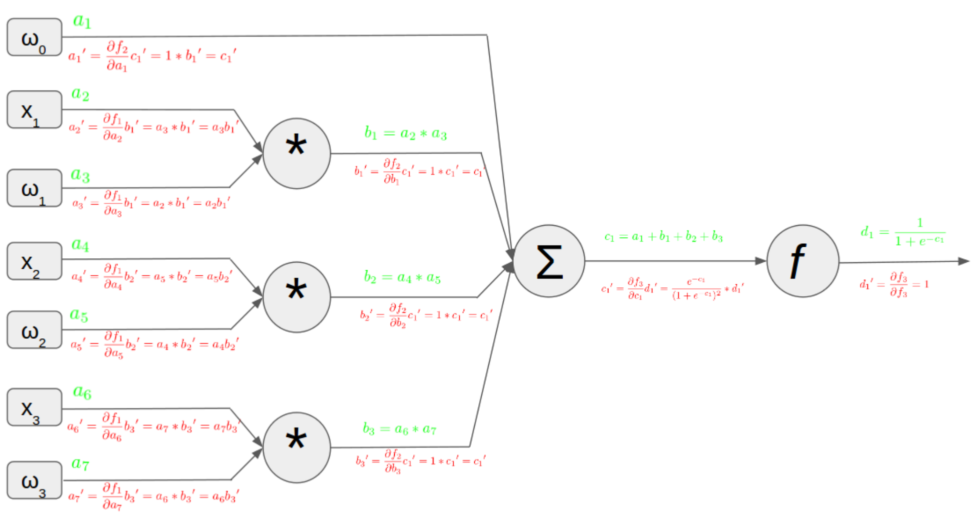

ω0
x1
ω1
x2
ω2
x3
ω3
a_1
$a_1' = \frac{\partial f_2}{\partial a_1} c_1' = 1 * b_1' = c_1'$
a_2
$a_2' = \frac{\partial f_1}{\partial a_2} b_1' = a_3 * b_1' = a_3 b_1'$
a_3
$a_3' = \frac{\partial f_1}{\partial a_3} b_1' = a_2 * b_1' = a_2 b_1'$
a_4
$a_4' = \frac{\partial f_1}{\partial a_4} b_2' = a_5 * b_2' = a_5 b_2'$
a_5
$a_5' = \frac{\partial f_1}{\partial a_5} b_2' = a_4 * b_2' = a_4 b_2'$
a_6
$a_6' = \frac{\partial f_1}{\partial a_6} b_3' = a_7 * b_3' = a_7 b_3'$
a_7
$a_7' = \frac{\partial f_1}{\partial a_7} b_3' = a_6 * b_3' = a_6 b_3'$
$b_1 = a_2 * a_3$
$b_1' = \frac{\partial f_2}{\partial b_1} c_1' = 1 * c_1' = c_1'$
$b_2 = a_4 * a_5$
$b_2' = \frac{\partial f_2}{\partial b_2} c_1' = 1 * c_1' = c_1'$
$b_3 = a_6 * a_7$
$b_3' = \frac{\partial f_2}{\partial b_3} c_1' = 1 * c_1' = c_1'$
$c_1 = a_1 + b_1 + b_2 + b_3$
$c_1' = \frac{\partial f_3}{\partial c_1} d_1' = \frac{e^{-c_1}}{(1+e^{-c_1})^2} * d_1'$
$d_1 = \frac{1}{1+e^{-c_1}}$
$d_1' = \frac{\partial f_3}{\partial f_3} = 1$
f

Figure 2.5: An example of backpropogation

Learning control

The simplest form of update is to move the weights along the negative gradient direction as in equation 2.14. However, there are many other updating rule. In stochastic gradient descent, there might be a bad update because of a bad example. An effective way to solve this problem is by using momentum update, defined as

$$\begin{aligned} v^{(k+1)} &= \gamma v^{(k)} - \mu \nabla_\theta L \\ \theta^{(k+1)} &= \theta^{(k)} + v^{(k+1)} \end{aligned} \tag{2.15}$$

where γ represents the momentum and v represents the velocity. Nesterov momentum is a more popular update form by slightly changing the momentum update. It replaces θ in $\nabla_\theta L$ of equation 2.15 with a predicted new θ which can be expressed by $\theta_{new} = \theta + \gamma v$. The update function for Nesterov momentum can be formalized in a similar way as 2.15 by

$$\begin{aligned} v^{(k+1)} &= \gamma v^{(k)} - \mu \nabla_\theta L \\ \theta^{(k+1)} &= \theta^{(k)} - \gamma v^{(k)} + (1+\gamma) v^{(k+1)} \end{aligned} \tag{2.16}$$

,

There are three other common methods which update adaptively according to the gradient. The first one is called Adagrad proposed by Duchi et al [15], which can be illustrated by

$$\begin{aligned} v &= v + d\theta^2 \\ \theta^{(k+1)} &= \theta^{(k)} - \mu \frac{d\theta}{\sqrt{v} + \delta} \end{aligned} \tag{2.17}$$

where v is the sum of all squared gradients, δ is a small value used to avoid division by zero. This method allows that weights with high gradients have relative small learning rate while weights with low gradients have relative high learning rate. The second method is called RMSprop with a small adjust on Adagrad [16], formulated by

$$\begin{aligned} v &= \eta v + (1-\eta)d\theta^2 \\ \theta^{(k+1)} &= \theta^{(k)} - \mu\frac{d\theta}{\sqrt{v}+\delta} \end{aligned} \tag{2.18}$$

where η is decay rate, usually set as 0.9, 0.99, or 0.999. The third method is similar to RMSprop but modified with momentum, which is called Adam [17].

$$\begin{aligned} m &= \beta_1 m + (1-\beta_1)d\theta \\ v &= \beta_2 v + (1-\beta_2)d\theta^2 \\ \theta^{(k+1)} &= \theta^{(k)} - \mu\frac{m}{\sqrt{(v)}+\delta} \end{aligned} \tag{2.19}$$

The recommended hyperparameter values are $\delta = 1 * 10^{-8}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. In practice, Adam is the most preferred among the three adaptive methods.

Beside all the methods mentioned above, learning rate is also an important hyperparameter which largely affect the learning process. Higher learning rate will lead to a faster update but is prone to a wrong direction especially during stochastic descent. However, a lower learning rate takes much longer to converge to the minimum loss. In order to balance the effect of high and low learning rate, a good strategy is to vary the learning rate during the training process. Use a higher learning rate at first to accelerate the training process and later decrease it to avoid missing the minimum. This is called learning rate decay. There are three common types of learning rate decay strategies: step decay, exponential decay and 1/t decay [18]. Step decay is to reduce the learning rate by multiplying a fixed constant every few epochs. For example, multiply 0.5 every 10 epochs. Practically, the learning rate is fixed until the validation error stops decreasing and then reduced by a factor of 0.5. Exponential decay is formed by

$$\mu = \mu_0 e^{-kt} \tag{2.20}$$

where μ_0 is the initial learning rate, k represents learning rate decay, and t represents the number of epochs. The $1/t$ decay could be illustrated by

$$\mu = \mu_0/(1+kt) \tag{2.21}$$

2.4 Network structures

In last section, it is shown that how to set up input and output of the neural network model, how to update and evaluate the model. This section will discuss different forms of the model.

2.4.1 Convolutional neural networks

Convolutional neural network (CNN) was first proposed by Yann LeCun in 1998 [19]. Due to computation limits, it didn't show its effectiveness and became popular at first. In 2012, Krizhevsky et al won the ImageNet LSVRC-2010 contest with top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry [1]. This is a remarkable achievements made CNN and it soon propagated worldwide. Besides this contest, CNNs outperform other methods in nearly all competitions related to image classification. In fact, CNNs are less generalized version of the artificial neural networks mentioned in 2.1. A convolutional neural network is usually a stack of convolutional layer (CONV), pooling layer (POOL) and fully connected (FC) layer.

Convolutional layer

Convolutional layer is the core part of convolutional neural networks. Unlike FC layer, CONV layer is sparsely connected. Figure 2.6 is an example of local connectivity compared with figure 2.3. In this way, each neuron in the CONV layer

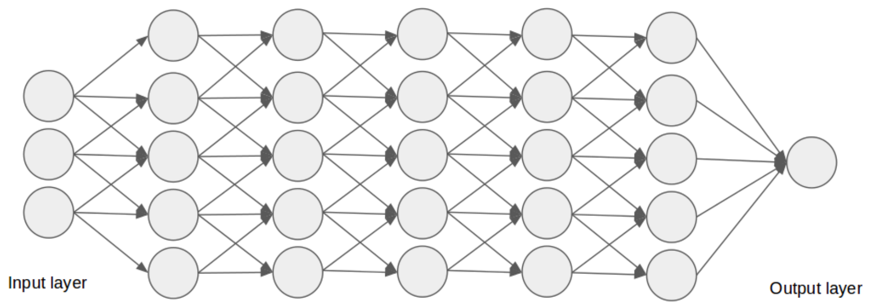


Figure 2.6: Local connectivity

has its own local receptive field and is only sensitive to this certain local field. The local connectivity is the first property of CONV layer. Assume the local connected neuron is used to extract a certain feature as the real neuron does in the brain, if the weights of the neuron is applicable in one certain position, it should also be useful in another position. This implies the second property of CONV layer, called weights sharing, which means all the neurons in one layer share weights. Based on these two properties, the forward pass of the CONV layer can be interpreted as convolution between weights and input layer. This explains why the name has convolution and why the weights matrix is always referred as a filter or a kernel.

Compared with fully connected layer, CONV layer has much fewer parameters due to local connectivity and weights sharing. However, while dramatically decreasing the complexity, the amount information extracted by such layer is also decreasing. This introduces the third property of CONV layer, depth. Because of weights sharing, more than one set of weights are needed to extract sufficient features.

Pooling layer

POOL layer is a downsampling method. It can reduce the amount of parameters and help control overfitting. The most commonly used pooling is max pooling with size 2*2 and stride 2, which can discard 75% of the activations. The operation is quiet simple. As the case mentioned above, it is to keep the maximum value of the 2*2 region while discard other 3 numbers and the head to a next 2*2 region with stride 2. Apart from max pooling, there are other kinds of POOL layer, for example, average pooling. But they are not used very often, because max pooling has slight better performance in practice.

Activation layer

Activation layer composes of neurons performing activation function. Activation function, or called non-linearity, takes a single number and output a single number based on a certain fixed mathematical operation. There are three commonly used activation functions. The first one is sigmoid, formulated by $\delta(t) = 1/(1+e^{-t})$. It squashes a input number into range between [0,1]. It was used frequently before because it has a reasonable interpretation for the firing of real neuron. However, it is not a good choice in artificial neural network. It is not zero centered which will lead to zig-zagging gradient update [18]. The second method is tanh, which is a zero-centered version of sigmoid. It simply squashes a input number into range between [-1,1] by using $tanh(t) = 2\delta(2t) - 1$, where $\delta(t)$ is the sigmoid function. Tanh non-linearity solve the problem of zero-center but it has the saturate problem as sigmoid. When the input value is too large or too small, the gradient will be very small as shown in figure 2.7. This will decrease the learning process. The third method

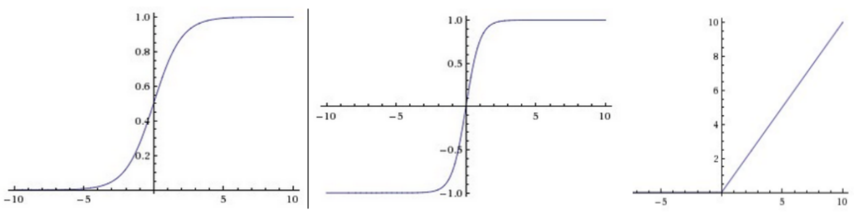


Figure 2.7: Activation functions. Left:sigmoid, Middle:tanh, Right: ReLU

is the most commonly used one, called ReLU. The operation is relatively simple, represented by $f(x) = max(0, x)$. It is thresholded at zero allowing numbers larger than zero pass and kill numbers smaller than zero. It is proved to accelerate the convergence of stochastic gradient descent compared to the sigmoid/tanh functions [1].

The most common CNN architecture can be represented by :

INPUT -> [[CONV->RELU]*N->POOL?]*M->[FC->RELU]*K->FC (2.22)

where * represent repetition, and N, M, K indicate the number of repetition [18]. POOL? here indicates that pooling layer is optional. Usually K is smaller than 3 and N is is smaller than 4.

Existing Networks

The first convolutional network is proposed by LeCun in 1998 called LeNet [19]. It was the first successful implementation of CNN in image classification. The architecture of LeNet is shown in figure 2.8. It is a quiet simple network with basic structure of CNN network, i.e. [CONV-POOL-CONV-POOL-CONV-FC]. Although the network is proposed pretty early, the hardware at that time cannot utilize the full power of the CNN. And the network is pretty shallow with only 7 layers. Alexnet

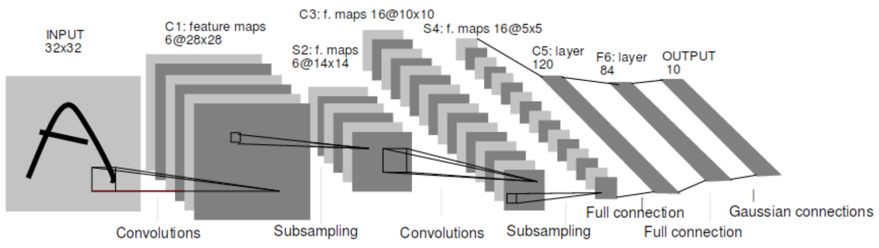


Figure 2.8: Architecture of LeNet [19]

is the first most famous network structure which won the first prize in ImageNet LSVRC-2010 contest [1]. This is the most influential event in the history of CNN. It soon propagate the effectiveness of CNN. It uses heavy data augmentation. It is the first network use ReLU as activation layer. In the same competition, VGGnet is

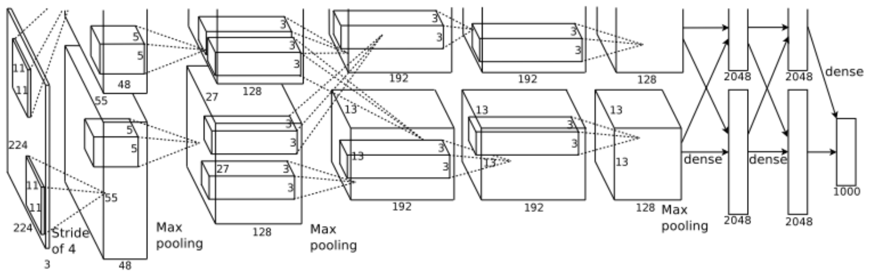
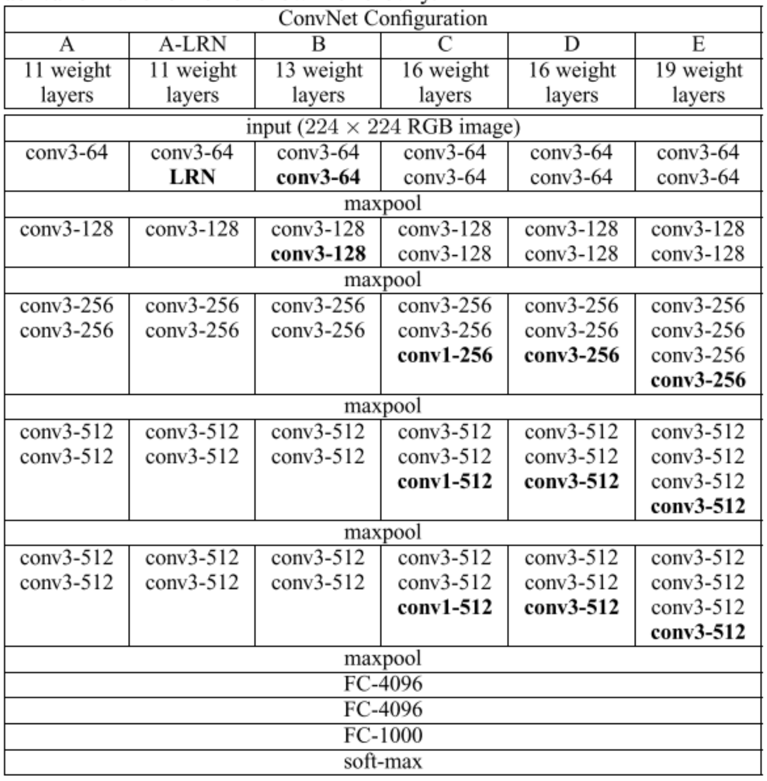


Figure 2.9: Architecture of AlexNet [1]

proposed in ILSVRC-2013 with top 5 error 11.2%. And the Although VGGnet did't win in that competition, it still received lots of attention. The architecture used in VGGnet is still the common structure of CNN with much deeper layers. It proved that deeper network has better performance. Because it provides the best result using conventional convolutional neural network, the model of VGGnet is popular and widely used. GoogLeNet is the winner of ILSVRC-2014 with top 5 error 6.7%



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.10: Architecture of VGGnet [20]

while the top-5 error achieved by Alexnet is 16.4%. The most important part of GoogLeNet is that it introduced inception module. As a result, it has much less parameters, which is 12 times less compared with Alexnet. The fully connected layer is discarded in GoogLeNet due to large amounts of needed parameters.

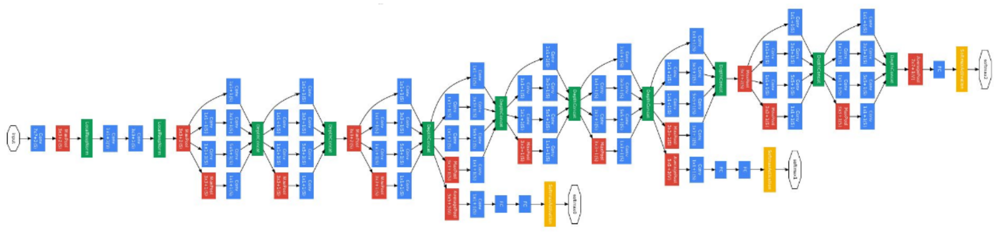


Figure 2.11: Architecture of GoogLeNet [21]

The winner of ILSVRC-2015 is Resnet with top 5 error 3.6%. It won the first place in all tracks including imagenet classification, imagenet detection, imagenet localization, COCO detection and COCO segmentaton. Compared with other network, Resnet has very deep layers, 152 layers compared with 22 layers in GoogLenet. It

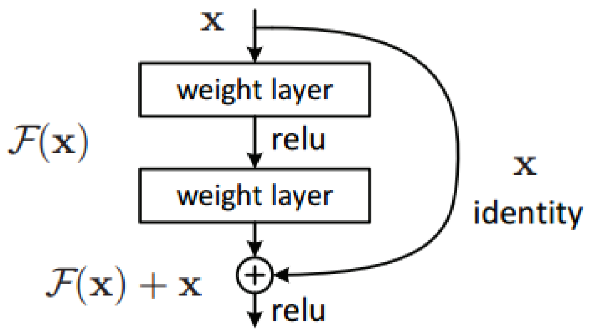


Figure 2.12: A building block of residual learning [22]

took 2-3 weeks to train Resnet with 8 GPU. However, it runs fairly fast during test, faster than VGG which has 19 layers.

3D convolutional neural network

Unlike 2D CNN, filters in 3D CNN have 3 dimensions rather than 2 dimensions. A pictorial illustration of 2D and 3D convolution operations is shown in figure 2.13. The filter size of 2D convolution is notated as $f_1 * f_2$ and the filter size of 3D convolution is notated as $f_1 * f_2 * f_3$. In figure 2.13, the depth of the filter f_3 is set as 3 which results in 2 output frames with 5 input frames without zero padding.

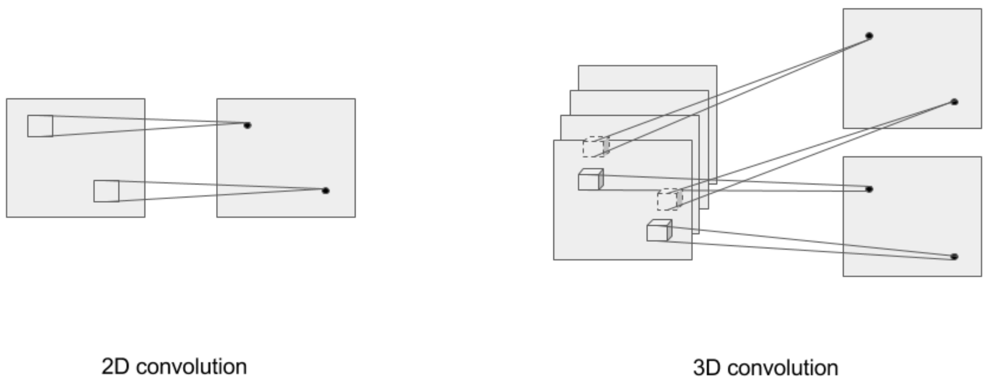


Figure 2.13: 2D and 3D convolution comparison in single channel

The above illustration is for input with only one channel. A illustration of difference between 2D and 3D convolution is shown in figure 2.14. The number of channels in the first layer is n_1 and the number of channels in the second layer is n_2. The convolved result of all channels in previous layer will be added up and construct a single channel in the next layer. As a result, the number of parameters for 2D convolution is $n_2 * (f_1 * f_2 * n_1 + 1)$. 1 here is a intercept term. And the number of paremeters for 3D convolution is $n_2 * (f_1 * f_2 * f_3 * n_1 + 1)$. It is worth noting that the number of parameters is independent from the input frame size and length.

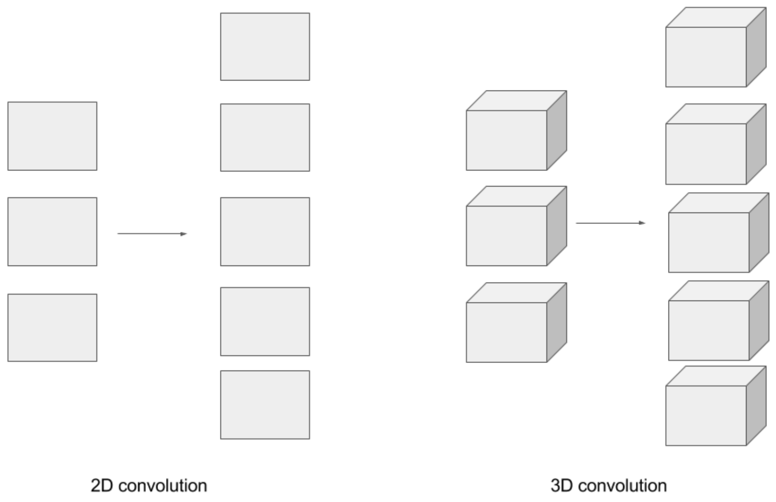


Figure 2.14: 2D and 3D convolution comparison in multiple channels

2.4.2 Recurrent neural networks

Previous neural network including convolutional neural network is designed for current input, which means it won't utilize information from previous inputs. Also, it could not deal with data without a fixed input size and a fixed output size. This becomes a problem while dealing with sequential or temporal data. That is the reason why Recurrent Neural Network (RNN) is needed.

RNN could be roughly explained as combining input vector and state vector (from previous hidden layer) into hidden layer resulting in a new state vector. This is illustrated by equation 2.23. h_t represents the state vector, x_t represents input vector, and f_W is fixed function with parameters W. The function for final output depends on what kind of RNN to be used. In this way, the output of RNN is not only determined by current input but also the entire history of inputs fed in.

$$h_t = f_W(h_{t-1}, x_t) \tag{2.23}$$

Figure 2.15 would be a good explanation for the difference between CNN and RNN. The left part of figure 2.15 shows a simple network structure of a one-layer CNN, where the red dot represents input, green dot represents hidden layer, and blue dot represents output. The right part of figure 2.15 shows different forms of RNN, namely, sequence output, sequence input, sequence input and output, synced sequence input and output. Just as in CNN we could have a stack of hidden layer

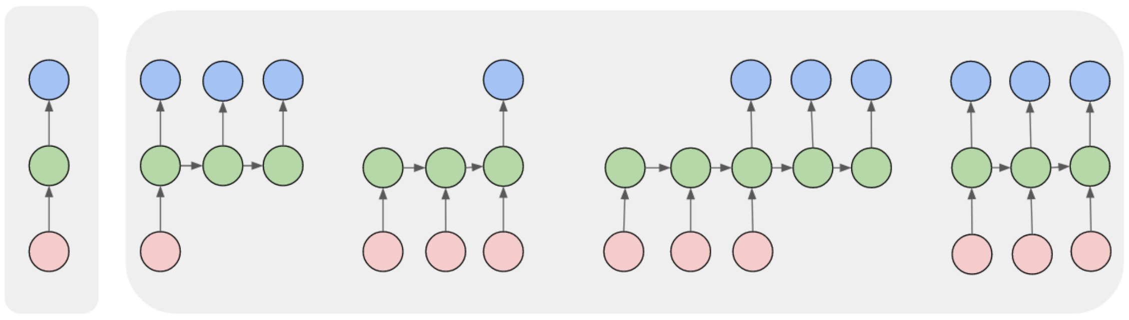


Figure 2.15: Illustration of CNN (left) and RNN (right)

(green dot), it is the same case in RNN, which means we could have another hidden layer (green dot) before output (blue layer). It is also important to keep in mind that hidden layer in the same horizontal line is identical, which mean all these green dots have exact same weights, the only difference would be the input and output. Consequently, there is no constraint on the length of the input and output sequence. This is illustrated in figure 2.16. The y-axis represents the depth of network and the x-axis represents the input time sequence.

Vanilla recurrent neural network

Vanilla RNN only contains a single hidden vector h. The f_W mentioned above would be tanh function here.

$$\begin{aligned} h_t &= tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= W_{hy}h_t \end{aligned} \tag{2.24}$$

However, vanilla RNN does not work well in practice. First, it has problem using long term information. Second, it has severe gradient vanishing problem. Almost all exciting results based on RNNs achieved by following more complex structures.

Long short term memory

Long Short Term Memory (LSTM) networks is a widely used RNN capable of learning long-term dependencies. It is first introduced in 1997 by Hochreiter and Schmidhuber [23] and refined by a lot of work later. Briefly, LSTMs add one more variable called cell state c between RNN blocks, which allows the network learn to choose

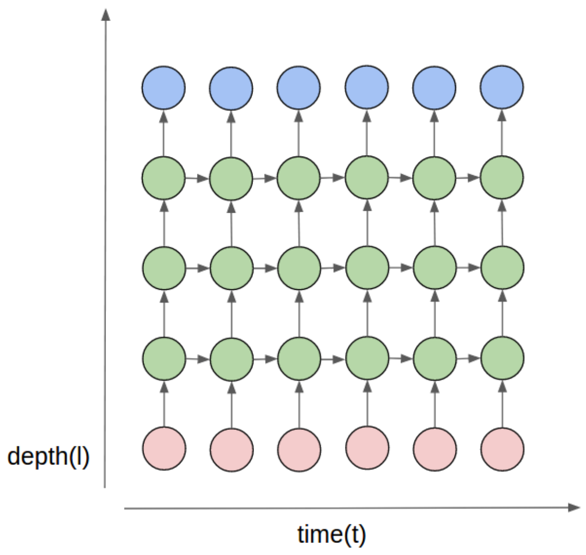


Figure 2.16: Multi-layer synced sequence input and output RNN

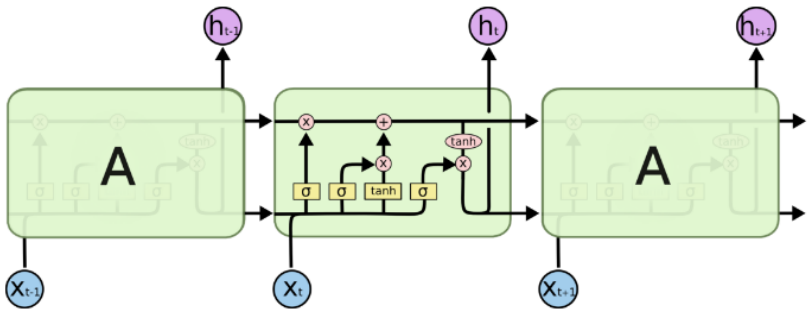


Figure 2.17: A diagram of RNN built with LSTM [24]

what to memorize and what to forget.

$$\begin{gathered}
\begin{matrix} i \\ f \\ o \\ g \end{matrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \\
c_t^l = f \odot c_{t-1}^l + i \odot g \\
h_t^l = o \odot tanh(c_t^l)
\end{gathered} \tag{2.25}$$

Here i is the input gate, f is the forget gate, o is the output gate, g is the block input. The W_l is the weight we are going to learn. W_l will determine the four gates, and the four gates will determine what will be passed. The cell state of time t and layer l c_t^l is updated by forgetting part of the old cell state c_{t-1}^l and add new information from input. The output h_t^l is based on the new cell state and filtered by output gate o. A more clear illustration is shown in figure 2.17.

Gated recurrent unit

Gated Recurrent Unit (GRU) is developed based on LSTM. It merges the input gate and forget gate in LSTM into a single gate. It also merges the cell state and hidden state, which means there will not be variables except hidden state and input flowing between RNN blocks. The equations are shown below.

$$\begin{aligned}
z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \\
r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \\
h'_t &= tanh(W \cdot [r_t * h_{t-1}, x_t]) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * h'_t
\end{aligned} \tag{2.26}$$

2.4.3 Multiple networks

From previous sections, it is mainly focused on one network structure with different layers. There is also another form of using neural network which combines different networks together. Intuitively, the networks could be connected in two ways. One is by concatenating all together and another one is by parallel networks. In deep learning field, the first concatenated one is called hierarchical network and the second parallel one is called ensemble network.

Hierarchical network uses the result from the former network as the input for the latter network. For example, in traffic sign classification case, there are in total 43 signs and these 43 signs can be classified into 6 categories, namely, speed limit signs, other prohibitory signs, derestriction signs, mandatory signs, danger signs and unique signs. So the scheme for hierarchical network is to classify the 6 categories using the first network and then classify the final sign independently in each category with 6 networks. This is a coarse to fine strategy.

Ensemble network is a much more popular strategy compared with hierarchical network. Usually, the ensemble network could enhance around 2% accuracy compared with one network. The scheme is to train multiple independent networks and average their predictions at test time. The performance is increasing with the number of ensemble networks, but the improvements will decrease at the same time. Normally, 5 networks are used in ensemble network. There are three common methods for forming an ensemble.

The first method could be to use same model with different initializations. It is not recommended because the only variation is the initialization which might lead to similar trained networks. The second method is to use the models with better performance. Since during model training, there will be lots of networks created when tuning different hyperparameters. It is pretty straightforward to use an ensemble of the top models. The third method is to use a single network trained at different time. This is the easiest way to do, since it doesn't require any more computation. The drawback is that it is lack of variety, but it actually works in practice. In fact, all the aforementioned methods could be combined together to find the best ensemble network.

3

Methods

Deep learning is an empirical study without explicit mathematical basis. Some of the results could be explained by theory, but most of them haven't been covered yet. So in practice, when it comes to use deep learning for solving problems, the way to do it is usually by setting a commonly used structure and then tuning different hyperparameters.

3.1 Traffic sign recognition

The work performed in this part mainly focuses on using deep learning for traffic sign classification in the form of images. The goal of this part is to study the process of network training and further the influence of different hyperparameters on the learning process. Networks will be trained using training set and then evaluated by using validation set. The network with highest validation accuracy will be selected and tested using test set.

Since image classification is not the main part of this thesis and network structure for video classification has large difference from image classification, this thesis will focus more on data preprocessing and hyperparameter optimization. As a result, only one network structure will be applied here.

3.1.1 Set up the data and loss

The first thing to do is to set up the input data. There are two parts, image preprocessing and data set split.

Image preprocessing

Since normally the input data size is predefined for fixed neural networks, all the input images needed to be resized to a fixed size. The size of images from GTSRB varies from 25 to 243. Because the average image size for training set is 50 for both side, all the images from GTSRB will be resized to $(48, 48)$ using bilinear interpolation. Besides the mandatory size adjustment, there are three further voluntary preprocessing methods as mentioned in Section 2.3. In practice, mean subtraction is the most popular one. One reason is that images normally range from 0 to 255 which means normalization is not necessary since it has already had similar scale. Another reason is that mean subtraction is the one with least computation. In this thesis, the images are simply preprocessed by division of 128 and mean subtraction to roughly scale the intensity range to $[-1, 1]$. The mean subtraction is done independently for three color channels. For example, the mean pixel intensity of green channel images will be calculated first and then all the green channel images will subtract this mean value.

Data set split

GTSRB provides training dataset and test dataset. Here the test dataset is reserved for test set, and the training dataset is randomly splitted into training set and validation set with ratio 7:3. It is worth noting that the preprocessing mention above is done after dataset split to ensure the independence between training set and validation set which will be contaminated by mean subtraction.

Loss

The loss function used in this part is cross-entropy. No regularization is added to the loss. The reason is that this combination gives better result in classification task in practical. As as result, it is also the most popular loss setting in classification task. Since the traffic sign part in this thesis focus on learning control, the loss function is hence fixed with cross-entropy without further experiments.

Evaluation metric

The evaluation metric used to judge the performance of trained model is accuracy. The accuracy is defined by:

$$accuracy = \frac{number\ of\ correctly\ predicted\ labels}{total\ number\ of\ training\ samples} \tag{3.1}$$

3.1.2 Network structure

Among all the possible neural network structures, convolutional neural network excels for its extraordinary effectiveness in classification tasks. As stated in Section 2.4, the network used in this part applys the most common pattern with stacks of a

Layer	CONV	FC	Total
Number of parameters	19872	305920	325792

Table 3.1: The number of used parameters in the network

few CONV-RELU-MaxPOOL layers following with FC-RELU:

INPUT -> [CONV->RELU->MaxPOOL]*3->FC->RELU->FC (3.2)

In the first CONV layer, 32 filters with size 5*5 are used. Zero padding is used in all CONV layers to ensure input and output have same height and width. Besides, all CONV layers use stride 1. The size of the input is 48*48*3 and the output after the first CONV layer is 48*48*32. The number of parameters needed in the first CONV layer is 2400 (32*5*5*3). The second and third CONV layer both have 32 filters with size 3*3. For POOL layer, max pooling is used with filters of size 2*2 and stride 2. The first FC layer is of size 256. The second FC layer is of size 43 becuase the number of traffic sign classes is 43. Before each FC layer, a dropout of ratio 0.5 is used during training process. After the last FC layer, the output is sent into softmax classifier which combined with cross-entropy as the final loss function. All in all, the number of weights used in the network can be shown by Table 3.1. The detail of the calculation can be found in Table 3.2. It is obvious that most of the weights come from FC layer.

The filter size is all set as 3*3. The reason is that smaller filter has fewer parameters and is able to reach the same receptive field with stacks of CONV layer. In practice, deeper network always works better than shallow one [20]. However, this network structure cannot extend as deep as we want. The input image size is 48*48, which means maximum 4 POOL layers are allowed. In addition, deeper network also means more parameters. As a trade-off between the computation limit and benefits from deep network, three CONV layers are stacked in this network.

3.1.3 Hyperparameter optimization

Apart from data preprocessing, loss and network structure, there are many other important hyperparameters. Some of them are very sensitive and have strong influence on the result while some not. Some hyperparameters have high freedom while some only have several limited options. There are so many hyperparameters, it is unlikely to finetune every hyperparameter due to computation limits. So in this thesis, the hyperparameters which have value proved to be currently optimized both theoretically and practically will be fixated. Apart from the hyperparameters related to data preprocessing, loss and network structure, the remaining hyperparameters are weight initialization criterion, weight update criterion, batch size, learning rate, number of epochs, etc.

Weight initialization is one of the fixated hyperparameters. As long as weights are

Layer	Kernel	Stride	Input size	Output size
CONV1	5*5*32	1	3*48*48	32*48*48
ReLU1	-	-	32*48*48	32*48*48
MaxPOOL1	2*2	2	32*48*48	32*24*24
CONV2	3*3*32	1	32*24*24	32*24*24
ReLU2	-	-	32*24*24	32*24*24
MaxPOOL2	2*2	2	32*24*24	32*12*12
CONV3	3*3*32	1	32*12*12	32*12*12
ReLU3	-	-	32*12*12	32*12*12
MaxPOOL3	2*2	2	32*12*12	32*6*6
FC1	1*256	-	32*6*6	256
ReLU4	-	-	256	256
FC2	1*43	-	256	43
Softmax	-	-	43	43

Table 3.2: The network architecture

not initialized by zero, the difference made by weight initialization will be gradually erased by weight update. In this part, Glorot initialization is used with uniform distribution. In detail, the weights are drawn from a uniform distribution with zero mean and standard deviation $\sqrt{1/(m+n)}$, where m is the number of neurons in previous layer and n is the number of neurons in next layer.

The weights update method used here is Nesterov momentum update combined with step learning rate decay. This method can be clearly illustrated by (2.21) and (2.16). Momentum, learning rate, and batch size are the hyperparameters that will be fine tuned. Learning rate is the factor has the biggest influence. So learning rate will be tuned first. In addition, because the existence of momentum and learning rate decay scheme, a higher initial rate is preferred as long as the learning process doesn't fall into a wrong direction.

Once the aforementioned three hyperparameters are tuned, the final training process will be performed with step decay. The scheme for step decay here is to reduce the learning rate by a half when validation error stops improving. While the learning rate is smaller than 0.000001 and validation error is not decreasing, the learning process will be stopped. The trained network will be evaluated on test set.

Notice that all mentioned hyperparameters to be tuned are involved with learning control. This is because that traffic sign classification part is only a prior experiment for action classification part. Consider there is a big difference in data preprocessing and network structure part between image classification and video classification, learning control part consequently becomes the main part for traffic sign classification.

3.2 Human fall classification

The difference between image and video classification is that video has additional temporal information. This thesis focuses on experimenting different network architectures for video classification. All these network architecture will be empirically evaluated to find the best network architecture.

3.2.1 Set up the data and loss

In this section, video sampling strategy, image processing, loss function definition, etc are presented.

Video preprocessing

Videos are composed of frames. Frames, which are images, means basic image preprocessing is applicable. For simplicity, only mean subtraction and rescale are used here. The mean values to be subtracted are 123.68, 116.779 and 103.939 respectively for three RGB channels. These mean values are adopted from ImageNet dataset not only for convenience but also for considering use of pretrained model from ImageNet. Apart from mean subtraction, another image preprocessing method, image resize, is needed if the image size is not identical. After mean subtraction, the input will multiply 1/255 which makes the input value sit in range -1.0 to 1.0.

The consecutive frames in video are highly redundant. It would be inefficient to feed all the frames into the network. A common way is to randomly downsample frames in each training epoch. This in one hand ensures smaller input to the network and in another hand make fully use of all the available information.

In training process, the input frames are randomly selected from the whole video. For single frame 2DCNN, one frame is randomly selected in each epoch. For other cases, which are mutiple frames 2DCNN and RNN, the video is divided into N_c clips with equal length and one frame in each clip will be randomly extracted. The number of segments N_c here is a hyperparmeter to be tuned. It varies from 2 to the maximum length of the video. This sampling strategy is applied in both training set and validation set during training process.

However, sampling strategy is different for test dataset. In test process, fixed frames are selected to ensure an identical test environment. For single frame 2DCNN, the first frame of test videos will be used as input. In other cases, the first frame of every segment is selected.

The whole process of setting up the data is summarized in Figure 3.1.

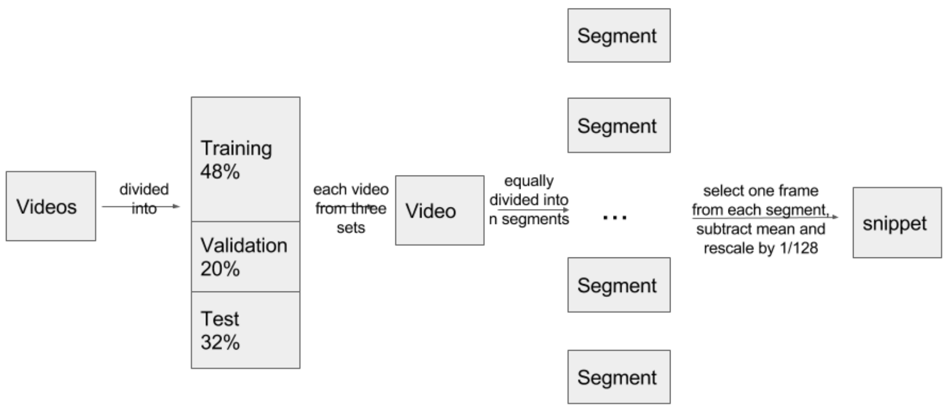

Videos
divided into
Training 48%
Validation 20%
Test 32%
each video from three sets
Video
equally divided into n segments
Segment
Segment
...
Segment
Segment
select one frame from each segment, subtract mean and rescale by 1/128
snippet

Figure 3.1: Video preprocessing

Cost function

Consider the task here is a classification task, softmax plus standard categorical cross entropy is used for loss function.

Evaluation metric

The evaluation metric used to judge the performance of trained model is accuracy. The defination of accuracy is same as (3.1).

3.2.2 Network Architecture

In this section, different network architectures to be experimented will be introduced here. There are three network architectures, namely, single frame 2DCNN, multi-frame 2DCNN, and RNN.

Single frame 2DCNN

Single frame 2DCNN is same as image classification. It is used here as a baseline to show classification results with only appearance information. There are four case studies performed for single frame 2DCNN.

The first case study investigates the influence of network size. There are two networks with different sizes. The first one is a comparatively small network with small

Case 1	Case 2	Case 3	Case 4
Network Size	Transfer learning	Number of nodes in 1st FC	Dropout rate

Table 3.3: Different case studies in single frame 2DCNN

number of layers and also small number of filters per convolutional layer, whose network architecture is shown in Table 3.4. The activation functions are not displayed here for simplicity. Despite the activation function of the last FC layer is softmax, all the other activation functions are set to Relu.

Conv1	Max	Conv2	Max	Conv3	Max	FC1	FC2
3*3*16	Pool	3*3*16	Pool	3*3*16	Pool	256	5

Table 3.4: The network architecture of small network

The second network structure is a modified VGG16 with top FC layers being replaced. The removal of FC layers is due to fewer number of categories to be classified compared with the original VGG16 trained for Imagenet with 1000 categories. The decision is also based on the reality of small dataset and limited computational resources. Hence, both the number of FC layers and the number of nodes are decreased. Here, the last three fully connected layers are replaced by two FC layers $[DP(p)- > FC(n_1)- > DP(p)- > FC(n_2)]$. The detail of modified VGG16 is shown in Table 3.5.

Conv1 3*3*64	Conv2 3*3*64	Max Pool	Conv3 3*3*128	Conv4 3*3*128	Max Pool	Conv5 3*3*256
Conv6 3*3*256	Conv7 3*3*256	Max Pool	Conv8 3*3*512	Conv9 3*3*512	Conv10 3*3*512	Max Pool
Conv11 3*3*512	Conv12 3*3*512	Conv13 3*3*512	Max Pool	FC1 256	FC2 5	Softmax

Table 3.5: The network architecture of a modified VGG16

The second case study explores the influence of transfer learning. There are in total 5 cases with different number of layers with fixed parameters. Train VGG16 from scratch, the whole network is randomly initialized and trained from scratch. Train VGG16 with pretrain, here all the parameters of convolutional layers are adapted from VGG 16 trained on Imagenet and fixed. The only trainable layers are the top two FC layers which are randomly initialized. Fine-tune VGG16 top layer, besides the two trainable FC layer, the last convolutional layer is also trainable. Fine tune VGG16 top 3 layer, the last three convolutional layers are all trainable. Fine tune VGG16 all layers, all the networks parameters are trainable in this scenario.

The third case study investigates the influence of number of nodes in the first FC layer, which is n_1. The default number of nodes used in previous cases is 256. And the default number of nodes used in the following cases is 512.

The fourth case study investigates the influence of dropout rate. Most of the machine learning related projects suffer from overfitting, thus dropout rate here is tuned to get better generalization ability. Both the second and third case study use the modified VGG16 network structure in the first case study.

Multi-frame 2DCNN

Multi-frame 2DCNN uses 2DCNN to extract features of multiple frames and then aggregates the features together for prediction. The aggregation of the features are done by a consensus function. The basic network structure is same as modified VGG16 in Table 3.5. One difference is that the number of nodes in the first FC layer is 512. Before the consensus layer, all the frames go through a same network with same parameters. After the consensus layer, the output will have the same size as one frame 2DCNN case and then feed through the following layers. There are three case studies performed using multi-frame 2DCNN.

Case 1	Case 2	Case 3
Consensus function	Position of consensus function	Number of input segments

Table 3.6: Different case studies in multi-frame 2DCNN

The first case study investigates different consensus functions. Consensus function defines how to aggregate features from multiple frames into the same size as feature of one frame. There are two consensus functions tested. The first one is max pooling function:

$$X = max(F_1, F_2, \dots, F_k) \tag{3.3}$$

Here F_i represents the feature of ith frame. k is the total number of frames. The second one is average pooling function:

$$X = \frac{F_1 + F_2 + \cdots + F_k}{k} \tag{3.4}$$

A demonstration of the network used in the first case study is shown in Figure 3.2.

The second case study explores the position of layer to extract features. The consensus function used here is max pooling. There are three different layer experimented to extract feature. The first case, the consensus function is inserted between the last convolutional layer of VGG16 and the first FC layer. In the second case. the consensus function is inserted between the first FC layer and the second FC layer. In the third case, the consensus function is applied after the second FC layer. The second case is performed in the above case study. And the architectures of the first and third case is respectively shown in Figure 3.3 and Figure 3.4.

The third case study investigates the influence of number of input frames. The consensus function used is max pooling and the final convolutional layer is used for feature extraction. The number of input frames being tested ranges from 5 to all the available frames.

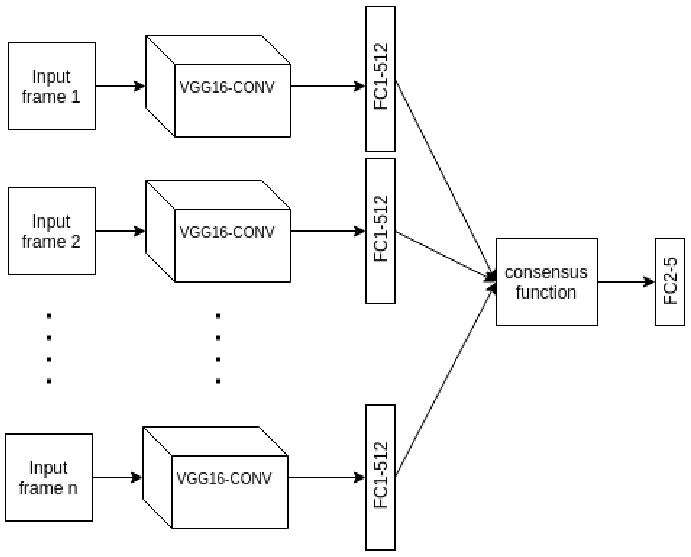


Figure 3.2: Architecture of multi-frame 2DCNN in the first case study

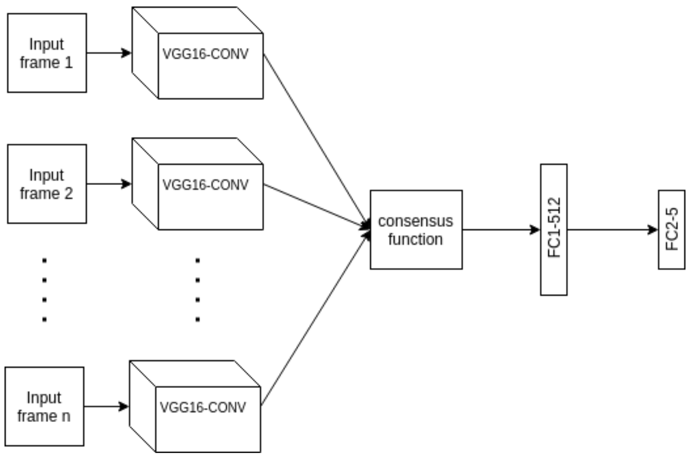


Figure 3.3: Architecture of multi-frame 2DCNN with in the second case study with consensus function between VGG16-CONV and FC1

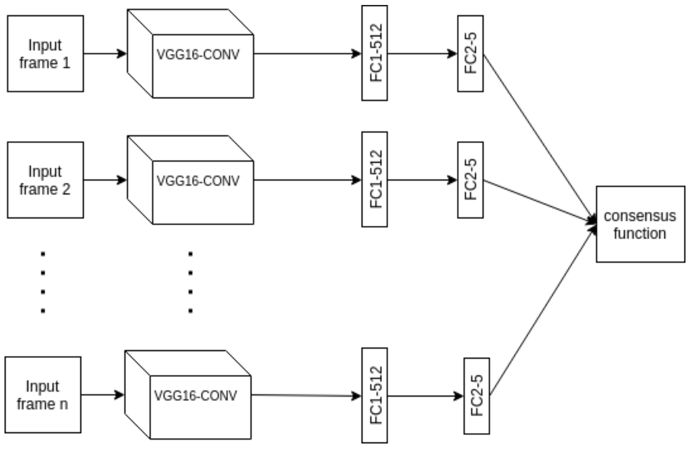


Figure 3.4: Architecture of multi-frame 2DCNN in the second case study with consensus function after FC2

RNN

As in multi-frame 2DCNN, the RNN network uses the same convolutional networks to extract features. Then the features are fed into recurrent neural network. There are three case studies performed for RNN.

Case 1	Case 2	Case 3
RNN unit	Number of RNN layers	Number of input segments

Table 3.7: Different case studies in RNN

The first case study investigates the influence of different recurrent unit. The first one is the simple RNN cell as defined in (2.24). The second one is LSTM (2.25). The third one is GRU (2.26). The details of the three RNN units are presented in Section 2.4.2. Here the number of input frames is five.

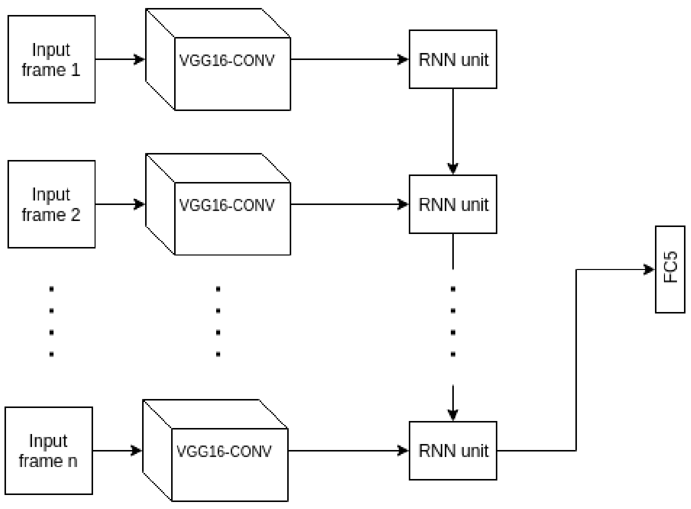


Figure 3.5: Architecture of RNN in the first case study

The second case study investigates the influence of number of recurrent layers. Here the default RNN unit is chosen as LSTM and the number of input frames is five.

The third case study investigates the influence of number of input frames. Here the default RNN unit is chose as LSTM. And only one LSTM layer is used.

3.2.3 Network Training

Stochastic gradient descent descent with Nesterov momentum is used here to learn the network parameters as in image classification. The momentum is set to 0.9. Step decay is used here with learning rate decay equals to 0.000001. The batch size is set to 1 due to memory limitation. The initial learning rate is set as 0.001. The learning rate is reduced to one-tenth of the original manually if severe divergence is observed.

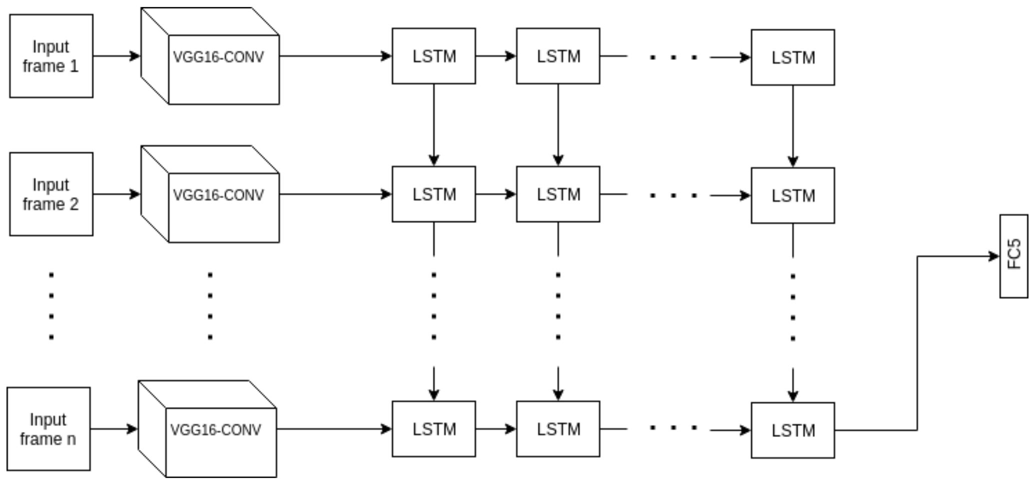


Figure 3.6: Architecture of RNN in the second case study

Each model will be trained for at least 15 epochs and will be prolonged if the network haven't converged yet. The convergence is considered when the validation loss is not decreasing for consecutive 10 epochs. After convergence of training loss and validation loss, the model with the smallest validation loss will be chosen as the final model for test. Due to computation resource limitation, the model will be trained maximum 100 epochs despite converge or not. Figure 3.7 shows the whole process.

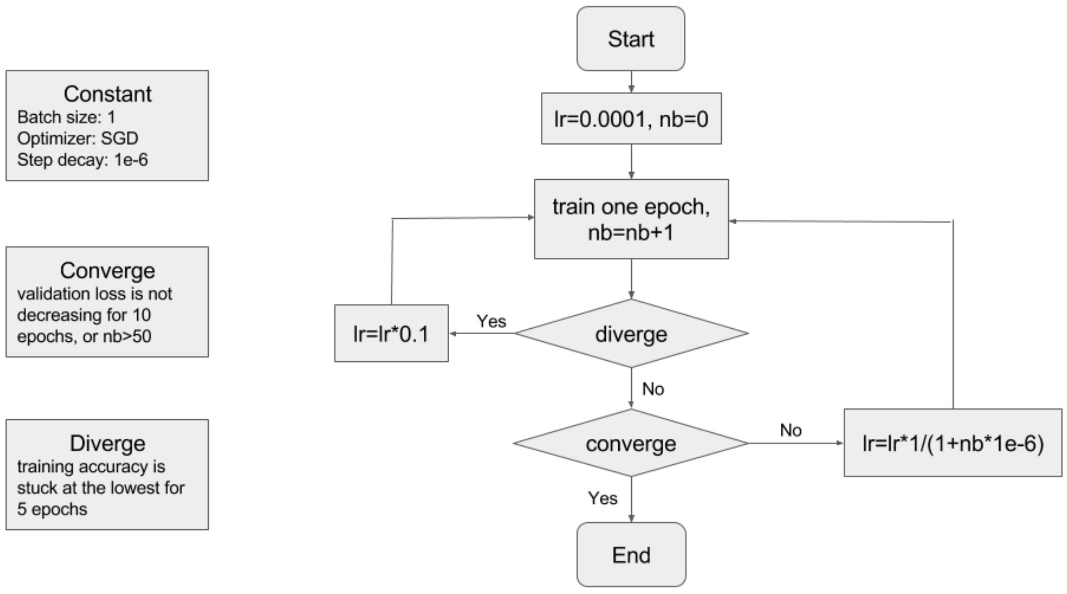


Figure 3.7: Demonstration of network training

4

Results and Analyses

4.1 Environmental setup

The experiments is done in two environments. One uses CPU Intel Core™2 Quad CPU Q9400 @ 2.66GHz $\times$ 4 with 2GB DDR3 RAM for testing. Another uses GPU Tesla K40m on a cluster for training.

The deep learning libraries used in traffic sign recognition part are Theano and Lasagne. And the deep learning libraries used in video classification are Keras and Tensorflow. The programming language is python.

4.2 Results for traffic sign recognition

4.2.1 Dataset description

The German Traffic Sign Recognition Benchmark is published for a classification challenge held in the International Joint Conference on Neural Networks (IJCNN) 2011. The GTSRB dataset was collected in different weather conditions by capturing videos from a car with varying speed [25]. Therefore, the collected images have large variations in visual appearances due to illumination changes, partial occlusions, rotations, weather conditions, etc. This makes the GTSRB a both valuable and challenging task. Because of the sufficient number of labeled training data, GTSRB is chosen as the data for image classification using CNN.

The GTSRB consists of training set with 39209 images and test set with 12630 images. It has in total 43 class. The size of the image varies between 15*15 and 222*192. Besides it has unbalanced class frequencies as shown in Figure 4.2.

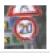
20 Speed limit class #00 	30 Speed limit class #01	50 Speed limit class #02
60 Speed limit class #03	70 Speed limit class #04	80 Speed limit class #05
End of 80 speed limit class #06	100 Speed limit class #07	120 Speed limit class #08
No overtaking class #09	No over taking by heavy vehicles class #10	Crossroads class #11
Priority road class #12	Give way class #13	Stop and give way class #14
Restricted vehicular access class #15	No large heavy vehic class #16	No entry for vehicular traffic class #17
Other danger class #18	Bend, to left class #19	Bend, to right class #20
Double bend, first to left class #21	Uneven road class #22	Slippery road class #23
Road narrows on right class #24	Road works class #25	Traffic lights class #26
Pedestrians class #27	Children crossing class #28	Bicycles crossing class #29
Snow class #30	Animal crossing class #31	End of all speed and passaging limits class #32
Turn right ahead class #33	Turn left ahead class #34	Ahead only class #35
Go straight or right class #36	Go straight or left class #37	Keep right class #38
Keep left class #39	Roundabout mandatory class #40	End of no overtaking class #41
End of no overtaking by trucks class #42		

Figure 4.1: Images of the GTSRB dataset

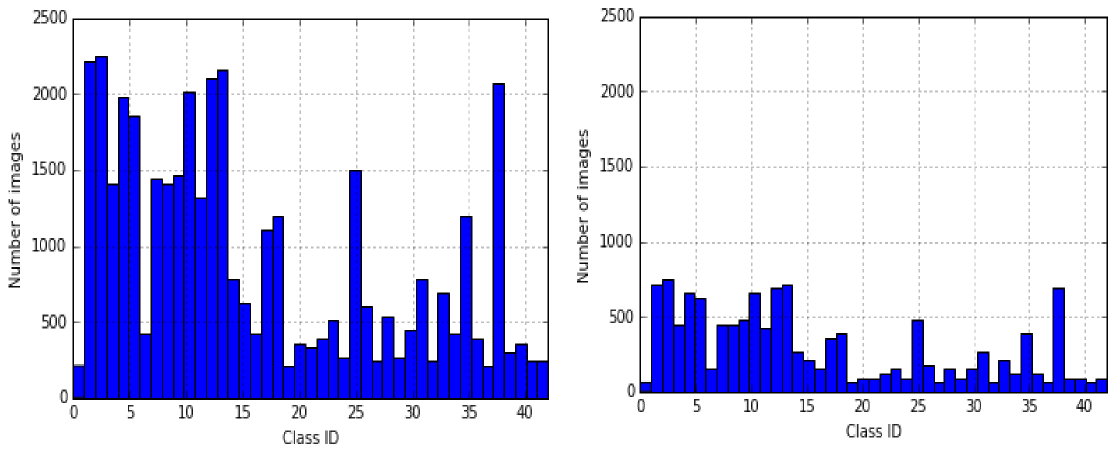


Figure 4.2: Left: Histogram of training data, Right: Histogram of test data

To give a basic impression of what the images in this dataset looks like, Figure 4.1 demonstrates samples from all 43 categories with name. Notice that the size of really images vary a lot, here images with moderate size are chose to have a better visualization.

4.2.2 Data preparation

The training data is split randomly with ratio 7:3, while 70% of images belong to training set and 30% of images belong to validation set. Totally, 27446 images are used in training set, 11763 images are used in validation set, and 12630 images are used in test set, as shown in table 4.1. The images are simply preprocessed by division of 128 and mean subtraction along each color channel to roughly scale the intensity range to $[-1.0, 1.0]$.

Data	Training set	Validation set	Test set
Number	27446	11763	12630
Ratio	52.94%	22.69%	24.36%

Table 4.1: The number of images in different sets

4.2.3 Hyperparameter tuning

Hyperparameter tuning is the main component for traffic sign recognition part. In this part, the procedure for hyperparmeter tuning will be demonstrated. The relationship between tuned hyperparameters and result will be discussed. There are four hyperparameters to be tune. Their default values are shown below:

Hyperparameter	Learning rate	Momentum	Batch size
Default value	0.0001	0.9	1

Table 4.2: The default value of hyperparameters to be tuned with SGD as the optimizer

Learning rate

The first tuned hyperparameter is learning rate. The initial learning rate is set to 0.001. Figure 4.3 shows the training and validation accuracy for 30 epochs while learning rate is equal to 0.001. It is clear that this learning rate is not leading the learning process in a right direction. This is because that the learning rate is too high which makes learning process easily affected by bad samples. Figure 4.4 shows the result under similar environment with only changing learning rate to 0.0001. This learning rate is considered as a good initial learning rate.

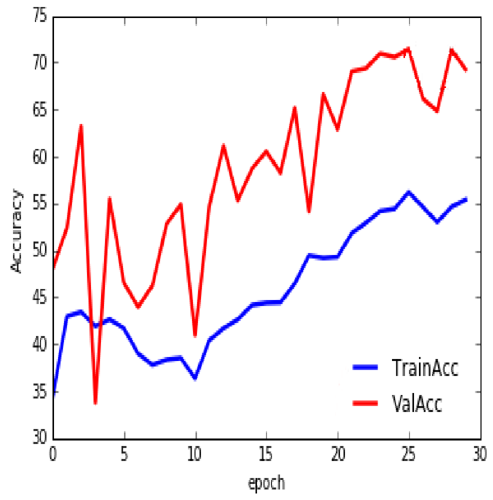


Figure 4.3: Learning process while learning rate is 0.001

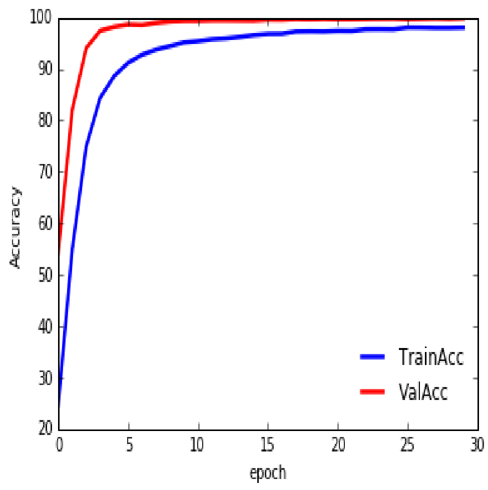


Figure 4.4: Learning process while learning rate is 0.0001

Momentum

The second hyperparameter to be tuned is momentum. The optimizer used here is Stochastic Gradient Descent (SGD) with Nesterov momentum. The range of momentum is 0 to 1. 10 random numbers are selected between [0,1]. A network is trained 5 epochs with these 10 different momentum value. Validation accuracy is used to evaluate. The result is shown in Figure 4.5. Based on the result above, it is reasonable to set momentum as 0.9.

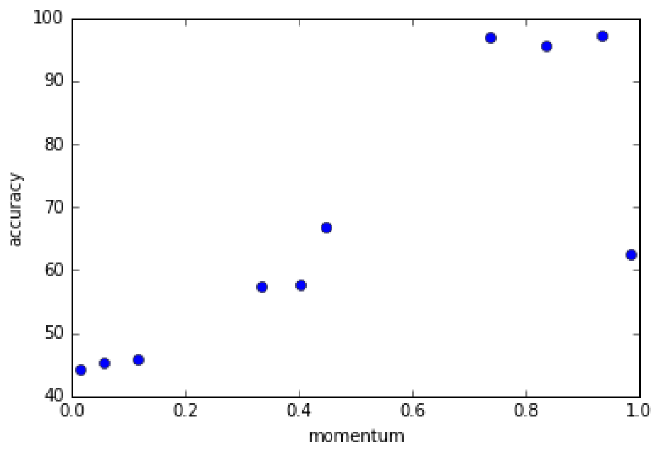


Figure 4.5: Validation accuracy after 5 epochs

Batch size

The third hyperparameter is batch size. It is not a good idea to fix the number of trained epochs while evaluating the effect of batch size as done before. If the learning rate is fixed, network with larger batch size will be updated less. So a better strategy is to train the model until validation error is not improving anymore. The result is shown in Table 4.3.

It is shown in the table that used time per epoch decreases with the increase of batch size. This is reasonable. As stated before, network weights will be updated less with a larger batch size. However, the total time taken for converge is not tightly related to batch size. Based on the result, batch size is selected as 1.

Batch size	1	10	50	100	500
time/epoch	51.27s	8.24s	5.22s	4.72s	4.25s
total time	11433s	2364s	12403s	7788s	21166s
training accuracy	99.31%	98.00%	98.64%	96.16%	94.55%
validation accuracy	99.80%	99.64%	99.76%	99.45%	99.13%

Table 4.3: The effect of batch size

4.2.4 Final test

After tuning the hyperparameters mentioned above, the model is trained for 200 epochs with training accuracy 99.67% and validation accuracy 99.80%. The model is considered as a converged model since the validation loss has been around 0.0085 for more than 100 epochs.

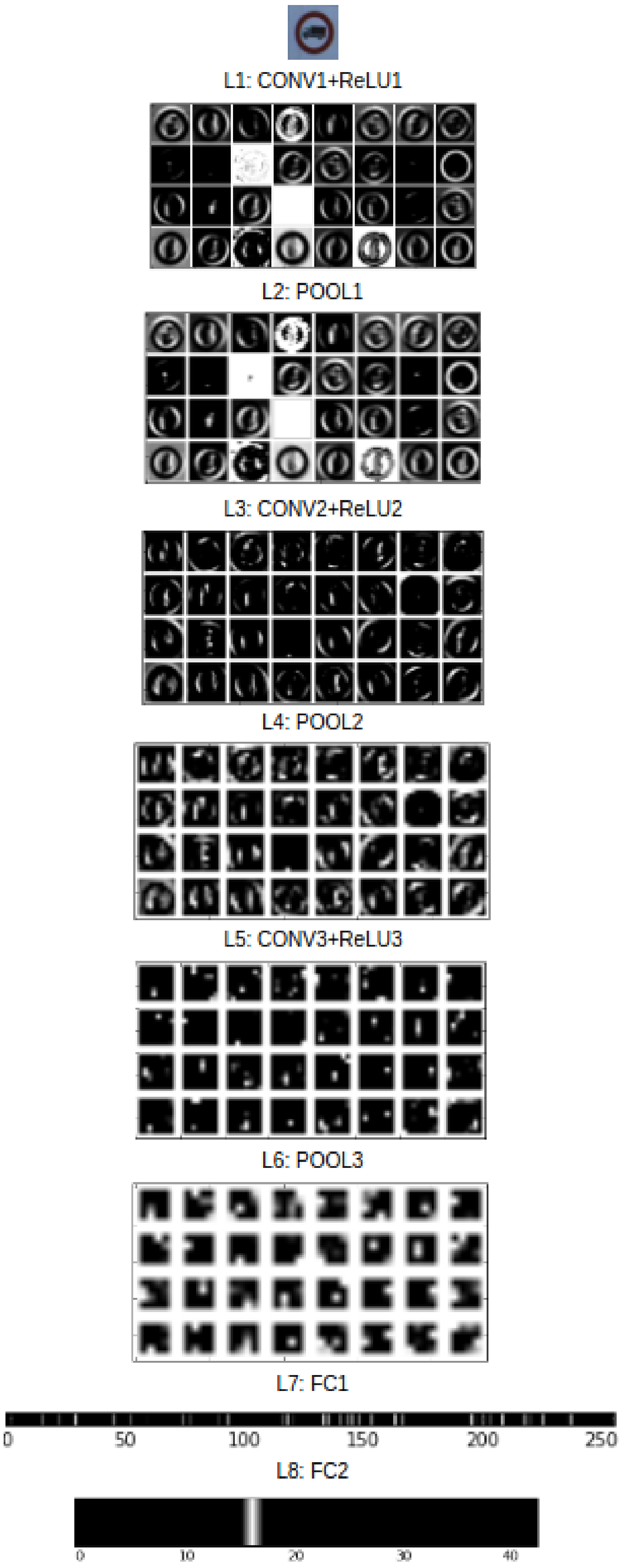


Figure 4.6: The change of data in the network

Figure 4.6 uses one image as an example to visualize what the input data will evolve in the network. Besides, the size of the data between each layer is also presented. Figure 4.7 shows the learning process.

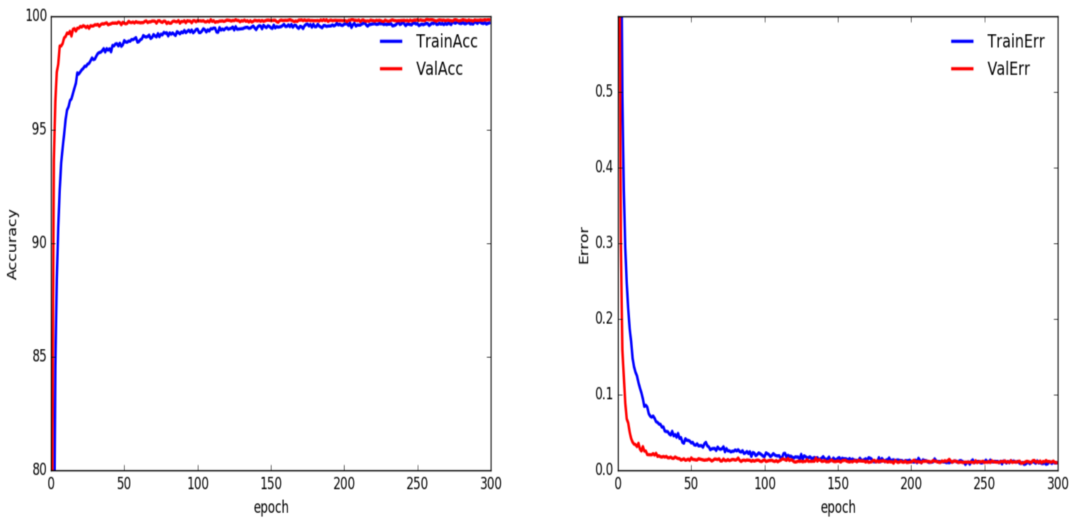


Figure 4.7: Learning process. Left: Accuracy, Right: Error

This final network is applied to the test data and get a test accuracy of 98.65%. The winner in the GTSRB competition achieved test accuracy 99.46% with the use of model ensemble [2]. Since the aim of this part in this thesis is to explore the learning control part of a convolutional neural network, model ensemble is not used here. As in single model case, they got the test accuracy around 98.5%. However, the data they used is central cropped, which has less background noise. The full image used in this thesis is more realistic. So experiment on central cropped images is also performed, and the result is shown in Table 4.4.

Data	Training accuarcy	Test Accuracy
Raw images	99.80%	98.65%
Central patches	99.85%	98.83%

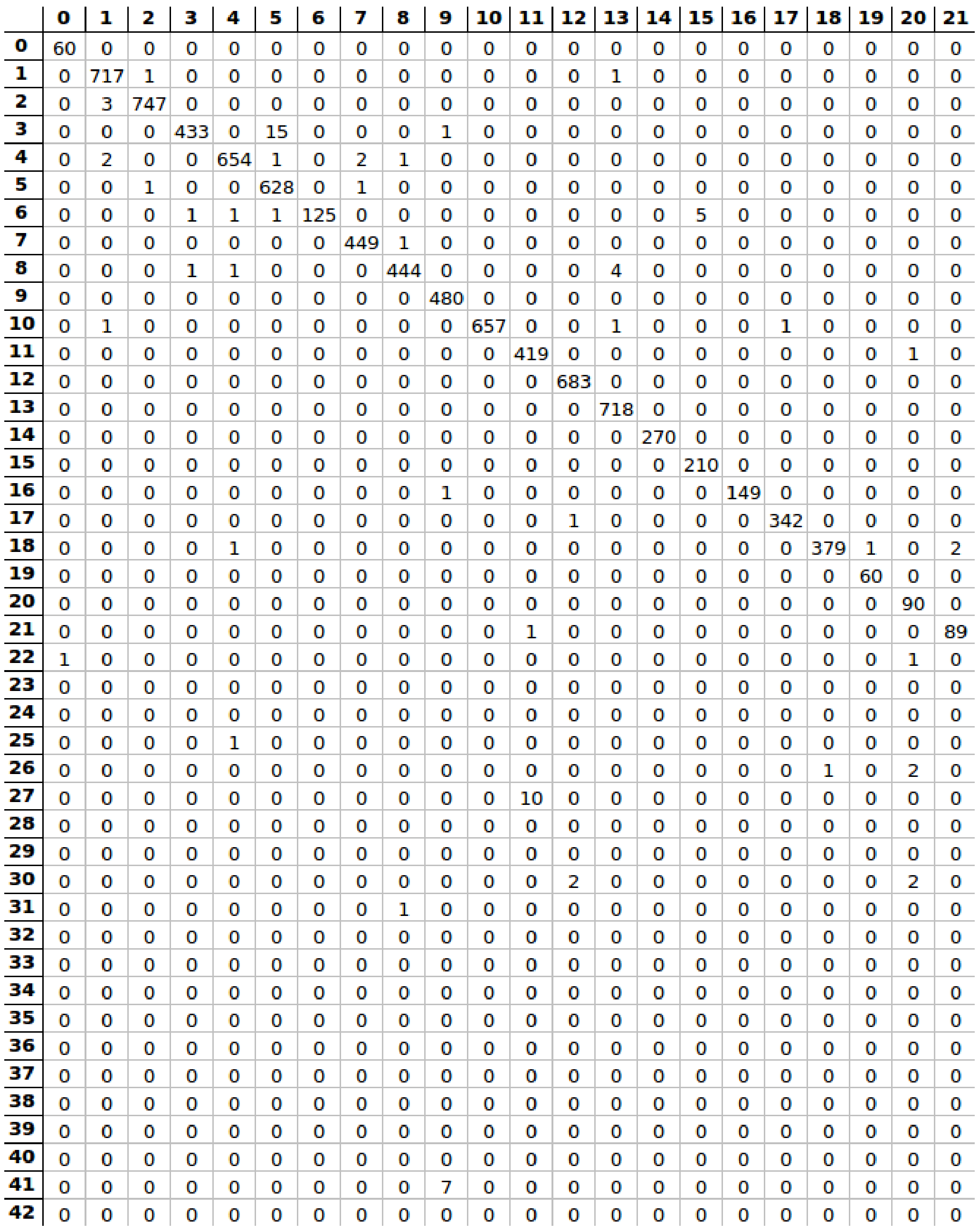
Table 4.4: Result comparison regarding different input

Compared with model in this thesis, they used a way much larger network compared with the network used in this thesis. As calculated in 3.1.2, the number of parameters used in this network is 325792. However, there are 2067600 parameters used in one network in their work, which is 6 times more than parameters used in this network. The comparison result is summarized in Table 4.5.

As for the newest published test accuracy 99.67% [3], they generated the test data by themselves which makes their result incomparable to the result in this thesis because of different test environment.

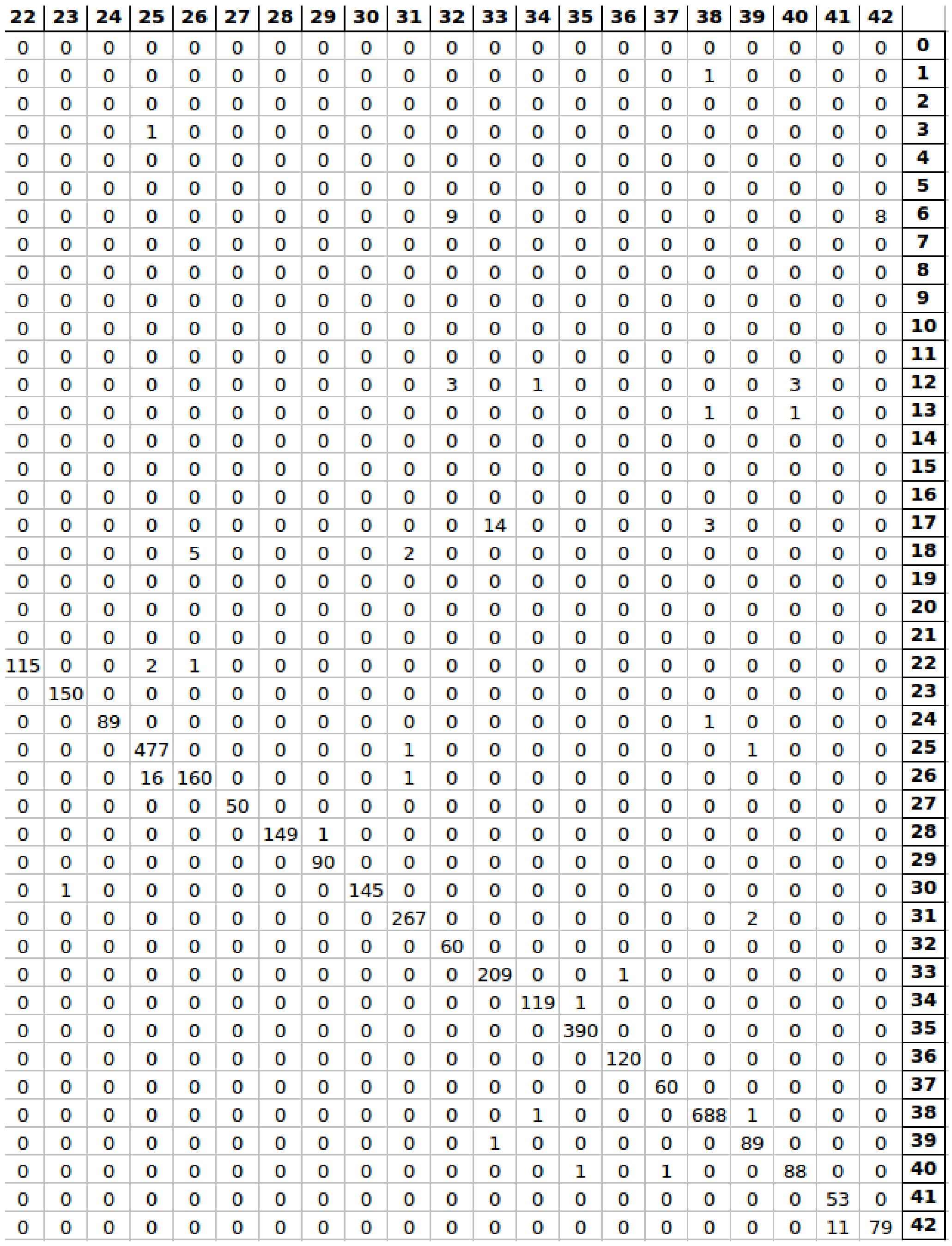
Work	Test accuarcy	Number of parameters
Cire's work [2]	98.52%	2,067,600
The proposed work	**98.83%**	154,260

Table 4.5: Result comparison with winner in the GTSRB competition using single network



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	717	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	0	3	747	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	433	0	15	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	2	0	0	654	1	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	1	0	0	628	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	1	1	125	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
7	0	0	0	0	0	0	0	449	1	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	1	1	0	0	0	444	0	0	0	0	4	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0
10	0	1	0	0	0	0	0	0	0	0	657	0	0	1	0	0	0	1	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	419	0	0	0	0	0	0	0	0	1	0
12	0	0	0	0	0	0	0	0	0	0	0	0	683	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	718	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	270	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	210	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	149	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	342	0	0	0	0
18	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	379	1	0	2
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	0
21	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	89
22	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0
27	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	2	0
31	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.8: Part 1 of the confusion matrix obtained from the final model

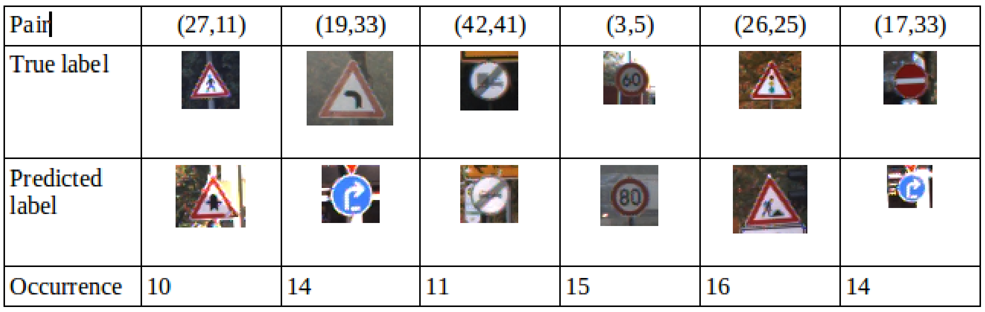


22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	**1**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**3**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**4**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**5**
0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	8	**6**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**7**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**8**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**9**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**10**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**11**
0	0	0	0	0	0	0	0	0	0	3	0	1	0	0	0	0	0	3	0	0	**12**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	**13**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**14**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**15**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**16**
0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	3	0	0	0	0	**17**
0	0	0	0	5	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	**18**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**19**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**20**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**21**
115	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**22**
0	150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**23**
0	0	89	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	**24**
0	0	0	477	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	**25**
0	0	0	16	160	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	**26**
0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**27**
0	0	0	0	0	0	149	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**28**
0	0	0	0	0	0	0	90	0	0	0	0	0	0	0	0	0	0	0	0	0	**29**
0	1	0	0	0	0	0	0	145	0	0	0	0	0	0	0	0	0	0	0	0	**30**
0	0	0	0	0	0	0	0	0	267	0	0	0	0	0	0	0	2	0	0	0	**31**
0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	**32**
0	0	0	0	0	0	0	0	0	0	0	209	0	0	1	0	0	0	0	0	0	**33**
0	0	0	0	0	0	0	0	0	0	0	0	119	1	0	0	0	0	0	0	0	**34**
0	0	0	0	0	0	0	0	0	0	0	0	0	390	0	0	0	0	0	0	0	**35**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	**36**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	**37**
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	688	1	0	0	0	**38**
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	89	0	0	0	**39**
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	88	0	0	**40**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	53	0	**41**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	79	**42**

Figure 4.9: Part 2 of the confusion matrix obtained from the final model

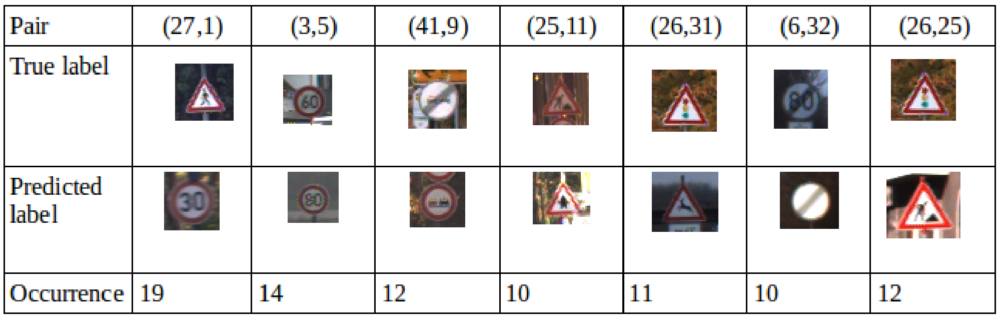
To have a better visualization of the outcome for the final network, a reasonable way is to show all the misclassified images. The total number of images in test set is 12630, which means around 185 images are not classified successfully using the network in this thesis. The confusion matrix of the final test model is shown by figure 4.8 and figure 4.9.

It is obvious that most of the test images are correctly classified. Here misclassified image pairs whose number is larger than 10 is selected, which are (27,11), (19,33), (42,41), (3,5), (26,25) and (17,33). A clear demonstration is shown in 4.10. To test if the result is consistent, another model is trained with test accuracy 99.53% is demonstrated in the same way in 4.11, with image pairs (27,1), (3,5), (41,9), (25,11), (26,31), (6,32), and (26,25).



Pair	(27,11)	(19,33)	(42,41)	(3,5)	(26,25)	(17,33)
True label						
Predicted label						
Occurrence	10	14	11	15	16	14

Figure 4.10: Misclassified image pairs of the final model



Pair	(27,1)	(3,5)	(41,9)	(25,11)	(26,31)	(6,32)	(26,25)
True label							
Predicted label							
Occurrence	19	14	12	10	11	10	12

Figure 4.11: Misclassified image pairs of another model

Based on Figure 4.10 and Figure 4.11, it is noticed that some misclassified pairs can be interpreted easily because the image similarity, for example, pedestrians and crossroads, however some are not relevant to the visual similarity, for example, pedestrians and 30 speed limit. Pair (3,5) and (26,25) appear in both situations, which means 60 speed limit is always misclassified as 80 speed limit, and traffic light is always misclassified as road work. This might be solved by further training the model with these hard examples. Additionally, there are also some pairs only present once in either of the two models. That illustrates why model ensemble can work pretty well.

4.3 Results for human activity recognition on 5 activity classes

4.3.1 Dataset description

Before working on fall datasets, experiments have been done on a sub set of UCF101 [26]. UCF101 has 101 action categories and each is grouped into 25 groups, where each group can consist of 4-7 videos of an action. The videos from the same group may share some common features, such as similar background, similar viewpoint, etc. The frame rate is 25 frames per second. The duration of the video clips varies from 0 to 10 seconds per video. On average, there are 185 frames for each video. The image size is identical, all is (320,240).

Due to the limit of computation budget, 101 categories in UCF101 dataset is too large for testing different networks. As a result, only 5 out of 101 actions are randomly chosen. They are bench press, boxing speed bag, jump rope, still rings and trampoline jumping. There are in total 322, 133, 214 videos for training, validation and test sets.

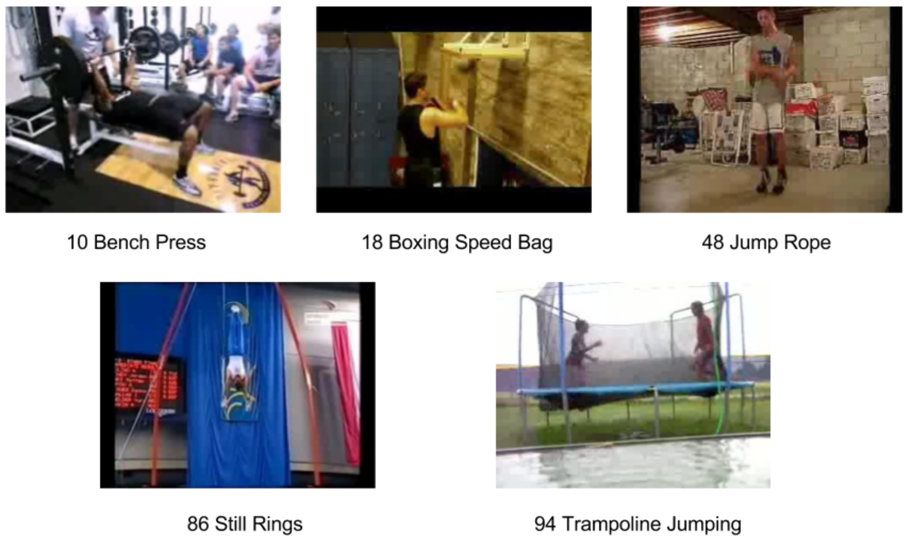


Figure 4.12: Illustration of the 5 actions used in our experiments chosen from UCF101. The number is the index of corresponding action

4.3.2 Results for single frame 2DCNN

2DCNN is used as a baseline to measure the accuracy with only spatial information. Three case studies are performed to pursue better performances.

Case study 1

The first case study investigates the influence of different network structures.The result is shown Table 4.6. Based on the result below, it shows that there is no big difference between shallow network and deep network in train from scratch case.

Model	Validation accuracy	Test accuracy	# of parameters
Small network	52.27%	58.41%	4,921,829
Modified VGG16	60.90%	50.47%	23,891,269

Table 4.6: Results on 5 classes for various networks trained from scratch in single frame 2DCNN

Case study 2

The second case study investigates the influence of various Transfer Learning approaches using the modified VGG network. It is clear that there is no big difference between the last three settings in Table 4.7. However, the second setting, "Train VGG16 with pre-train", shows relatively low accuracy. The reason is the number of nodes in the first FC layer is 256 and the parameters of the convolutional layers are fixed, which makes the model suffer from underfitting. Despite the second setting, it is clear that network trained from scratch has worse perfromance compared with network with pretrained parameters.

Training setting	Validation accuracy	Test accuracy
Train modified VGG16 from scratch	60.90%	50.47%
Train modified VGG16 with pre-train	18,18%	24.50%
Fine-tune modified VGG16 top layer	80.30%	82.00%
Fine-tune modified VGG16 top 3 layers	81.06%	69.50%
Fine-tune modified VGG16 all layers	80.83%	78.50%

Table 4.7: Results on 5 classes for various Transfer Learning approaches in single frame 2DCNN

Case study 3

In the second case study, the second setting displays relatively poor performance. The reason is assumed as underfitting with limited parameters. So here in the third case study, the number of nodes in the first FC layer is investigated. Since the number of nodes in the second FC layer is fixed as the number of categories to be classified, the number of nodes in the first FC layer is the only adjustable layer for changing the number of network parameters. Based on the result from Table 4.8, the assumption is proven to be true, that the previous model does suffer from underfitting. It also shows that 512 is a reasonable setting. 256 suffers from underfitting. And there is no improvement with increased number of nodes.

# of nodes	Validation accuracy	Test accuracy	# of parameters
256	18.18%	24.50%	9,176,581
512	94.74%	84.00%	18,353,157
1024	90.98%	79.50%	36,706,309

Table 4.8: Results on 5 classes for various number of nodes in the first FC layer in single frame 2DCNN

Case study 4

The fourth case study investigates the influence of dropout rate. Though the training accuracy is not shown here in this thesis, the training accuracy is actually always around 99%. It is clear the model is suffering from overfitting with relatively lower validation accuracy. A common strategy to alleviate overfitting is to increase dropout rate. However, based on the result from Table 4.9, the dropout rate doesn't play an important role in the performance in this scenario. The learning rate in the Table 4.9 means the initial learning rate required to converge, which means if the learning rate is increased by ten times the model will diverge. It shows that with the increase of dropout rate, the required learning rate is decreasing.

Dropout rate	Validation accuracy	Test accuracy	Learning rate
0.5	94.74%	84.00%	0.001
0.6	93.23%	83.00%	0.0001
0.7	93.98%	81.90%	0.00001

Table 4.9: Results on 5 classes for various dropout rate in single frame 2DCNN

4.3.3 Results for multi-frame 2DCNN

Compared with single frame 2DCNN, multi-frame 2DCNN utilizes the temporal information without adding any parameters. There are in total three case studies performed in this part.

Case study 1

The first case study investigates the influence of consensus function. As shown in Table 4.10, the consensus function, max pooling, consistently provides slightly better performance in both validation accuracy and test accuracy case. So here consensus function is chosen as max pooling for following case studies in multi-frame 2DCNN.

Consensus function	Validation accuracy	Test accuracy
Max pooling	95.49%	87.62%
Average pooling	94.74%	86.92%

Table 4.10: Results on 5 classes for various consensus function in multi-frame 2DCNN

Case study 2

The second case study investigates which layer of features should we extract. Consensus layer represents the last layer before appling consensus function. Based on Table 4.11, it shows that to extract feature from CONV layer gives the highest validation accuracy. However, extracting feature from the first FC layer produces slight higher test accuracy. Since the difference is not distinctive, CONV layer is chosen for extract feature for following case studies in multi-frame.

Consensus Layer	Validation accuracy	Test accuracy
CONV	98.50%	86.20%
FC1	95.49%	87.62%
FC2	94.74%	84.11%

Table 4.11: Results on 5 classes for various consensus layer in multi-frame 2DCNN. Details in Figure 3.3, Figure 3.2, Figure 3.4

Case study 3

The third case study investigates the number of segments needed for each video. As shown in Table 4.12, there is no big difference between different number of segments. Five segments per video is sufficient for classification in multi-frame 2DCNN case.

# of segments	Validation accuracy	Test accuracy
5	98.50%	86.20%
10	96.99%	87.62%
20	95.49%	85.71%
40	100.00%	87.14%
all	98.50%	85.51%

Table 4.12: Results on 5 classes for various segment numbers in multi-frame 2DCNN

4.3.4 Results for RNN

In multi-frame 2DCNN case, all the frames are treated equally without utilizing the time order. RNN provides a network to deal with time sequence data.

Case study 1

The first case study investigates the influence of different RNN units. There is no significant difference between different RNN units. In all three tested RNN unit, LSTM provides the best accuracy. Hence LSTM is chosen for the following case study in RNN.

RNN unit	Validation accuracy	Test accuracy	# of parameters
simple RNN	95.49%	80.95%	9,240,832
LSTM	97.00%	84.11%	36,964,613
GRU	93.98%	82.38%	27,723,781

Table 4.13: Results on 5 classes for various RNN unit in RNN

Case study 2

The second case study explores the influence of the number of RNN layers. It shows in Table 4.14 that there is no significant change with the increase of RNN depth. Hence, network with one LSTM layer is used in the following case study.

# of layers	Validation accuracy	Test accuracy	# of parameters
1	97.00%	84.11%	36,964,613
2	93.98%	86.66%	37,489,925
3	93.23%	84.76%	38,015,237
5	99.25%	83.81%	39,065,861

Table 4.14: Results on 5 classes for various number of LSTM layers in RNN

Case study 3

The thrid case study investigates the number of segments needed for each video. Based on result in Table 4.15, ten segments per video provides the best performance.

# of segments	Validation accuracy	Test accuracy
5	97.00%	84.11%
10	98.50%	85.51%
20	90.23%	82.71%
40	92.48%	85.05%
all	90.98%	81.31%

Table 4.15: Results on 5 classes for various segment numbers in RNN

4.3.5 Comparison

The best test performance of the three networks are summarized in Table 4.16. Compared with baseline single-frame 2DCNN, multi-frame 2DCNN and RNN provides slightly better results. Among all these models, multi-frame 2DCNN achieves the highest test accuracy, which is 87.62%. However, it is disappointing that there is no big improvements with more temporal information.

Accuracy	Single-frame 2DCNN	Multi-frame 2DCNN	RNN
Validation accuracy	94.74%	96.99%	93.98%
Test accuracy	84.00%	87.62%	86.66%

Table 4.16: Results on 5 classes for various networks

4.4 Results for human fall classification

From previous study in Section 4.3, there is no evident improvement with additional temporal information. The reason could be the selected activities are associated with static objects, which means the model could be able to classify the action by directly detecting the corresponding objects without further temporal information. Hence, to better study the effectiveness of the models presented, another human activity dataset called multiple cameras fall dataset [27] is experimented in this section.

4.4.1 Dataset description

Multiple cameras fall dataset has 24 scenarios recorded with 8 IP video cameras. The first 22 scenarios contain a fall and confounding events, the last 2 scenarios contain only confounding events. There are in total 400 video clips including 184 video clips for fall and 216 video clips for other activities. For simplicity, all the 400 video clips are downsampled into 5 frame. Figure 4.13 shows 4 examples for the video clips used in this thesis. There are in total 5 classes, falling down and other negative classes including sitting down, crouching down, lying down and walking.

# of video clips	Training	validation	test	total
Fall	112	32	40	184
Negative class	112	40	64	216
Total	224	72	104	400
Ratio	56%	18%	26%	100%

Table 4.17: Number of video clips in the three sets

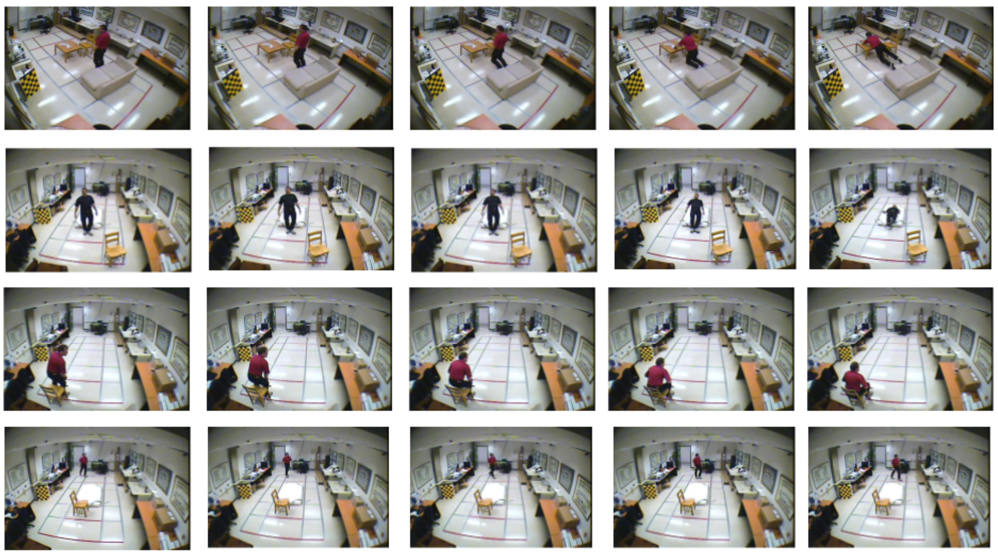


Figure 4.13: Examples of multiple cameras fall dataset. The above two rows are fall activities and the bottom two rows are two examples (sitting down and walking) from negative class containing 4 activities including sitting down, crouching down, lying down and walking.

The whole dataset is randomly splitted into training, validation and test set. Video clips recording the same event but with different camera views are assigned to same set. Further details are shown in Table 4.17. The preprocesing step is same as in Section 4.3. All the frames in each video clips are first resized into 224*224 and then the pixel intensity values are rescaled to range -1.0 to 1.0.

4.4.2 Results

Same as in Section 4.3, single-frame CNN, multi-frame CNN and RNN are investigated here. Due to time and computation limits, each model selects only one factor to research.

Results of single-frame CNN

Single frame CNN is basically image classification. Here mainly two different network structures are examined. The first network is of small size, which has the same structure as in the small network in Section 4.3. The only difference is the number of nodes in the last FC layer due to different number of classes to be classified. The second network is a combination of pretrained VGG16 CONV part and two randomly initialized FC layers with number of nodes 256 and 2. All the parameters in the second network is trainable. A detailed description of these two network is shown in Section 3.2.2.

The result on the two network structure in shown in Table 4.18. It shows that there is no big performance difference in the two networks. The second network, VGG16 with finetune provides slightly better result.

Model	Validation accuracy	Test accuracy
Small network	66.67%	72.92%
Modified VGG16 with pre-train	65.28%	77.08%

Table 4.18: Results on multiple cameras fall dataset using single-frame CNN with different network structures

Results of multi-frame CNN

In multi-frame CNN, the factor to be experimented is the position of consensus function, which is introduced in Section 3.2.2. The consensus function used here is maxpooling. And the CNN part is pretrained VGG16. The number of nodes in FC layers are 256 and 2. The parameters in VGG16 is not trainable except the top three CONV layers, which are Conv11-Conv13. This act could speed up the training time while in the mean time tune the model to fit this dataset. A detailed demonstration of the network structure is shown in Figure 3.3 and Figure 3.2 in Section 3.2.2. The result is presented in Table 4.19. It shows that the network with features aggregated right after CONV layer has slightly better performance.

Consensus layer	Validation accuracy	Test accuracy
CONV	80.55%	85.41%
FC1	81.94%	84.38%

Table 4.19: Results on multiple cameras fall dataset for various

Results of RNN

The network structure used in this part is demonstrated in Figure 3.5 in Section 3.2.2. The RNN unit chosen here is LSTM. And the dimensionality of the output space of LSTM is set to 256. The effect of transfer learning in RNN is explored here. As shown in Table 4.20, the bigger networks which uses VGG16 as the CONV part consistently provides better result compared with a small network. Among all the bigger networks, the one only finetunes the last CONV layer of VGG16 gives the best result, which reaches 85.42% test accuracy and 84.72% validation accuracy.

Model	Validation accuracy	Test accuracy
Small network	77.78%	79.17%
VGG16 without fine-tuning	70.83%	81.25%
Fine-tune modified VGG16 Conv13	84.72%	85.42%
Fine-tune modified VGG16 Conv11-Conv13	83.33%	81.25%
Fine-tune modified VGG16 all layers	81.94%	77.08%

Table 4.20: Results on multiple cameras fall dataset on exploring different transfer learning strategy

Comparison

In this part, the result with highest validation accuracy in each of the three models are selected. The results are summarized in Table 4.21 to compare the performance of the three models on multiple camera fall dataset. As clearly shown in Table 4.21, both multi-frame 2DCNN and RNN provide much better performance compared with single-frame 2DCNN model. The result is different from the situation in Section 4.3 where no big difference is observed. Hence, regarding classifying actions rely heavily on time, models like multi-frame 2DCNN and RNN do provide better results. Additionally, the RNN model provides slightly better result compared with multi-frame 2DCNN. Multi-frame 2DCNN simply aggregates all the information from different frame with no emphasis on order. But order does play an important role in action recognition. This could be the reason why RNN has better result.

Accuracy	Single-frame 2DCNN	Multi-frame 2DCNN	RNN
Validation accuracy	66.67%	81.94%	**84.72%**
Test accuracy	72.92%	84.38%	**85.42%**

Table 4.21: Results on multiple cameras fall dataset for various networks

5

Conclusion and Future Work

Conclusion

The work done in this thesis is to use deep learning for image classification on traffic sign and video classification on human activity. Satisfactory result are gotten in both cases.

In traffic sign part, this thesis mainly focuses on tuning the hyperparameters related to learning control. Learning rate, batch size, and momentum are tuned. The final test accuracy is 98.83%.

In human fall classification part, this thesis mainly focues on the influence of network structures. In total, two dataset are investigated on mainly three models. The three models are single-frame 2DCNN, multi-frame 2DCNN and RNN. In the first dataset, which is 5 classes out of 101 classes in UCF101, there is no big difference in the performance among three models. While in the second dataset, multiple camera fall dataset, multi-frame CNN and especially RNN provides much better result compared with single-frame 2DCNN. It shows the effectiveness of RNN model on classifying activities strongly correlated to time.

Future work

Due to time and computation limitation, there are lots things to be improved based on this thesis. The future work could be:

- Do hyperparameter search in traffic sign part to further optimize the result,
- Apply the model trained on GTSRB to real world application,
- Try 3DCNN on video classification dataset,
- Apply the three models used in video classification part to a much larger dataset.

Bibliography

[1] 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.

[2] Dan CireşAn, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[3] Xuehong Mao, Samer Hijazi, Raúl Casas, Piyush Kaul, Rishi Kumar, and Chris Rowen. Hierarchical cnn for traffic sign recognition. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 130–135. IEEE, 2016.

[4] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[5] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016.

[6] Ali Diba, Vivek Sharma, and Luc Van Gool. Deep temporal linear encoding networks. *arXiv preprint arXiv:1611.06678*, 2016.

[7] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[8] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[9] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages

4489–4497, 2015.

[10] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[11] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[12] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[15] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[16] T. Tieleman and G. Hinton. Neural networks for machine learning, 2012. Lecture 6.5 - rmsprop.

[17] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition, 2016.

[19] 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.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] Chris Ohah. Understanding lstm networks, 2015.

[25] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

[26] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[27] J.Meunier A. St-Arnaud J. Rousseau E. Auvinet, C. Rougier. Multiple cameras fall dataset. *Technical report 1350, DIRO - Université de Montréal*, 2010.