The University of Hong Kong

ELEC6604 Neural Networks,
Fuzzy Systems and Genetic Algorithms

Assignment 3

Lecturer : *Dr. G. Pang*

Student : *Zhao Yunqing*
UID: 3035541156

05/03/2019 Hong Kong

Contents

Part 1. Description of the Task

1.1 Retrospection and Setup New Tasks

What we have submitted, the Assignment 1 and Assignment 2, depicted the process of partition of the two classes of the points and the MNIST, respectively.

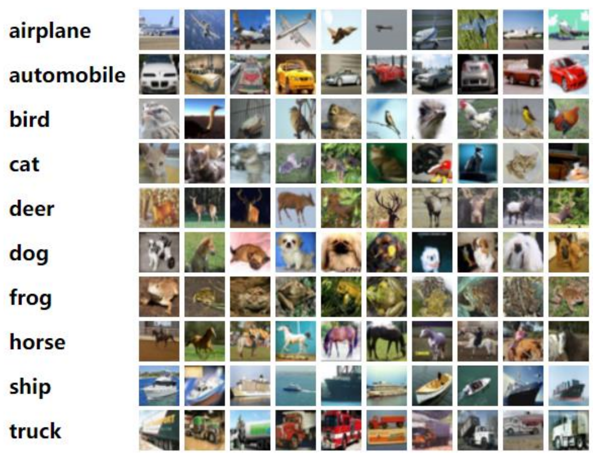
Hence, I think it is time to try to classify object images of a little bit complexity, with RGB channels rather than simply grayscale or single points . As for me, I finally decided to choose another famous dataset: **CIFAR-10**.

1.2 Introduction of the Dataset

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class(the same as MNIST). There are 50000 training images, including 10000 validation images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Following Graph 1 the classes in the dataset, as well as 10 random images from each:

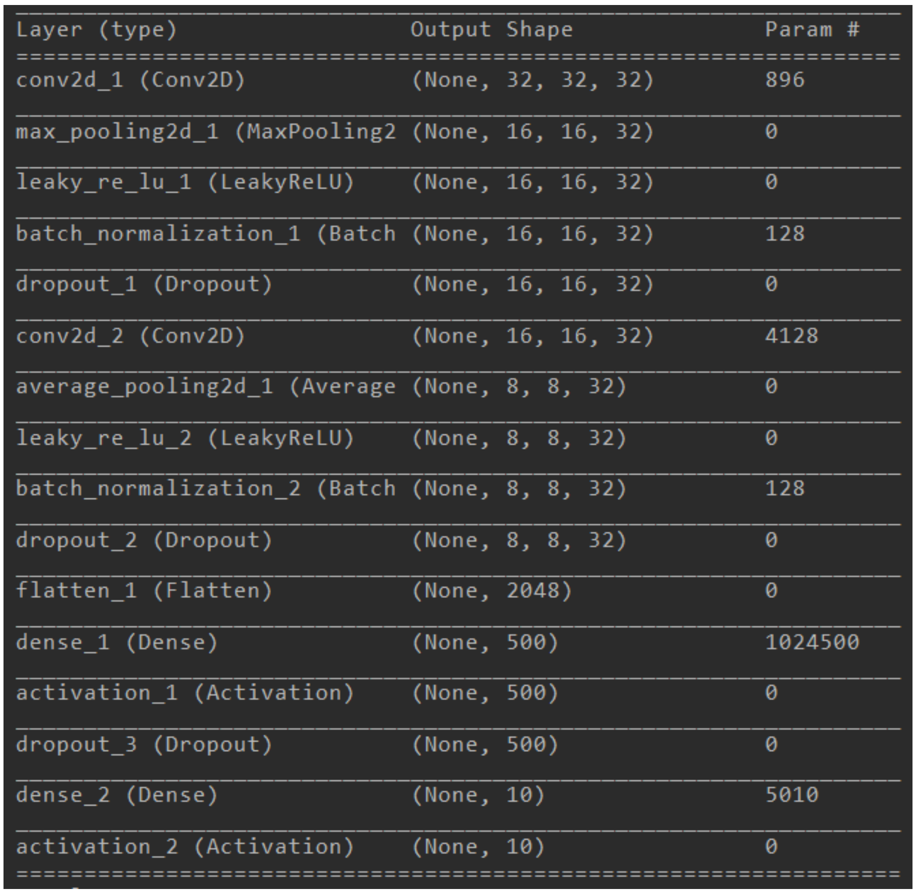


Graph 1 10 classes of images

Now, **the task is**: Using CNN to try to achieve the best performance of classification of the CIFAR 10 images.

Part 2 Build up My Network.

2.1 Primary Version Structure of my CNN



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 16, 16, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 32)        128
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        4128
_________________________________________________________________
average_pooling2d_1 (Average (None, 8, 8, 32)          0
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 8, 8, 32)          0
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 32)          128
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               1024500
_________________________________________________________________
activation_1 (Activation)    (None, 500)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5010
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
```

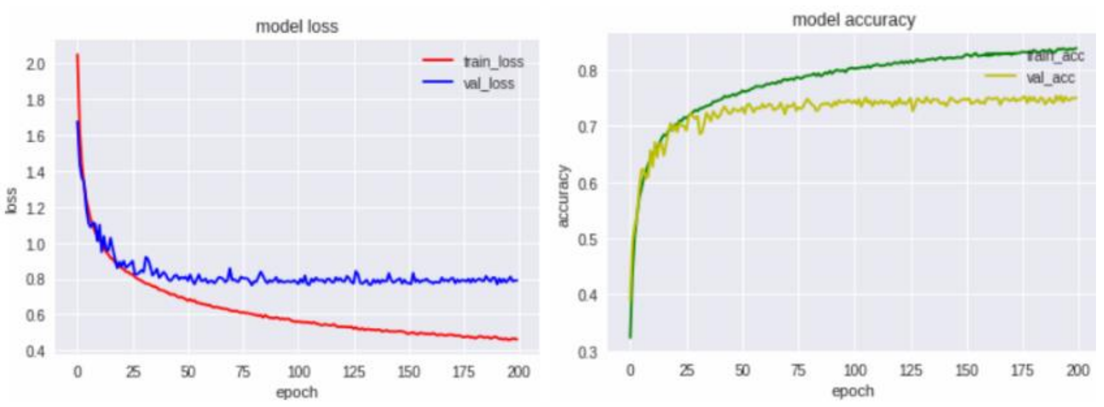
Graph 2 Layers of CNN (Primary Version)

Since CIFAR-10 is a more complex dataset, so we cannot use the normal kind of neural network. Even if the Convolutional neural network, we cannot use a single layer CNN, which would cause the underfitting by learning less features to achieve the better performance. ***As usual, for each network, normalization of the image and the one-hot label are the default settings.***

Actually, I chose several kinds of structure of CNN(Graph 2 as a primary one), and they showed the different performance about the CIFAR-10.

This primary version of CNN as above got the 95% accuracy on the training set, which means the network almost learn the all features of the images.

However, only 75% accuracy on the validation set(as Graph 3), which is a normal performance. It seems that I must make parameters tuning or refine my CNN structure.



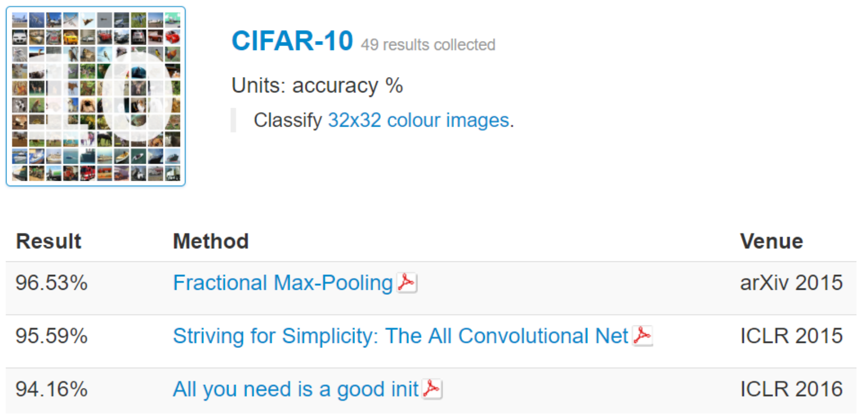
Graph 3 The Result of ***Primary Version*** of CNN

2.2 Simplified VGG-Like Network

The experiments proved that the constructed primary network is of no-good performance.

Dating back to the MNIST model training, it is very easy to get a 95% or above accuracy, hence I think the reason of this situation in priority is the complexion of the dataset, or the images. The MNIST images is of grayscale, and we only need to recognize the edge of the picture, by the way the size of the picture is 28*28. When compared to the CIFAR-10, it is apparent that the CIFAR-10 is more complicated, that's why this network is poor on that. For a deeper reason, I think it is also a kind of overfitting, because the high accuracy on training set and low accuracy on testing set.

As a solution, I refer to many past publications by some researcher who made great contributed on this work as follows:



CIFAR-10 49 results collected

Units: accuracy %

Classify 32x32 colour images.

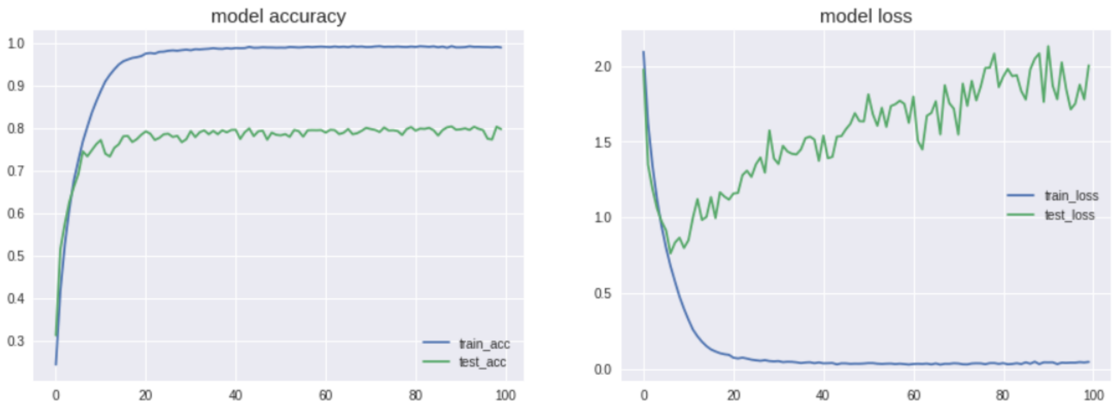
Result	Method	Venue
96.53%	Fractional Max-Pooling	arXiv 2015
95.59%	Striving for Simplicity: The All Convolutional Net	ICLR 2015
94.16%	All you need is a good init	ICLR 2016

Graph 4 ***State-of-art*** Structures of all the world

We can clearly see that the most state-of-art work can achieve the 96.53% on test set. Taken limited computational resources in consideration, I will refine the classical structure, VGG-16, by making it a simpler structure to make a **training time and accuracy Tradeoff**. The structure is as the Graph 4: (The convolutional neural network will be activated by "Relu" function)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        1792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 256)         590080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               524416
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
```

Graph 5 The structure of the 2nd Version (VGG-like)



Graph 6 Result of Simplified VGG

It is apparent that the result is better than the Primary one, by achieving the 80% accuracy on the test(or say validation here) set.

Up to now, I find that we may tune our neural network structure by referring some classical, and proved structure, which can make sure the reliability of the result, even though 80% is not a perfect result when compared by the primary version, however it brought some improvement.

Now, it is time to do something else. The above networks just bring the raw picture to the network; however, I did not preprocess the images by some **augmentation method.** In the next part, I will pose some optimization methods to help training of my network.

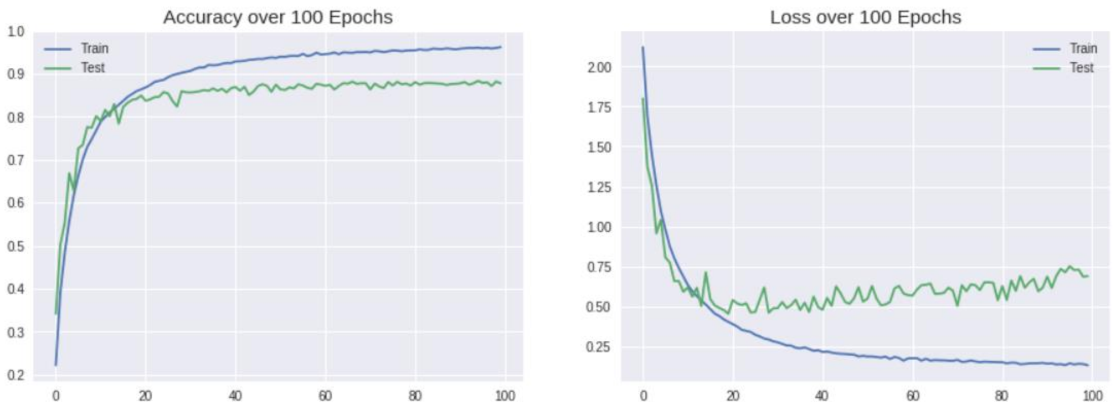
Part 3 Tuning with network

3.1 Network with Image Augmentation

Why this part? Because the raw picture is non-augmented. By using the Keras image-generator, we may consider using some process method to the images before the input to my neural network, which will potentially improve the prediction result.

The image augmentation contains several kinds of image processing methods. In this case, I applied the height/width_shift_range and the horizontal method to refine the input images.

Note that I do not recommend for the vertical flip, since the image seems to be normal by horizontal flip, but there is no need to let network see the vertical flipped figure, by which the pictures maybe strange and lose the original features. This can also be clarified from real eyes by human.



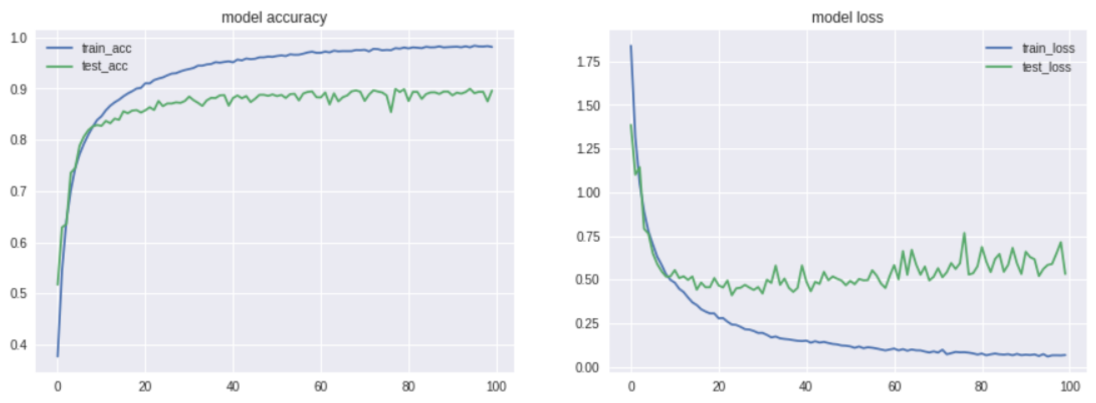
Graph 7 Image augmentation result

As the Graph 7 shows in the following, the result shows the image augmentation is ***useful***, which improved the validation accuracy by 9%, from 80% to the 90% around. As a conclusion, I think the situation is by letting the neural network model to learn from the more general pictures, which can enhance the effect of generalization of this VGG-like model.

3.2 Batch Normalization

Batch normalization is a kind of layers in the neural network. Because the computational result after the non-linear activation layer maybe too big or small to cause the gradient vanishment, so we can imagine that this layer can help the value shrink in a small interval, letting the gradient keep stable.

The result showed that the time for convergence is faster than the previous version, and the accuracy rate is even better than that of part 2.3, the accuracy of which is around 89.6%. Hence, we may consider using the layer of Batch Norm in the next part.



Graph 8 The training process with Batch Norm Layer

Up to now, the final structure of the CNN network has been decided, since the too many layers, please refer to the appendix

3.3 Changing batch size

Batch Size	Test Accuracy
32	89.27%
64	**91.39%**
128	91.06%
256	89.60%

Table 1 Testing Accuracy by different batch size

After the decision of the whole neural network, I changed the batch size to tune for the better improvement potentially.

We can clearly see from the Table 1 that, the best performance based on the VGG-Like network with Batch Normalization can be achieved when I set the batch size as 64.

There is another conclusion of the batch size, the bigger batch size, the less possible of the overfitting, however the slower convergence speed. The smaller batch size, the faster convergence speed. However, the extremely situation also caused the bad result.

Based on my observation on the Graph 6-8, the loss will grow gradually by increasing the epoch time. And also, the final accuracy rate is not the best one.

Intuitively, I regard it as a kind of Overfitting, so I should use some method to eliminate this happening. See the 2.4.

3.4 Early Stopping

Normally, the simplest way to solve the problem is to just down the number of epochs.

However, in this assignment, in order to achieve the better and accurate result in a smart way, I use a special API of Keras, named “Early Stopping”, which can monitor the training process to prohibit the overfitting, once the accuracy of validation set become lower in an apparent trend, it will stop training and output the optimized result.

This API belongs to the “Callback” when we execute the command model.fit. The result shows that the network stopped training at the 63th epoch. Since I set the patience interval is 20 epochs, so this result means the best result acquired at the 43th epoch of the training process, which can prohibit the overfitting, and the more time needed for training.

In the real experiment, you can set the value of patience as you like. E.g. I set the monitor = val_loss here, it means if the val_loss did not go down in the next 20 epochs, the training process will be stopped. So, you can do whatever you like to make it customized.

3.5 some other tuning methods

3.5.1 Dropout Layer

Sometimes we need to drop some of the weights randomly, to prohibit the overfitting, to use fewer computational resource as well.

In this case I only use the last layer to drop out 50% of the weights randomly, the result shows a little improvement, so I will not dig out deeper on this in my assignment 3 in this case.

By the way, since I research on this in the assignment 2 in a deeper detail, base on that experience, I think it is better to apply less probability on the dropout operation, otherwise the underfitting situation may appear, and I believe that everyone isn’t willing to see it.

3.5.2 Weight matrix Initialization

In this assignment, I use the “he_normal” and random normal distribution method to initialize my network. In the latter case, I set the standard deviation from 0.001 to 0.1 in different layers orderly, letting the characteristic of network being more elasticity.

However, the result shows that there is not big difference between 2 methods, but the “he_normal” performs better in a little degree. Hence, I decide to choose this way.

3.5.3 Number of Epoch

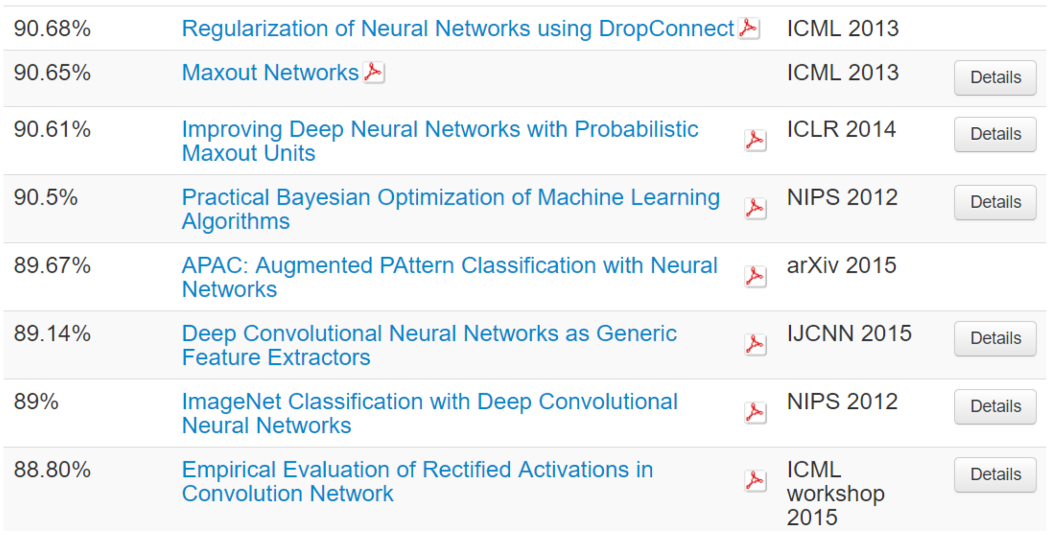
Initially, I chose the number of epoch as 200, but I found the no-grow effect on the training process, and even growing on loss value, so I adjust it to 100 as epoch time.

Part 4 Results and Discussion

4.1 Results of the Assignment 3

The above learning and training process show the improvement of my neural network model in a gradually way, by changing the structure, and the tuning of the parameters. The results show that my work has a positive result, encouraging me to do further exploration in the future.

Actually, from the Graph 9 as below, the World's top-class neural network models have achieved great performance, on the same dataset, CIFAR-10. However, my own work in this assignment is not worse than that of them. (90% accuracy on the testing set of my network, as a comparison, the world's top 1 accuracy is ***96.5***%, by using Fractional Max-Pooling)



90.68%	Regularization of Neural Networks using DropConnect	ICML 2013	
90.65%	Maxout Networks	ICML 2013	Details
90.61%	Improving Deep Neural Networks with Probabilistic Maxout Units	ICLR 2014	Details
90.5%	Practical Bayesian Optimization of Machine Learning Algorithms	NIPS 2012	Details
89.67%	APAC: Augmented PAttern Classification with Neural Networks	arXiv 2015	
89.14%	Deep Convolutional Neural Networks as Generic Feature Extractors	IJCNN 2015	Details
89%	ImageNet Classification with Deep Convolutional Neural Networks	NIPS 2012	Details
88.80%	Empirical Evaluation of Rectified Activations in Convolution Network	ICML workshop 2015	Details

Graph 9 Accuracy of the world's top-class network

4.2 Discussion

The whole learning process shows the strength of different structure of the neural network, indicating that we can not make the network deeper and deeper, and we cannot make the number of the epochs bigger and bigger, which will introduce the phenomenon of overfitting, even causing the Gradient vanishment and explosion.

The Batch Normalization means transformation of the activated value in a small region, which can keep the gradient stable.

Image augmentation posts an apparent improvement on the accuracy rate about the network, directly from 80% to 88%, which means the higher level of generalization, by changing some angles, or denoise of the images.

In the final stage I used the Early-Stopping of the fit process, to prohibit the overfitting, extracting the most optimized model as the output. The experiment also proved that this operation can shorten the training time by stopping at the right place.

By adjusting the value of patience in Early-Stopping, maybe the better result can be achieved. Limited by the number of epoch in this assignment I chose 20 as a compromised value.

Part 5 Reference and the summary

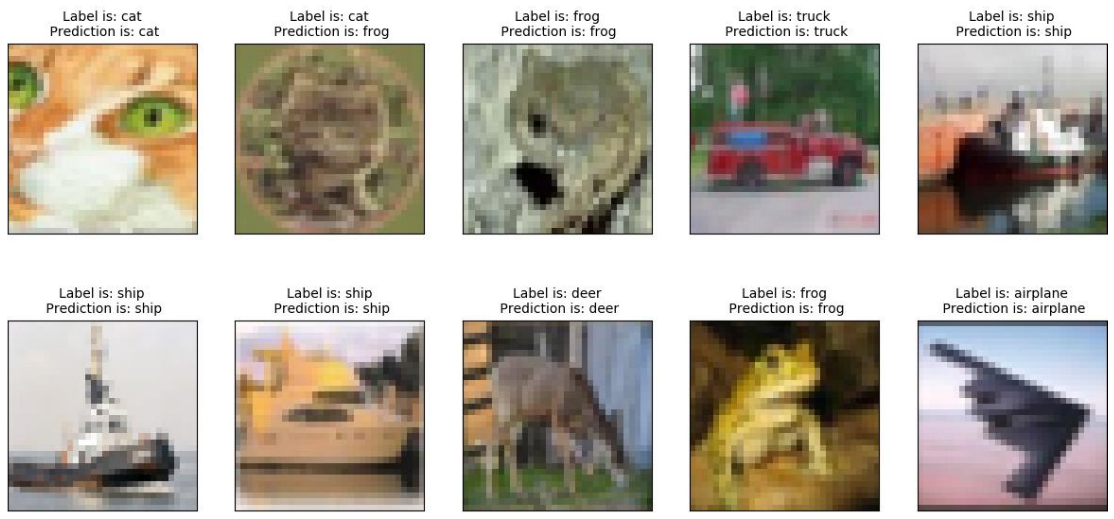
5.1 Summary Table

Classification task	CIFAR-10
Objective of investigation	Airplane, bird, cat, deer, dog, etc..
Image size, Color or grayscale	RGB images with the 32*32 size
Reference/source of the images	https://www.cs.toronto.edu/~kriz/cifar.html
Number of classes	10
Available Images per class	5000 for train, 1000 for test(validation)
Number of training images	50000
Number of testing images	10000
Structure of CNN	Please see the Appendix
Parameters of variation	Batch-size, Weight initialize, optimizer, epoch, learning rate, filter number, kernel size, strides, pool size, activation function, dropout rate, number of dense units
Training time	1794.62s
Testing accuracy	91.39%
Software platform	Keras
Hardware platform	Google Colab GPU / Remote GPU Server
Discussion of results	See the Part 3/4
Observations and Comments	See the Part 3/4

Table 2 Summary Table

5.2 Visualization

In this part, I will pose some examples from the dataset, and give my prediction result. The result is as follows:



Graph 10 Comparison of prediction result and True Label

As the above picture, the model can classify all the 10 samples randomly chosen from the test set, which matches the 91.39% accuracy with result before. The prediction results fully verified our experiment success.

5.3 Reference

[1] https://arxiv.org/abs/1412.6071

[2] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[3] https://arxiv.org/abs/1511.06422

[4] https://keras-cn.readthedocs.io/en/latest/other/initializations/

[5] https://keras.io/callbacks/#earlystopping

[6] https://blog.csdn.net/ukakasu/article/details/80089866

[7] https://blog.csdn.net/zeuseign/article/details/72773342

[8] https://arxiv.org/abs/1512.03385

[9] https://arxiv.org/abs/1511.02583

[10] Jia-Ren Chang, Yong-Sheng Chen, Batch-normalized Maxout Network in Network, arXiv, 2015

[11] https://arxiv.org/abs/1412.6830

[12] https://www.cs.toronto.edu/~kriz/cifar.html

5.4 Appendix: Final Network Structure

```
Layer (type)                Output Shape              Param #
=================================================================
input 8 (Input Layer)      (None, 32, 32, 3)           0
conv2d 43 (Conv2D)          (None, 32, 32, 64)        1792
batch normalization 43     (None, 32, 32, 64)        256
                                                          ________
conv2d 44 (Conv2D)          (None, 32, 32, 64)       36928
batch normalization 44     (None, 32, 32, 64)         256
max pooling2d 22            (None, 16, 16, 64)         0
                                                          _______
conv2d 45 (Conv2D)           (None, 16, 16, 128)       73856
batch normalization 45      (None, 16, 16, 128)        512
conv2d 46 (Conv2D)           (None, 16, 16, 128)      147584
batch normalization 46      (None, 16, 16, 128)        512
max pooling2d 23            (None, 8, 8, 128)           0
                                                           ______
conv2d 47 (Conv2D)           (None, 8, 8, 256)        295168
batch_normalization_47      (None, 8, 8, 256)          1024

conv2d 48 (Conv2D)           (None, 8, 8, 256)        590080
batch normalization 48      (None, 8, 8, 256)          1024
max pooling2d 24            (None, 4, 4, 256)           0

flatten 8 (Flatten)           (None, 4096)              0
dense 15 (Dense)              (None, 128)             524416
dropout 8 (Dropout)            (None, 128)              0
dense 16 (Dense)              (None, 10)               1290
=================================================================
```