EE569 HW#5

Student Name: Shi-Lin Chen
Student ID: 2991911997
Student Email: shilinch@usc.edu
Submission Date: April 07, 2020

Problem 1: CNN Training on LeNet-5

1.1 Motivation and Abstract

In recent years, Convolution Neural Network (CNN) has brought a huge improvement in the field of Computer Vision, such as image analysis and object recognition. In this assignment, we will train a simple CNN called the LeNet-5 and apply it to the CIFAR-10 dataset.

1.2 Approach and Result

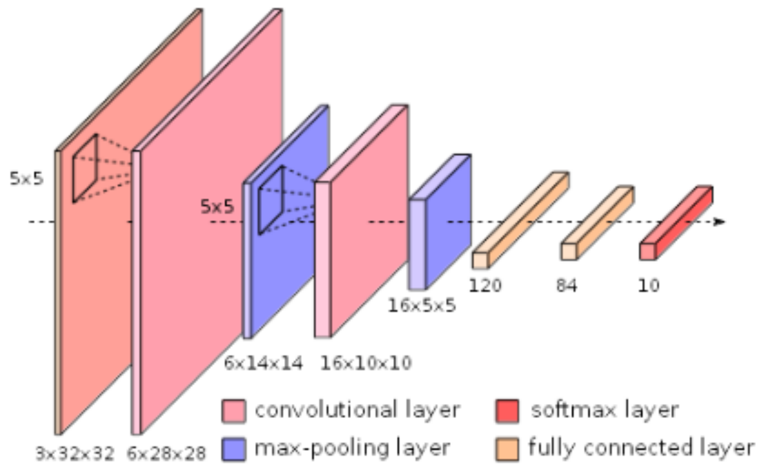
LeNet-5

In this problem, we have already given the architecture of CNN, which contains all necessary parts for CNNs, convolutional layer, max-pooling layer…and so on. We will use this architecture to train

CIFAR-10

CIFAR-10 dataset consists of 60,000 (a labeled training set of 50,000 images and a test set of 10,000 images) RGB 32*32pixel images in 10 classes. In this problem, we will use the given CNN architecture to train on this dataset.

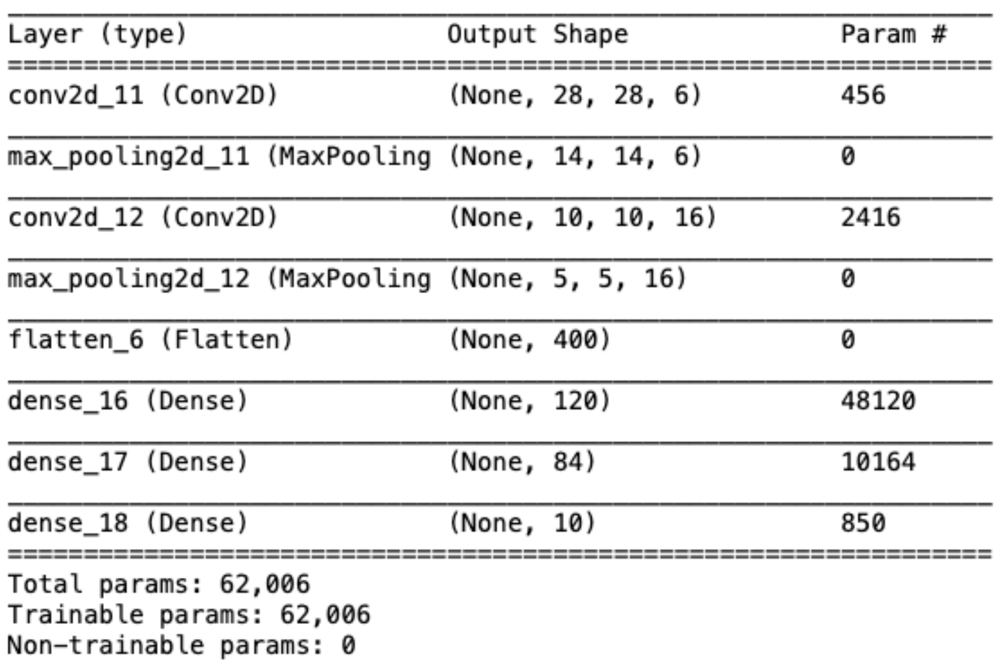
I implement this problem using Keras on Jupyter Notebook (Anaconda). First, import the library we need, like **numpy** and **Keras dataset**. Second, we need to load the training data and test data from Keras dataset and create the CNN with given architecture:



Once we give the parameters like the batch of size, epoch, learning rate and so on, then we could start training. In part b, we need to use 5 different parameter settings to train in order to observe the difference of performance. (The initialization and modification of parameter set will be provided later in result part.) After training 5 time with different parameter sets, choose the best performance and plot the performance curve.

Training Result:

Network Summary:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_11 (Conv2D)           (None, 28, 28, 6)         456       
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 14, 14, 6)         0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 10, 10, 16)        2416      
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 5, 5, 16)          0         
_________________________________________________________________
flatten_6 (Flatten)          (None, 400)               0         
_________________________________________________________________
dense_16 (Dense)             (None, 120)               48120     
_________________________________________________________________
dense_17 (Dense)             (None, 84)                10164     
_________________________________________________________________
dense_18 (Dense)             (None, 10)                850       
=================================================================
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
```

Parameter Setting:

Original Parameter setting:

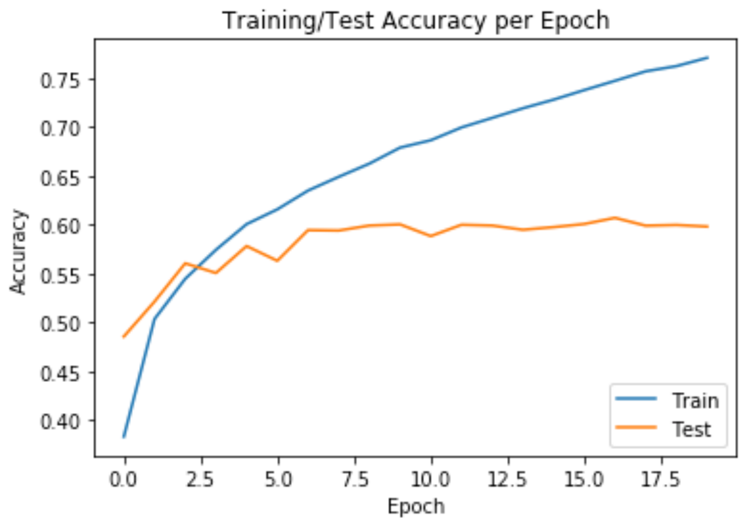
➢ Base Model (setting#1) – given Architecture using:

20 epochs, **5*5** kernel, **20** batch size, MaxPooling, ReLU Activation

SGD optimizer (learning rate = **0.01**, decay = **0.0001**, momentum = **0.85**)

Training Accuracy = 0.7711

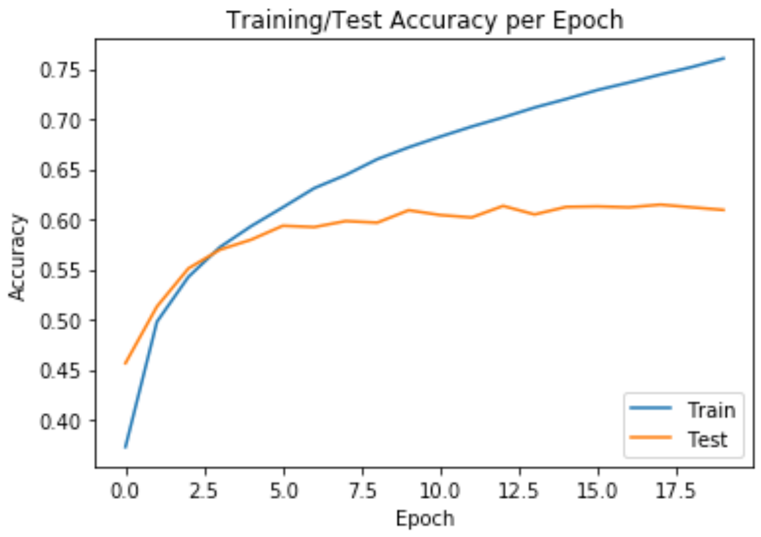
Test Accuracy = 0.6072



➢ Setting#2: Changing batch size from 20 to 50

Training Accuracy = 0.7606

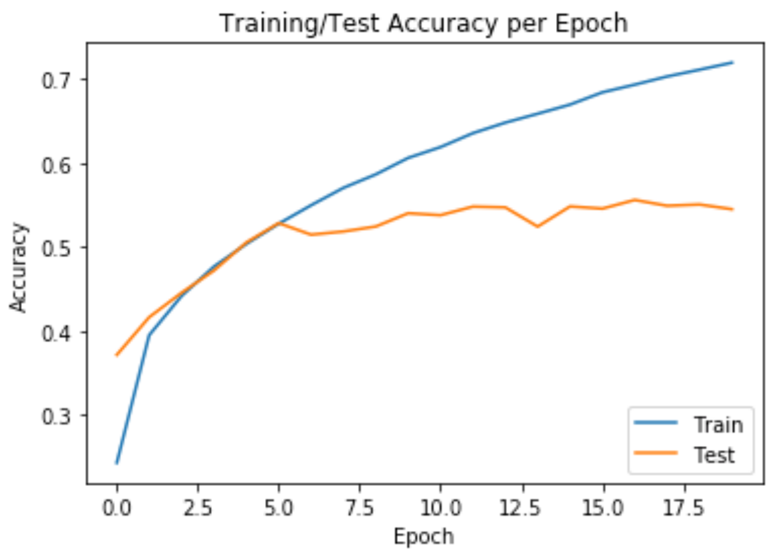
Test Accuracy = 0.6147



➢ Setting#3: Changing learning rate from 0.01 to 0.04

Training Accuracy = 0.7190

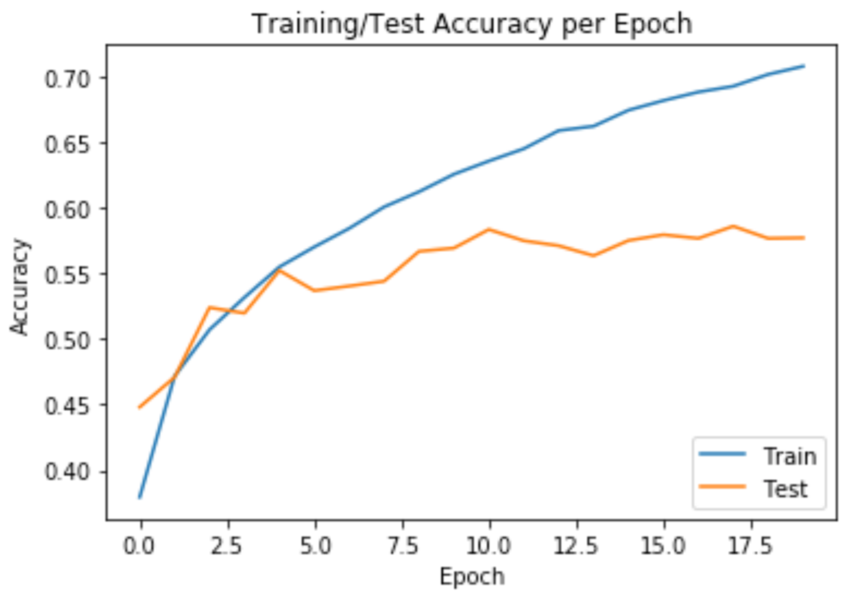
Test Accuracy = 0.5557



➢ Setting#4: Changing decay from 0.0001 to 0.00001

Training Accuracy = 0.7083

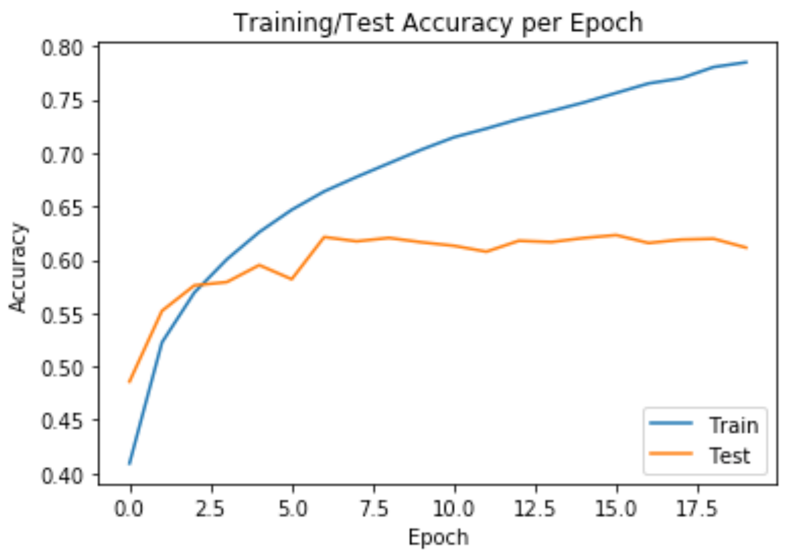
Test Accuracy = 0.5862



- Setting#5: Changing optimizer from Stochastic Gradient Descent to Adadelta (with default parameters setting) – **best result**

Training Accuracy = **0.7849**

Test Accuracy = **0.6230**



Overall:

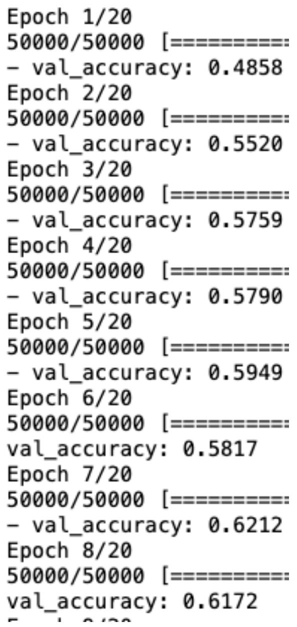
Train Accuracy Mean: 0.7487

Test Accuracy Mean: 0.5974

Observation:

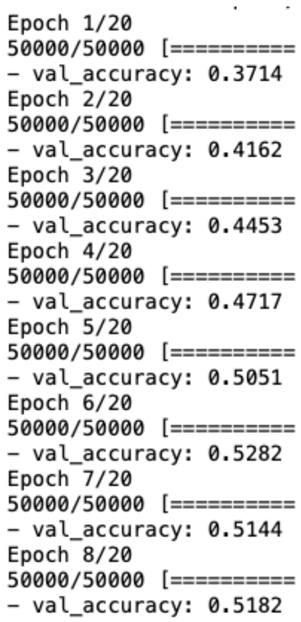
In the parameters sets, I found that adjusting learning rate could influence the accuracy a lot (more than other parameters changing). On the other hand, changing batch size does not really affect the accuracy in this case and lowering the decay might decrease the accuracy a little bit. I also found out that changing optimizer could cause a different result during training. When using Adadelta optimizer instead of SGD, the accuracy grows faster during the training, compare to other 4 sets using SGD. The final accuracy in training part and testing part is also better when using Adadelta optimizer. We could see the image below, that when observing the training after 8 epochs, the test accuracy using Adadelta go from 0.4858 to 0.6172, on the RHS, the test accuracy using SGD go from 0.3714 to 0.5182.

Since the problem ask us to plot the performance curve of both training and testing data, I use testing data as validation data during training, in order to see the accuracy curve of testing data in the view of epochs.



```
Epoch 1/20
50000/50000 [=========
- val_accuracy: 0.4858
Epoch 2/20
50000/50000 [=========
- val_accuracy: 0.5520
Epoch 3/20
50000/50000 [=========
- val_accuracy: 0.5759
Epoch 4/20
50000/50000 [=========
- val_accuracy: 0.5790
Epoch 5/20
50000/50000 [=========
- val_accuracy: 0.5949
Epoch 6/20
50000/50000 [=========
val_accuracy: 0.5817
Epoch 7/20
50000/50000 [=========
- val_accuracy: 0.6212
Epoch 8/20
50000/50000 [=========
val_accuracy: 0.6172
```

using Adadelta optimizer



```
Epoch 1/20
50000/50000 [==========
- val_accuracy: 0.3714
Epoch 2/20
50000/50000 [==========
- val_accuracy: 0.4162
Epoch 3/20
50000/50000 [==========
- val_accuracy: 0.4453
Epoch 4/20
50000/50000 [==========
- val_accuracy: 0.4717
Epoch 5/20
50000/50000 [==========
- val_accuracy: 0.5051
Epoch 6/20
50000/50000 [==========
- val_accuracy: 0.5282
Epoch 7/20
50000/50000 [==========
- val_accuracy: 0.5144
Epoch 8/20
50000/50000 [==========
- val_accuracy: 0.5182
```

using SGD optimzizer

1.3 Discussion

Part(a.) CNN Architecture

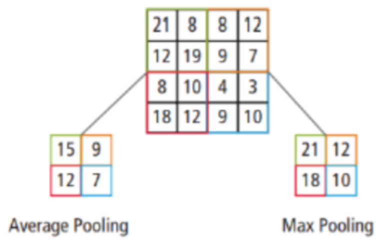
1. Explain different layer in CNN

<u>Convolutional layer:</u>

The convolutional layer is the first layer of CNN and it could work on the input to the network. The main purpose of this layer is to extract image features. Each layer has its filter to extract local features from its input, like the convolution process in image processing. It scans a few pixels (here is 5*5*3 RGB images) at a time and create a feature map. If the input has multiple channel, then there are also multiple filters for all input channel in convolution layers.

<u>Max-pooling layer:</u>

Pooling layer is like down sampling in signal processing, it will reduce the amount of the information in each feature which obtained in convolutional layer while maintaining the most important information. That is, Pooling layer is like a filter to be applied to feature map we got in convolutional layer, and its output is a feature set with less dimensions. This process could increase the speed of calculation. Max-pooling layer is one sort of pooling layer, it picks the maximum value within its scanning window.

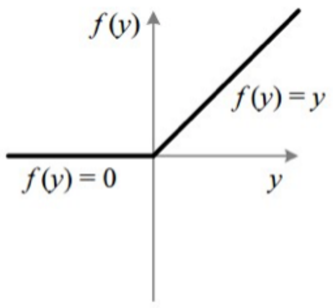


Fully connected layer:

It is a layer that connects to inputs and outputs (for example, in CNN, it could be different layers as input and output). This layer takes the input value(features) and outputs the result of N dimensional vector where N is the number of classes (for example, N = 10 in CIFAR-10 dataset). Basically, this layer could look at what features most correlate to which class and have some particular weight that we could compute the products between weight and previous layer.

Activation function:

Activation function is the function that we use to get the output of node. The reason why we need activation function is that we could apply non-linear transformation for the data. If we don't apply activation function, the output would simply be some linear function which is just a polynomial of one degree. This might influence training process a lot due to their limitation of complexity. There are many different kinds of activation function like Sigmoid, Tanh and leaky ReLU. ReLU is one of the most commonly used activation functions nowadays, its advantage is that it doesn't activate all neurons at the same time.



Softmax function:

Softmax function is used in the final step of the classification problem. It could normalize an input value into a vector of values between 0~1 that follows the probability distribution, and their total sums is 1. Since Softmax function could convert the input value into probabilities, it could help us to pick the particular class by selecting the largest probability.

2. Overfitting issue and the method to avoid it

Overfitting is a machine learning issue that even the accuracy in training set is really good (let's say accuracy = 99%), but when the run our model on new unseen dataset, accuracy might not be as high as the accuracy in training data. Basically, it means that the model has almost memorize the training data and the model can not generalize and adapt to any new information.

There are many ways to avoid overfitting, such as cross-validation, train with more data and regularization. One of the easiest ways to avoid overfitting is called early stopping, that is, don't run too many iterations in the training process, for example, stop the training when the accuracy does not change.

3.

For image classification, people usually design some filters that could match objects in class needed, which is also the way traditional method do. However, filters designed by human can't not be complicated enough to fit all kinds of different objects or images. While CNN is data driven, it learns the filters from labeled data automatically, with some adjustable parameters so that people could learn and design. Through this learning process, CNN could better classify common patterns of the object in classes than traditional method.

4. Explain the loss function and BP optimization.

Loss function is a function that could measure the difference between the network output and expected output. Therefore, the lower the loss is, the better the network performance. If we could minimize the lost, we could get a better trained network. Backward Propagation (BP) is an efficient way to adjust parameters (or weights) in order to get lower loss in machine learning. It adjusts the weight of neural net based on the error rate (lost in this case) obtained in the previous epoch. Since there are many layers in the model, we need to implement gradient descent through chain rule. This proper tuning of the weight could lower the lost and increase the generalization of the model.

Part(b.) CIFAR-10 Classification

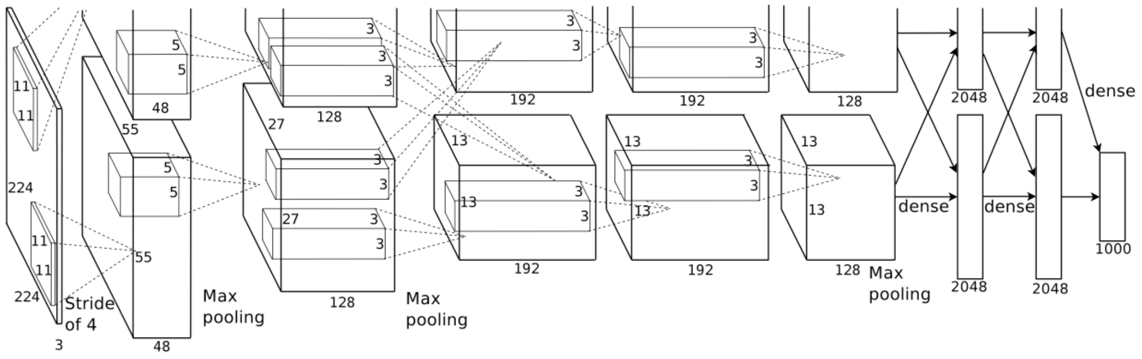
Answers are provided in result part above!

Part(c.) State-of-the-Art CIFAR-10 Classification

I choose the paper ***ImageNet Classification with Deep Convolution Neural Networks*** published by Alex Krizhevsky from University of Toronto. **[1]**

1.

In this paper, author use a different Neural Network called Deep Convolution Neural Networks. The architecture is showed below:



Architecture of CNNs (source from the paper**[1]**)

It contains eight learned layers – five convolutional layers and three fully connected layers, which is different compared with the architecture of LeNet-5.

The author used **ReLU Nonlinearity, Training on Multiple GPUs, Local Response Normalization** and **Overlapping Pooling** to improve traditional CNN**.**

ReLU and **Training on multiple GPUs:**

These two could increase the speed when training large dataset, the author states that “faster learning has a great influence on the performance of large models trained on large datasets.” The author found out that ReLU function could speed up the training, and people use ReLU a lot in training after Alex published the paper. (LeNet-5 originally used tanh sigmoid function during the training.) The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

Local Response Normalization:

In this paper, they use a form of normalization called “Local Response Normalization”. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating a competition for big activities amongst neuron outputs computed using different kernels. It is a kind of “brightness normalization” since the normalization do not subtract the mean activity. They verify that using this normalization could reduce 2% of error rate when training on CIFAR-10.

Overlapping Pooling:

There is another method introduced in this paper, which is **Overlapping Pooling**. Pooling layer could summarize the outputs of neighboring groups of neurons and it is commonly used in many CNNs. They state that models with overlapping pooling is slightly more difficult to overfit during training.
They also introduce data augmentation and dropout to reduce overfitting.

2. Compare with LeNet-5, cons and pros

After reading this paper, I think the main reason that they could improve the training is the architecture of their CNN. First of all, there are more layer compared to LeNet-5, they also initialized the weights in each layer from zero-mean Gaussian distribution with standard deviation 0.01 and initialized the neuron biases in both convolutional layers and fully connected layers with constant 1. They also manually adjusted the learning rate for all layers when validation error rate stopped improving with the current learning rate.

LeNet-5:
Pros:
Easy to implement like less layers.
Cons:
Lower accuracy when training on CIFAR-10

Deep CNN:
Pros:
Higher accuracy rate.
Cons:
Difficult to implement, there are more layers in this CNNs and the normalization is also more difficult to understand.
Slower speed compared to LeNet-5.

Reference:

[1] Alex Krizhevsky. ImageNet Classification with Deep Convolution Neural Networks