Convolutional Neural Network

Contents

Overview

In this report, I will build 5 convolutional neural networks to do the image multi-categorical classification based on CIFAR-10 dataset. The CIFAR-10 is a collection of images that are commonly used to train machine learning and computer vision algorithms. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class. (Eckersley, 2017)

For each model I constructed, I recorded its running time, minimum training loss, minimum test loss, maximum training accuracy, maximum test accuracy, and optimal epoch time.

Base network

(part2: with at least three convolutional layers and one dense layer)

Description of Modeling and Training Procedures

1. I import the necessary libraries such as keras.
2. I divide the dataset "cifar10.load_data()" into training dataset (x_train,y_train) and test dataset (x_test,y_test).
3. I scale the both x_train and x_test by its maximum number, 255.
4. I convert the response y_test and y_train into categorical variables.
5. As follows, I build a model with three convolutional layers for feature extraction and three max pooling layer for downsizing. In addition, I use flatten layer to transform the filter maps and then use two additional dense layers before the output dense layer with a dimension of 10, corresponding to the number of dummy variables of y. Specifically, I use a 3*3 convolutional patch for each layer. For activation function, I used 'relu' to turn negative values to zero.

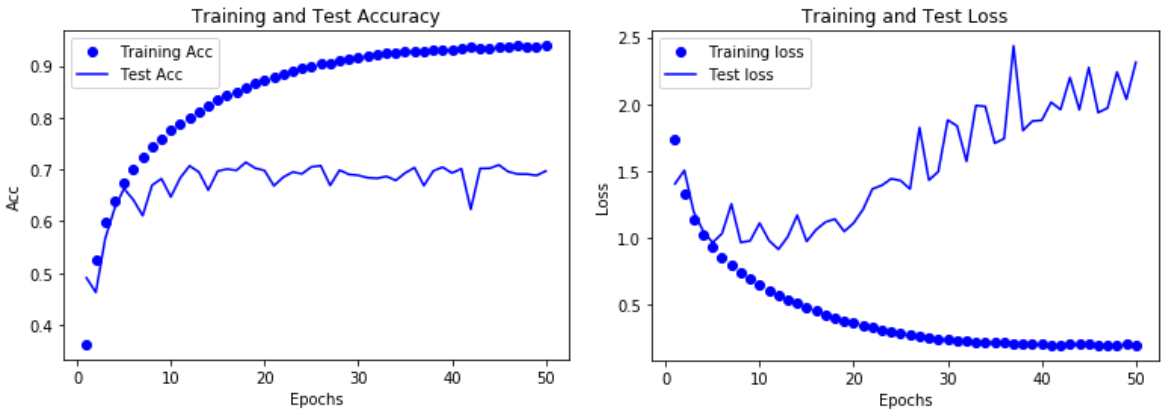
```
network=models.Sequential()
network.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Flatten())
network.add(layers.Dense(128, activation='relu'))
network.add(layers.Dense(64, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))
network.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_16 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_17 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_18 (Dense)             (None, 10)                650
=================================================================
Total params: 98,122
Trainable params: 98,122
Non-trainable params: 0
```

6. To compile the model, I used 'rmsprop' as the optimizer. Moreover, since this is a multi-categorical classification problem, I used accuracy as metrics and categorical_crossentropy as loss function.
7. Then I fit the model with 50 as epoch number and 64 as batch size.
8. Finally I plot the training and testing loss/accuracy chart.

Plotting charts

For both training dataset and test dataset, the accuracy and loss are plotted as follows.



Comment on performance

```
Epoch 50/50
50000/50000 [==============================] - 7s 132us/step - loss: 0.1828 - acc: 0.9425 - val_loss: 2.0825 - val_acc: 0.7050
Time required for training: 0:05:39.584591
```

It takes 5min40s to run. This is the baseline of other models.

As shown in the plot above, when the epoch number reaches 13, the test loss reaches its lowest number and after that, the test accuracy will not increase further. This fact shows that after epoch number reaches 13, the model becomes overfitting.

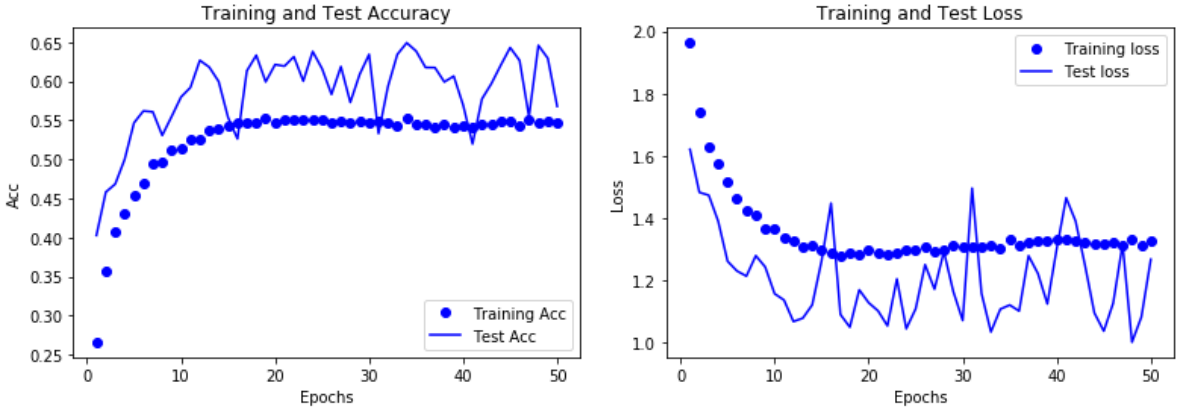
The lowest test loss is around 0.9 and the highest test accuracy is above 70%.

Since overfitting occurs here, the highest training accuracy could be 94% and the lowest training loss is around 0.1.

Preprocessing

(part3: include augmentation)

Plotting charts



Comment on performance

```
Epoch 50/50
782/781 [==============================] - 10s 13ms/step - loss: 1.3443 - acc: 0.5415 - val_loss: 1.6585 - val_acc: 0.5500
Time required for training: 0:08:14.664867
```

It takes 8min 15s to run, which is 3 minutes longer than the base model. This is because augmentation procedure is used here and more transformed images are used when training the model.

As shown in the charts above, when the epoch number reaches 20, the performance of both training dataset and testing dataset will not be improved. Moreover, the training accuracy even reduces a little bit when the epoch number becomes larger. This is because the augmentation procedure generates different training images based on the original training images. As a result, overfitting is avoided.

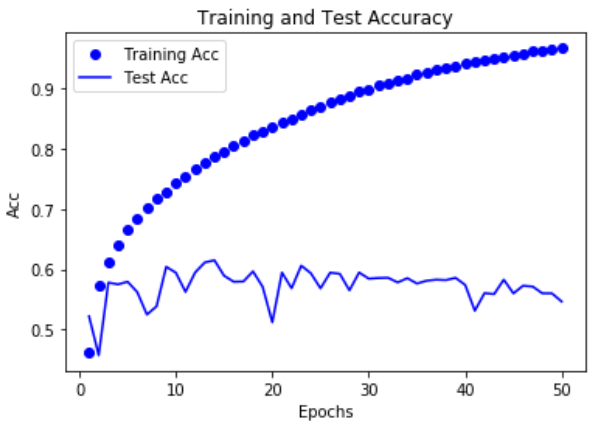
The lowest test loss is around 1.02 and the highest test accuracy is above 65%.

Since overfitting is avoided here, the highest training accuracy is about 55% and the lowest training loss is around 1.28.

Model with only one-hidden convolutional layer

(part 4: only one hidden convolutional layer)

Plotting charts



Comment on performance

```
Epoch 50/50
50000/50000 [==============================] - 5s 103us/step - loss: 0.1147 - acc: 0.9680 - val_loss: 2.6985 - val_acc: 0.5558
Time required for training: 0:04:15.923652
```

It takes 4min 15s to run, which is 1 minute shorter than the base model because a simpler model with fewer layers is used here.

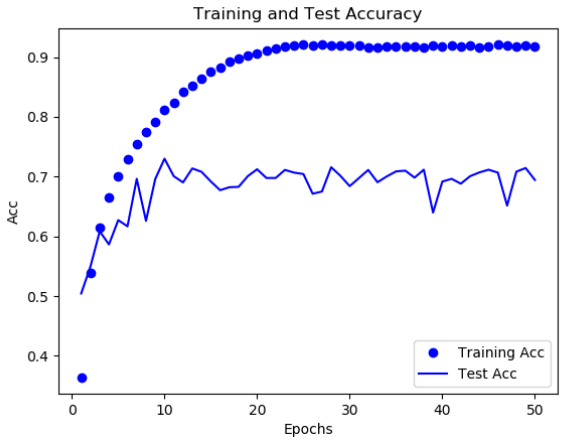
As plotted above, when epoch number increases, training accuracy continuously grow and training loss continuously decrease. This is the proof of overfitting. The largest training accuracy here is 96% but it could be higher and the minimum training loss here approaches 0.1.

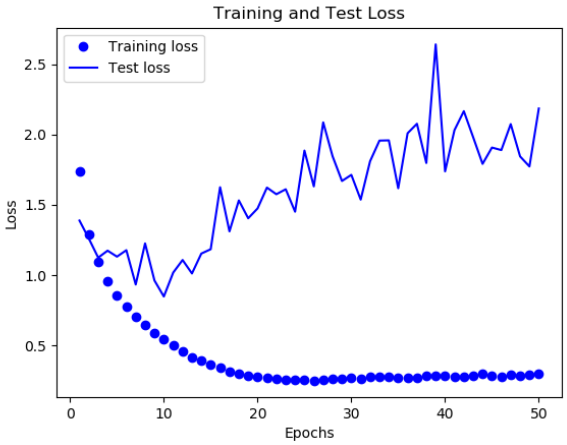
On the other hand, since only one convolutional layer is used here, the overall result of testing data is obviously worse than the base model. The minimum test loss is 1.21, which is larger than the test loss of base mode, which is 0.9. Moreover, the best test accuracy is 62%, which is lower than the accuracy of the base model, which is 70%.

Model with one more hidden layer

(part 5: one more hidden layer than base network)

Plotting charts





Comment on performance

```
Epoch 50/50
50000/50000 [==============================] - 8s 151us/step - loss: 0.2977 - acc: 0.9170 - val_loss: 2.1859 - val_acc: 0.6944
Time required for training: 0:05:53.345491
```

It takes 5min 53s to run, which is a little bit longer than the base model, which is 5 min 40s. This is due to an additional convolutional layer. However, the increase of time is insignificant.

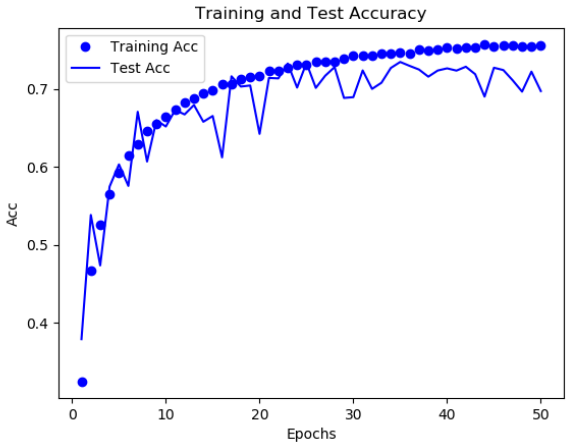
As shown above, again, overfitting occurs. Highest training accuracy reaches 93% while lowest training loss 0.17. The optimal epoch number is 11 where we can get the lowest test loss, which is 0.8, and the highest test accuracy, which is 72.5%.

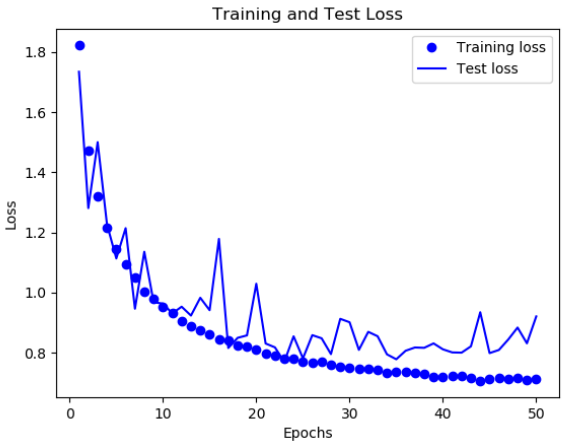
This result is a little better than the performance of the base model. This shows that an additional convolutional layer does extract more useful information.

Model with Dropout

(part 6: add dropout to base network)

Plotting charts





Comment on performance

```
Epoch 50/50
50000/50000 [==============================] - 6s 130us/step - loss: 0.7133 - acc: 0.7557 - val_loss:
0.9211 - val_acc: 0.6976
Time required for training: 0:05:24.863247
```

It takes 5min 25s to run, which is 16 seconds shorter than the base model. This is due to the dropout layer, where we abandon some extracted information on purpose to avoid overfitting. Therefore, less calculation is involved and the algorithm is a little bit faster.

As shown above, the training performance is very close to testing result since overfitting is somewhat avoided by the dropout layer. The highest training accuracy is 76% while the highest testing accuracy is 72.8%. The lowest training loss is 0.7 while the lowest testing loss is 0.8.

This model is obviously superior than the base model. First of all, it runs faster. Second, it achieve a higher test accuracy and a lower test loss while avoiding overfitting. However, the optimal epoch number is substantially larger than the other models because the learning process is slower due to the dropout layer.

Conclusion

The results of the five models above are summarized as follows (red represents worst result of a model comparable to others while green shows the best result of model comparable to others).

	Model1: Base	Model2: Preprocessing	Model3: Only one layer	Model4: One more layer	Model5: Dropout layer
Running Time	5min 40s	8min 15s	4min 15s	5min 53s	5min 25s
Min training loss	0.1	1.28	0.1	0.17	0.7
Min test loss	0.9	1.02	1.21	0.8	0.8
Max training acc	94%	55%	96%	93%	76%
Max test acc	70%	65%	62%	72.5%	72.8%

Best Epoch number	13	20	13	11	35

Reference

Chollet, F. (2018). *Deep Learning with Python.*

Eckersley, P. (2017, 6 12). Measuring the Progress of AI Research.

Appendix

Appendix A: Codes of Model 1

```
################################################################################
Import Libraries Section
################################################################################
import matplotlib.pyplot as plt
import keras
import datetime
from keras import layers
from keras import models
from keras import optimizers
from keras.utils import to_categorical
import numpy as np
from keras.datasets import cifar10

################################################################################
Load Data Section
################################################################################
#Divide into training dataset and testing dataset
(x_train,y_train),(x_test,y_test) = cifar10.load_data()

#set the starting time in order to obtain the running time later
start_time = datetime.datetime.now()
################################################################################
Pretreat Data Section
################################################################################
#scale the data (255 is maximum number)
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

#Convert the response to categorical numbers
y_test = to_categorical(y_test.astype(np.int))
y_train = to_categorical(y_train.astype(np.int))

################################################################################
Define Model Section
################################################################################
#Build the model
network=models.Sequential()
#the input shape is (32,32,3) and I used a 3*3 convolutional patch
network.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
#max pooling is to extract from input feature maps and output max value of each channel, this is doen with 2*2 windows
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
```

```
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
#transform the format to be read by a dense layer
network.add(layers.Flatten())
network.add(layers.Dense(128, activation='relu'))
network.add(layers.Dense(64, activation='relu'))
#this is the output layer in a format of dummy variables
network.add(layers.Dense(10, activation='softmax'))
network.summary()

#compile the model, since this is a multi-categorical classification,
#using accuracy as metrics and categorical_crossentropy as loss function
network.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

#fit the model with an epoch number of 50 and a batch size of 64
history = network.fit(x_train,
                    y_train,
                    epochs=50,
                    batch_size=64,
                    validation_data=(x_test, y_test))

#count the time
stop_time=datetime.datetime.now()
print("Time required for training:",stop_time-start_time)

#plot the training loss and testing loss against number of epochs
plt.figure()
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Test loss')
plt.title('Training and Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#plot the training accuracy and testing accuracy against number of epochs
plt.figure()
acc = history.history['acc']
val_acc= history.history['val_acc']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training Acc')
plt.plot(epochs, val_acc, 'b', label='Test Acc')
plt.title('Training and Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()
plt.show()
```

Appendix B: Codes of Model 2

```
#########################################################################
Import Libraries Section
#########################################################################
import matplotlib.pyplot as plt
import keras
import datetime
from keras import layers
from keras import models
from keras import optimizers
from keras.utils import to_categorical
import numpy as np
```

```
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator

###############################################################################
Load Data Section
###############################################################################
#Divide into training dataset and testing dataset
(x_train,y_train),(x_test,y_test) = cifar10.load_data()

#set the starting time in order to obtain the running time later
start_time = datetime.datetime.now()
###############################################################################
Pretreat Data Section
###############################################################################
#scale the data (255 is maximum number)
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

#Convert the response to categorical numbers
y_test = to_categorical(y_test.astype(np.int))
y_train = to_categorical(y_train.astype(np.int))

###############################################################################
Define Model Section
###############################################################################
#Build the model
network=models.Sequential()
#the input shape is (32,32,3) and I used a 3*3 convolutional patch
network.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
#max pooling is to extract from input feature maps and output max value of each channel, this is done with 2*2 windows
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
#transform the format to be read by a dense layer
network.add(layers.Flatten())
network.add(layers.Dense(128, activation='relu'))
network.add(layers.Dense(64, activation='relu'))
#this is the output layer in a format of dummy variables
network.add(layers.Dense(10, activation='softmax'))
network.summary()

#compile the model, since this is a multi-categorical classification,
#using accuracy as metrics and categorical_crossentropy as loss function
network.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

###############################################################################
Preprocessing Section
###############################################################################
#Set a way to use data augmentation generators
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

#fit the model with an epoch number of 50 and a batch size of 64
#Using the training data treated by the data augmentation generators
history=network.fit_generator(  datagen.flow(x_train, y_train),
                        steps_per_epoch=x_train.shape[0]/64,
                        validation_data=(x_test, y_test),
                        validation_steps=x_test.shape[0]/64,
                        epochs=50,
                        verbose=1)

#count the running time
```

```
stop_time=datetime.datetime.now()
print("Time required for training:",stop_time-start_time)

#plot the training loss and testing loss against number of epochs
plt.figure()
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Test loss')
plt.title('Training and Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#plot the training accuracy and testing accuracy against number of epochs
plt.figure()
acc = history.history['acc']
val_acc= history.history['val_acc']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training Acc')
plt.plot(epochs, val_acc, 'b', label='Test Acc')
plt.title('Training and Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()
plt.show()
```

Appendix C: Codes of Model 3

```
################################################################################
Import Libraries Section
################################################################################
import matplotlib.pyplot as plt
import keras
import datetime
from keras import layers
from keras import models
from keras import optimizers
from keras.utils import to_categorical
import numpy as np
from keras.datasets import cifar10

################################################################################
Load Data Section
################################################################################
#Divide into training dataset and testing dataset
(x_train,y_train),(x_test,y_test) = cifar10.load_data()

#set the starting time in order to obtain the running time later
start_time = datetime.datetime.now()
################################################################################
Pretreat Data Section
################################################################################
#scale the data (255 is maximum number)
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

#Convert the response to categorical numbers
y_test = to_categorical(y_test.astype(np.int))
y_train = to_categorical(y_train.astype(np.int))

################################################################################
Define Model Section
################################################################################
network=models.Sequential()
#just use only one convolutional layer instead
```

```
network.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
network.add(layers.Flatten())
network.add(layers.Dense(10, activation='softmax'))
network.summary()

#compile the model, since this is a multi-categorical classification,
#using accuracy as metrics and categorical_crossentropy as loss function
network.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

#fit the model with an epoch number of 50 and a batch size of 64
history = network.fit(x_train,
                      y_train,
                      epochs=50,
                      batch_size=64,
                      validation_data=(x_test, y_test))

#count the time
stop_time=datetime.datetime.now()
print("Time required for training:",stop_time-start_time)

#plot the training loss and testing loss against number of epochs
plt.figure()
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Test loss')
plt.title('Training and Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#plot the training accuracy and testing accuracy against number of epochs
plt.figure()
acc = history.history['acc']
val_acc= history.history['val_acc']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training Acc')
plt.plot(epochs, val_acc, 'b', label='Test Acc')
plt.title('Training and Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()
plt.show()
```

Appendix D: Codes of Model 4

```
#########################################################################
Import Libraries Section
#########################################################################
import matplotlib.pyplot as plt
import keras
import datetime
from keras import layers
from keras import models
from keras import optimizers
from keras.utils import to_categorical
import numpy as np
from keras.datasets import cifar10

#########################################################################
Load Data Section
#########################################################################
```

```
#Divide into training dataset and testing dataset
(x_train,y_train),(x_test,y_test) = cifar10.load_data()

#set the starting time in order to obtain the running time later
start_time = datetime.datetime.now()
################################################################################
Pretreat Data Section
################################################################################
#scale the data (255 is maximum number)
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

#Convert the response to categorical numbers
y_test = to_categorical(y_test.astype(np.int))
y_train = to_categorical(y_train.astype(np.int))
################################################################################
Define Model Section
################################################################################
network=models.Sequential()
network.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
#Just add another layer here
network.add(layers.Conv2D(128, (3, 3), activation='relu'))
network.add(layers.Flatten())
network.add(layers.Dense(128, activation='relu'))
network.add(layers.Dense(64, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))
network.summary()

#compile the model, since this is a multi-categorical classification,
#using accuracy as metrics and categorical_crossentropy as loss function
network.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

#fit the model with an epoch number of 50 and a batch size of 64
history = network.fit(x_train,
                    y_train,
                    epochs=50,
                    batch_size=64,
                    validation_data=(x_test, y_test))

#count the time
stop_time=datetime.datetime.now()
print("Time required for training:",stop_time-start_time)

#plot the training loss and testing loss against number of epochs
plt.figure()
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Test loss')
plt.title('Training and Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#plot the training accuracy and testing accuracy against number of epochs
plt.figure()
acc = history.history['acc']
val_acc= history.history['val_acc']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training Acc')
plt.plot(epochs, val_acc, 'b', label='Test Acc')
plt.title('Training and Test Accuracy')
plt.xlabel('Epochs')
```

```
plt.ylabel('Acc')
plt.legend()
plt.show()
```

Appendix E: Codes of Model 5

```
##############################################################################
Import Libraries Section
##############################################################################
import matplotlib.pyplot as plt
import keras
import datetime
from keras import layers
from keras import models
from keras import optimizers
from keras.utils import to_categorical
import numpy as np
from keras.datasets import cifar10

##############################################################################
Load Data Section
##############################################################################
#Divide into training dataset and testing dataset
(x_train,y_train),(x_test,y_test) = cifar10.load_data()

#set the starting time in order to obtain the running time later
start_time = datetime.datetime.now()
##############################################################################
Pretreat Data Section
##############################################################################
#scale the data (255 is maximum number)
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

#Convert the response to categorical numbers
y_test = to_categorical(y_test.astype(np.int))
y_train = to_categorical(y_train.astype(np.int))
##############################################################################
Define Model Section
##############################################################################
network=models.Sequential()
network.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Flatten())
#just add the dropout layer here
network.add(layers.Dropout(0.5))
network.add(layers.Dense(128, activation='relu'))
network.add(layers.Dense(64, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))
network.summary()


#compile the model, since this is a multi-categorical classification,
#using accuracy as metrics and categorical_crossentropy as loss function
network.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

#fit the model with an epoch number of 50 and a batch size of 64
history = network.fit(x_train,
                    y_train,
                    epochs=50,
                    batch_size=64,
                    validation_data=(x_test, y_test))
```

```
#count the time
stop_time=datetime.datetime.now()
print("Time required for training:",stop_time-start_time)

#plot the training loss and testing loss against number of epochs
plt.figure()
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Test loss')
plt.title('Training and Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#plot the training accuracy and testing accuracy against number of epochs
plt.figure()
acc = history.history['acc']
val_acc= history.history['val_acc']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training Acc')
plt.plot(epochs, val_acc, 'b', label='Test Acc')
plt.title('Training and Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()
plt.show()
```