Convolutional Neural Networks (CNN)

Xiao (Megan) Ling

A Convolutional Neural Network is an artificial neural network that can detect patterns. Pattern detection makes CNN's popularly used for analyzing images for computer vision tasks. Like other neural networks, CNN is made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum and pass it through an activation function, then respond with an output.

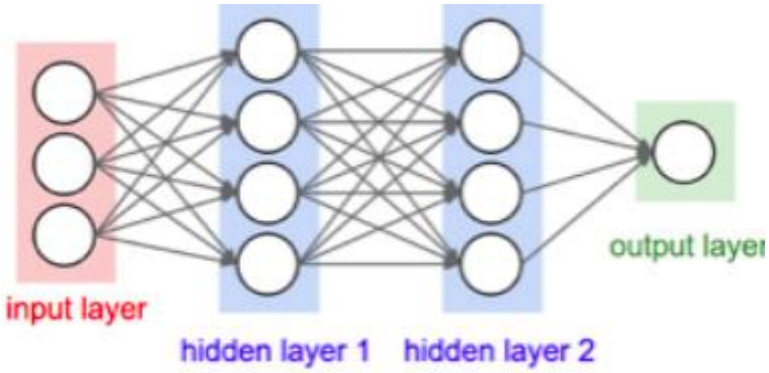

Figure 1.1

Different from other Neural Networks, CNN's' inputs are not vectors. The input would be a 3-dimension describing a multichannel image. After applying with models, our 3-dimensional layers are going to be converted all the resultant 2-dimensional arrays into a single long continuous linear vector.

The other significant architecture difference than regular Neural Networks is that each layer is not fully connected to the next layer. CNN's have hidden layers called convolutional layers "Conv2D" etc. These layers have a special feature filter that creates receptive filed and perform output on its new layer.

According to Figure 1.2, from the Input stage to pool 2 stage includes 2 hidden layers, each layer filters out a small portion of the former layer. After Pool 2, the transformed data finally a full connected to produce an output. In other words, the neurons in one layer connect to a small region of the next layer, and the layers output is going to connect fully until the model finished running each layer.

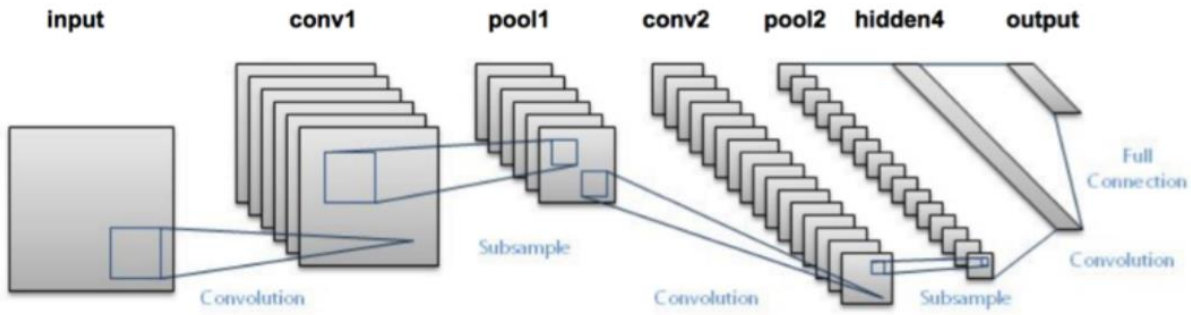

Figure 1.2

In the following 5 Models, we will explore CNN from simple to complex and learn different results generated from different model architectures. To generate a comparable result, we separated data into same Train and Test set for each model. The Train set has 50000

samples. Test set has 10000 samples. Next, we set batch size to 128; a binary batch size would improve our running time. I also applied epoch 20 and 50 to test if we can improve model accuracy.

The 1st Model – 4 convolutional layers and 2 fully connected layers

In the 1st CNN model, we build 4 convolutional layers and 2 fully connected layers. Figure 2.1 shows a basic convent with a stack of Conv2D and MaxPooling2D layers. CNN usually consists of 4 major steps that are Convolution, Pooling, Flatting, and Full connection.

```
model = Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(NB_CLASSES,activation='softmax'))
# we are doing 10-way classification, using a final layer with 10 outputs and
#a softmax activation
model.summary()
```

Figure 2.1

The Conv2D takes input_shape of (32,32,3), which is the format of each picture. The (3,3) of the Conv2D (64, (3,3)) is the filter of each convolutional layer that detects patterns. The deeper the network goes, the more sophisticated the filters become. In the later layers, our filters would be able to detect the specific object that describes the details of imagines, such as eyes, feathers, etc.

The activation 'relu' is the wildly used activation function in the world right now. Since it is **used** in almost all the convolutional neural networks or deep learning.

"Maxpooling" is a sample-based discretization process. The objective is to reduce Conv2D's dimensionality. It also reduces the computational cost by reducing the number of parameters, so, "Maxpooling" prevents overfitting to some degree.

Finally, the fully connected layers "Layers. Dense" perform classification on the features extracted by the convolutional layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.

The final activation parameter is "SoftMax," which takes an un-normalized vector, and normalizes it into a probability distribution. It scales the outputs of each unit to between 0 and 1. Generally, the final layer of density "SoftMax" tells the probability that any of the classes are true.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 12, 12, 64)        36928
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 10, 10, 64)        36928
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                102464
_________________________________________________________________
dense_12 (Dense)             (None, 10)                650
=================================================================
Total params: 196,362
Trainable params: 196,362
Non-trainable params: 0
```

Figure 2.2

From the Model.Summary() result (Figure2.2), we learned that the output of every Convf2D and MaxPooling2D layer is a 3D tensor of shape (height, width, channels). The width and height dimensions tend to shrink as we put more layers into the model and dig deeper. Next, we feed the last output tensor into a densely connected classifier network; We flatten the 3D outputs to 1D, and then add 2 dense layers afterward; The (10,10,64) output from the 2nd Conv2D layer are flattened into vectors of shape (, 1600).

Then we apply the model into Train and Test data set. It took me approximate 5 mins to run each epoch on my computer with Batch = 128, **Epoch = 20**. The result has a Test score of 1.2568523509979248 and **Test accuracy of 0.728**

To improve Test accuracy, I increase the **Epoch to 50**. Surprisingly, the result has not improved significantly with a Test score of 2.4732305290222167 and **Test accuracy of 0.71866679.** One possible explanation is that each time model generates different neuron linkage; and the weighted neuron might contain less useful information. So, the model "misleading" to a less accurate result. I need to improve the model by introducing layers such as Dropout.

Figure 2.3 shows the test accuracy along with 20 epochs. With the model accuracy increasing, the train model loss decreased, however, the test model loss increased. So it is a sign that our model has an overfitting problem.

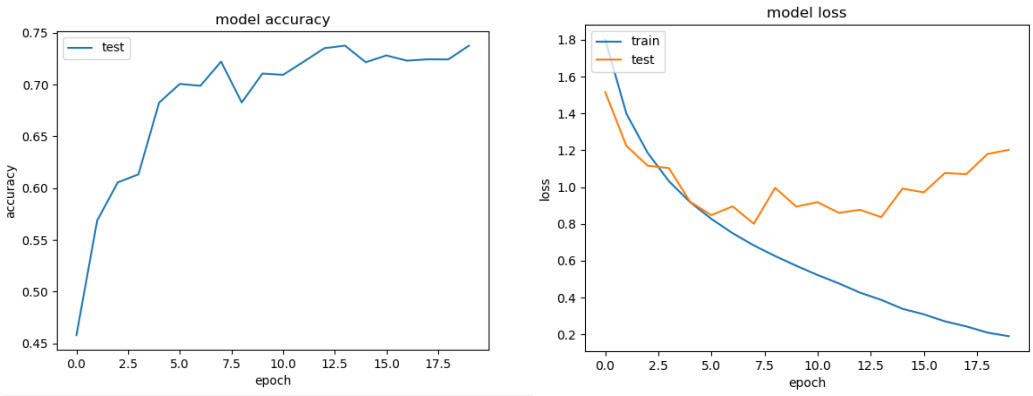

Figure 2.3

The 2nd Model a CNN model with data augmentation

In Model 1, we designed a CNN model with 4 convolutional layers and 2 dense layers. The Test accuracy reached 68% with epoch = 50 and 67% with epoch = 20. In the 2nd Model. To build a powerful image classifier, we add data argument into the 1st Model to see if the data argument would boost the deep learning performance.

Image augmentation is wildly used in image classification. It artificially creates training images through different ways of processing the existing in training dataset to generate the altered versions of the same images.

```
#data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    )
datagen.fit(X_train)
```

Figure 3.1

Figure 3.1 is the image augmentation code. There is a wild range of image augmentation methods, such as change the existing image into a broader variety of lighting and coloring. Here, we create diverse types of existing images in the Train set through rotating, changing width and heights, and horizontal flipping it.

Since the image argument only ample the train set images, the model structure will not change. I expect a slightly higher test accuracy and a less overfitting result from the image argument since the model was trained by “diversified” training images.

The result is favorable; model 2 has a Test accuracy of **Test accuracy: 0.7968.**

According to the model loss graph in Figure 3.2, the test loss is lower than the train loss, which means there is less overfitting problem as the model become more accurate.

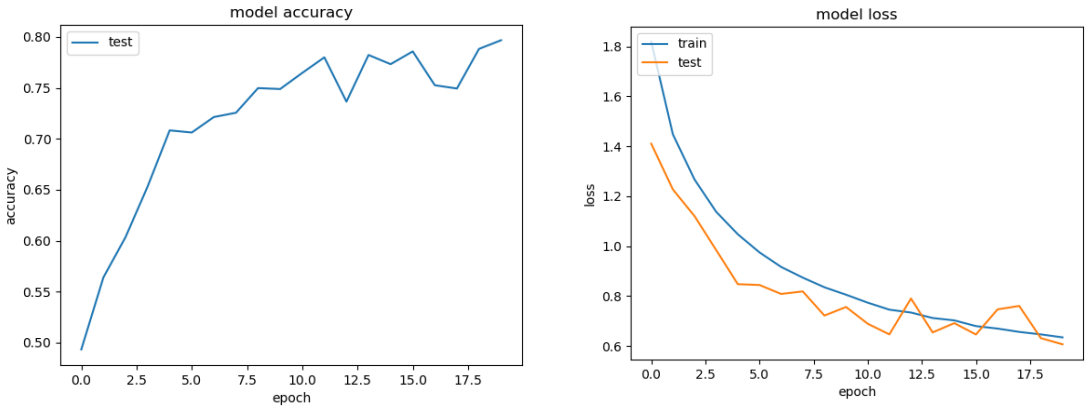

Figure 3.2

The 3rd Model a network that only has one hidden convolutional layer

The 3rd Model has a network that only has one hidden convolutional layer. According to the CNN pattern recognition theory, with fewer convolutional layer, the test image would be roughly recognized by our model. As a result, the test accuracy will much lower than the 1st Model. With our assumption, we build a model as Figure 4.1 shows.

```
model = Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
#model.add(Dropout(0.25))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
#model.add(Dropout(0.5))
model.add(layers.Dense(NB_CLASSES,activation='softmax'))
```

Figure 4.1

Layer (type)	Output Shape	Param #
conv2d_31 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_9 (MaxPooling2	(None, 15, 15, 32)	0
flatten_7 (Flatten)	(None, 7200)	0
dense_12 (Dense)	(None, 64)	460864
dense_13 (Dense)	(None, 10)	650

Total params: 462,410
Trainable params: 462,410
Non-trainable params: 0

Figure 4.2

According to Figure 4.2, the last convolutional layer output a (15, 15, 32) area, which is much broader than our 1st model with 3 convolutional layers. Also, since the filter Conv2D (3,3) applied in only 1 hidden layer, we can expect a less accuracy test result.

After implementing the model with 20 epochs, the test accuracy results in **0.6063**, lower than the test accuracy result from model 1, which is **0.728**. As Figure 4.3 shows below, the test model loss is increasing along with increasing in epochs, higher than the train model loss. It means the CNN model with only 1 convolutional layer is easy to result in overfitting.

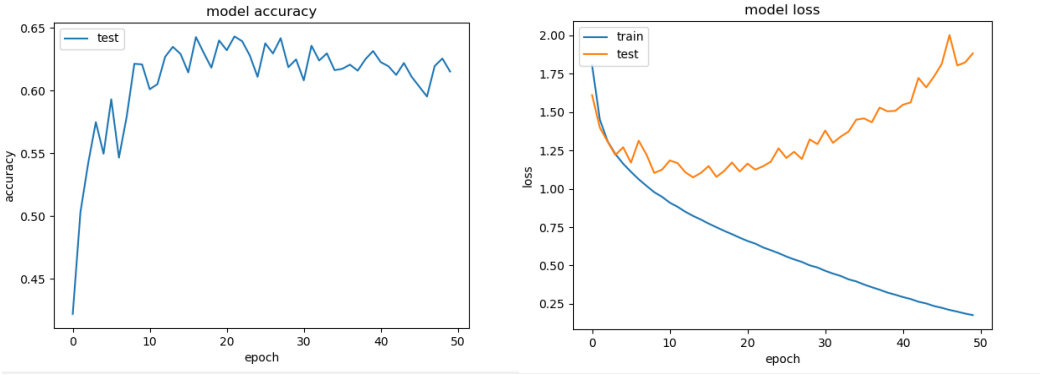

Figure 4.3

Overall, with the less convolutional layer, the test result accuracy would decrease. Since we filter the 3*3 block of pixels for once and compute the dot product for once. There is no additional “future map” to filter so that the preliminary image after the 1 hidden convolutional layer is blurry. It is hard for a machine to determine image details with “less detailed clues.”

The 4th Model – 5 hidden layers introduced model parameters

The 4th Model has 5 hidden layers. As we discussed before, more convolutional layers result in a more complex convolutional process; The better the machine learn, the more accuracy the test result would be. With this assumption, let us run the model.

```
model.add(Conv2D(32, kernel_size=3, padding='same',
                 input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(Activation('relu'))
model.add(Conv2D(32, kernel_size=3, padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=3, padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, kernel_size=3, padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=3, padding='same'))
model.add(Activation('relu'))Z
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
```

Figure 5.1

In Figure 5.1, there are 5 hidden layers activated by 'relu,' which is popular in image recognition. I also introduce the parameter "padding" into the model. "Padding = ' same'" means the input image ought to have zero padding so that the output in convolution does not differ in size as input. In simple, we add an n-pixel border of zero to tell that 'do not' reduce the dimension and have the same as the input. The important thing is that when we add boundaries of zero pixels to input, then we reduce contract in feature maps.

The model.summary() result showed in Figure 5.2 illustrated the convolutional hierarchy neural network model. There are 4 hidden layers commanded by Conv2D, reducing the 3-dimensional layers from (32, 32, 32) to (8, 8, 64); After each convolution step, a dropout applied to the model, we will discuss dropout in detail in the next model 5. The fattened convolutional output is (, 4096).

To prevent overfitting in this 5 hidden layers model, I added 2 dropout layers. We will further discuss the function of dropout layers in Model 5.

Layer (type)	Output Shape	Param #
conv2d_90 (Conv2D)	(None, 32, 32, 32)	896
activation_7 (Activation)	(None, 32, 32, 32)	0
conv2d_91 (Conv2D)	(None, 32, 32, 32)	9248
activation_8 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_41 (MaxPooling	(None, 16, 16, 32)	0
dropout_4 (Dropout)	(None, 16, 16, 32)	0
conv2d_92 (Conv2D)	(None, 16, 16, 64)	18496
activation_9 (Activation)	(None, 16, 16, 64)	0
conv2d_93 (Conv2D)	(None, 16, 16, 64)	36928
activation_10 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_42 (MaxPooling	(None, 8, 8, 64)	0
conv2d_94 (Conv2D)	(None, 8, 8, 64)	36928
activation_11 (Activation)	(None, 8, 8, 64)	0
dropout_5 (Dropout)	(None, 8, 8, 64)	0
flatten_23 (Flatten)	(None, 4096)	0
dense_45 (Dense)	(None, 512)	2097664
activation_12 (Activation)	(None, 512)	0
dropout_6 (Dropout)	(None, 512)	0
dense_46 (Dense)	(None, 10)	5130
activation_13 (Activation)	(None, 10)	0

Figure 5.2

Test Accuracy is significantly improved to **0.7723** compared with test accuracy of 72% from the 1st model. Because the more layers applied, the deeper neuron network has, and the more detailed information were captured.

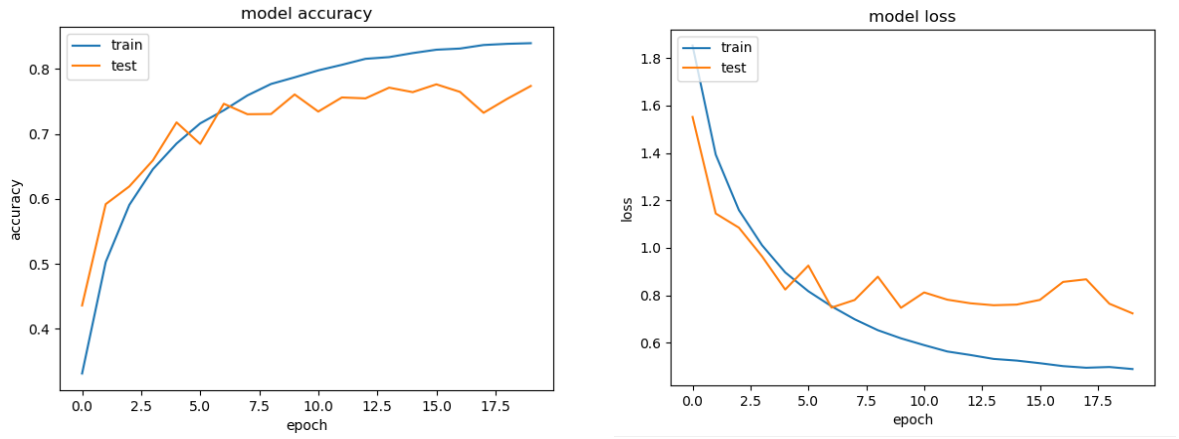

Figure 5.3

At the same time, even the test model loss is higher than the train model loss; the test model accuracy experienced fluctuations and lower than the train model accuracy. So, Model 4 should not have an overfitting problem.

The 5th Model – Add Dropout layers for Model 1

In Model 5, a regulatory technique “Dropout” is introduced in the model since 5 convolutional layers have a high probability of generating an overfit result. To prevent neural networks from overfitting, “Dropout” is a technique where randomly selected neurons are ignored during training.

```
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(Dropout(0.5))
model.add(layers.Dense(NB_CLASSES,activation='softmax'))
model.summary()
```

Figure 6.1

In Figure 6.1, the Dropout layers applied after convolution layers and the fully connected layers. Because, dropout works per-neuron, dropping a neuron means that the corresponding feature map is dropped. So, after each “aggregated” future map is generated, Dropout some linkage between neurons so that a clean future map is created to produce an un-overfitting result.

Layer (type)	Output Shape	Param #
conv2d_99 (Conv2D)	(None, 30, 30, 32)	896
conv2d_100 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_45 (MaxPooling	(None, 14, 14, 64)	0
dropout_10 (Dropout)	(None, 14, 14, 64)	0
conv2d_101 (Conv2D)	(None, 12, 12, 64)	36928
conv2d_102 (Conv2D)	(None, 10, 10, 64)	36928
max_pooling2d_46 (MaxPooling	(None, 5, 5, 64)	0
dropout_11 (Dropout)	(None, 5, 5, 64)	0
flatten_25 (Flatten)	(None, 1600)	0
dense_49 (Dense)	(None, 64)	102464
dropout_12 (Dropout)	(None, 64)	0
dense_50 (Dense)	(None, 10)	650

Figure 6.2

From model. Summary (), we can tell the dropout shape identical to the former layer shape because of its dropout neural in the map that its former layer created. What Dropout do in the 2 processes is to random drop the paths to the weighted neurons. So that the network becomes less sensitive to the specific weights of neurons.

With the Dropout layers, the overfitting problem in the 1st Model is solved, and the test accuracy would improve. The model loss picture should have a test model loss that is lower than the train model loss.

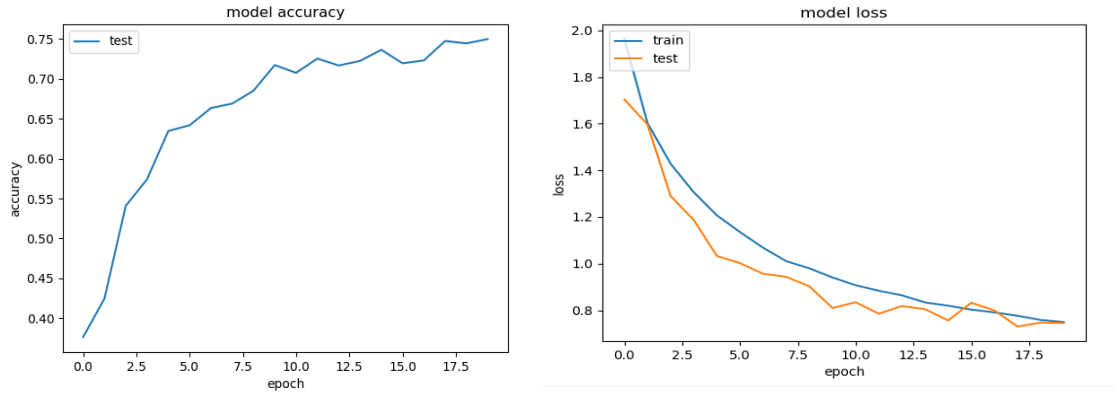

Figure 6.3

References:

Cornelisse, D., & Cornelisse, D. (2018, April 24). An intuitive guide to Convolutional Neural Networks. Retrieved from https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. (2018, December 12). Retrieved from https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Dropout Regularization in Deep Learning Models With Keras. (2017, March 30). Retrieved from https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

Abhijeet3922. (n.d.). Abhijeet3922/Object-recognition-CIFAR-10. Retrieved from https://github.com/abhijeet3922/Object-recognition-CIFAR-10/blob/master/cifar10_90.py

Lau, S., & Lau, S. (2017, July 10). Image Augmentation for Deep Learning. Retrieved from https://towardsdatascience.com/image-augmentation-for-deep-learning-histogram-equalization-a71387f609b2

Python Deep Learning: Part 4 – Jon C-137 – Medium. (n.d.). Retrieved from https://medium.com/@jon.froiland/python-deep-learning-part-4-ea745ed9cd77

Appendix:

Model 1

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
at least 3 convolutional layers and at least 1 fully connected layer
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

from keras.datasets import cifar10
from keras.utils import np_utils
from keras.models import Sequential
from keras import layers
from keras import models
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.optimizers import SGD, Adam, RMSprop
from keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Parameters Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

#from quiver_engine import server
# CIFAR_10 is a set of 60K images 32x32 pixels on 3 channels
IMG_CHANNELS = 3
IMG_ROWS = 32
IMG_COLS = 32
```

```
#constant
BATCH_SIZE = 128
NB_EPOCH = 50
NB_CLASSES = 10
VERBOSE = 1
VALIDATION_SPLIT = 0.2
OPTIM = RMSprop()


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Load Data Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
#load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Pretreat Data Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
# convert to categorical
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# float and normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Define Model Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
# network

model = Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                    input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(NB_CLASSES,activation='softmax'))
# we are doing 10-way classification, using a final layer with 10 outputs and
#a softmax activation
model.summary()

model.compile(loss='categorical_crossentropy', optimizer=OPTIM,
        metrics=['accuracy'])

history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE,
        epochs=NB_EPOCH, validation_split=VALIDATION_SPLIT,
        verbose=VERBOSE)
```

```
"""
Show output Section
"""
print('Testing...')
score = model.evaluate(X_test, Y_test,
                       batch_size=BATCH_SIZE, verbose=VERBOSE)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])

#server.launch(model)


#save model
model_json = model.to_json()
open('cifar10_architecture.json', 'w').write(model_json)
model.save_weights('cifar10_weights.h5', overwrite=True)


# list all data in history
print(history.history.keys())
# summarize history for accuracy
#plt.plot(mo)
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Model 2

```
"""
data augmentation
"""
"""
Import Libraries Section
"""
from keras.datasets import cifar10
from keras.utils import np_utils
from keras.models import Sequential
from keras import layers
from keras import models
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.optimizers import SGD, Adam, RMSprop
from keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt
```

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Parameters Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
#from quiver_engine import server
# CIFAR_10 is a set of 60K images 32x32 pixels on 3 channels
IMG_CHANNELS = 3
IMG_ROWS = 32
IMG_COLS = 32

#constant
BATCH_SIZE = 128
NB_EPOCH = 20
NB_CLASSES = 10
VERBOSE = 1
VALIDATION_SPLIT = 0.2
OPTIM = RMSprop()


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Load Data Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
#load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Pretreat Data Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
# convert to categorical
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# float and normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Define Model Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
# network

model = Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                    input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(NB_CLASSES,activation='softmax'))
```

```
# we are doing 10-way classification, using a final layer with 10 outputs and
#a softmax activation
model.summary()

#data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    )
datagen.fit(X_train)

model.compile(loss='categorical_crossentropy', optimizer=OPTIM,
          metrics=['accuracy'])

model.fit_generator(datagen.flow(X_train, Y_train, batch_size=BATCH_SIZE),\
                 steps_per_epoch=X_train.shape[0] // BATCH_SIZE,epochs=20,\
                 verbose=1,validation_data=(X_test,Y_test))

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Show output Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
print('Testing...')
score = model.evaluate(X_test, Y_test,
                  batch_size=BATCH_SIZE, verbose=VERBOSE)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])

#server.launch(model)


#save model
model_json = model.to_json()
open('cifar10_architecture.json', 'w').write(model_json)
model.save_weights('cifar10_weights.h5', overwrite=True)


# list all data in history
print(history.history.keys())
# summarize history for accuracy
#plt.plot(mo)
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
plt.show()
```

Model 3

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 20 21:01:41 2019

@author: megan
"""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
a network that only has one hidden convolutional layer
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
from keras.datasets import cifar10
from keras.utils import np_utils
from keras.models import Sequential
from keras import layers
from keras import models
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.optimizers import SGD, Adam, RMSprop

import matplotlib.pyplot as plt

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Parameters Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#from quiver_engine import server
# CIFAR_10 is a set of 60K images 32x32 pixels on 3 channels
IMG_CHANNELS = 3
IMG_ROWS = 32
IMG_COLS = 32

#constant
BATCH_SIZE = 128
NB_EPOCH = 50
NB_CLASSES = 10
VERBOSE = 1
VALIDATION_SPLIT = 0.2
OPTIM = RMSprop()


""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# convert to categorical
```

```
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# float and normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Define Model Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
# network

model = Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                    input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
#model.add(Dropout(0.25))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
#model.add(Dropout(0.5))
model.add(layers.Dense(NB_CLASSES,activation='softmax'))
# we are doing 10-way classification, using a final layer with 10 outputs and
#a softmax activation
model.summary()
#(4,4,64) output from the 2nd Conv2D layer are flattened into vectors of shape(1024,)

# train
#optim = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=OPTIM,
          metrics=['accuracy'])

history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE,
          epochs=NB_EPOCH, validation_split=VALIDATION_SPLIT,
          verbose=VERBOSE)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Show output Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
print('Testing...')
score = model.evaluate(X_test, Y_test,
                   batch_size=BATCH_SIZE, verbose=VERBOSE)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])

#server.launch(model)


#save model
model_json = model.to_json()
open('cifar10_architecture.json', 'w').write(model_json)
model.save_weights('cifar10_weights.h5', overwrite=True)


# list all data in history
print(history.history.keys())
```

```
# summarize history for accuracy
#plt.plot(mo)
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Model 4

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Import Libraries Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

from keras.datasets import cifar10
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.optimizers import SGD, Adam, RMSprop

from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# CIFAR_10 is a set of 60K images 32x32 pixels on 3 channels
IMG_CHANNELS = 3
IMG_ROWS = 32
IMG_COLS = 32

#constant
BATCH_SIZE = 128
NB_EPOCH = 20
NB_CLASSES = 10
VERBOSE = 1
VALIDATION_SPLIT = 0.2
OPTIM = RMSprop()


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Load Data Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

#load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
"""
Pretreat Data Section
"""
# convert to categorical
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# float and normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
"""
Define Model Section
"""
# network

model = Sequential()

model.add(Conv2D(32, kernel_size=3, padding='same',
                 input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(Activation('relu'))
model.add(Conv2D(32, kernel_size=3, padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=3, padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, kernel_size=3, padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=3, padding='same'))
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))

model.summary()

model.compile(loss='categorical_crossentropy', optimizer=OPTIM,
      metrics=['accuracy'])


datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
```

```
    rotation_range=0,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=True,  # randomly flip images
    vertical_flip=False)  # randomly flip images

datagen.fit(X_train)

# train

history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE,
        epochs=NB_EPOCH, validation_split=VALIDATION_SPLIT,
        verbose=VERBOSE)

#model.fit_generator(datagen.flow(X_train, Y_train,
#                        batch_size=BATCH_SIZE),
#                        samples_per_epoch=X_train.shape[0],
#                        nb_epoch=NB_EPOCH,
#                        verbose=VERBOSE)

#server.launch(model)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Show output Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
print('Testing...')
score = model.evaluate(X_test, Y_test,
                batch_size=BATCH_SIZE, verbose=VERBOSE)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])

#save model
model_json = model.to_json()
open('cifar10_architecture.json', 'w').write(model_json)
model.save_weights('cifar10_weights.h5', overwrite=True)

# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Model 5

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 21 10:10:59 2019

@author: megan
"""

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
at least 3 convolutional layers and at least 1 fully connected layer
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
from keras.datasets import cifar10
from keras.utils import np_utils
from keras.models import Sequential
from keras import layers
from keras import models
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.optimizers import SGD, Adam, RMSprop

import matplotlib.pyplot as plt

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Parameters Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#from quiver_engine import server
# CIFAR_10 is a set of 60K images 32x32 pixels on 3 channels
IMG_CHANNELS = 3
IMG_ROWS = 32
IMG_COLS = 32

#constant
BATCH_SIZE = 128
NB_EPOCH = 20
NB_CLASSES = 10
VERBOSE = 1
VALIDATION_SPLIT = 0.2
OPTIM = RMSprop()


""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# convert to categorical
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
```

```
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# float and normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Define Model Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
# network

model = Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu',
                  input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(Dropout(0.5))
model.add(layers.Dense(NB_CLASSES,activation='softmax'))
model.summary()
#model.add(Dropout(0.25))
# train
#optim = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=OPTIM,
     metrics=['accuracy'])

history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE,
     epochs=NB_EPOCH, validation_split=VALIDATION_SPLIT,
     verbose=VERBOSE)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Show output Section
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
print('Testing...')
score = model.evaluate(X_test, Y_test,
                 batch_size=BATCH_SIZE, verbose=VERBOSE)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])

#server.launch(model)
#save model
model_json = model.to_json()
open('cifar10_architecture.json', 'w').write(model_json)
model.save_weights('cifar10_weights.h5', overwrite=True)
# list all data in history
print(history.history.keys())
# summarize history for accuracy
#plt.plot(mo)
```

```
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```