CS5542 BIG DATA APPS AND ANALYTICS

In Class Programming –5

Convolutional Neural Network (CNN)

Kalyani Nikure
Kmn6ng@umkc.edu

Table of Contents

Description

Use the same data that we used in ICP4

```
from keras.datasets import cifar10
```

and use the model provided in ICP5 to perform image classification. You must change 4 hyper parameters in the source code. Report your findings in detail.

Note: please indicate in your reports which 4 hyperparameters you changed in the source code and why in your opinion these changes are logical.

Detailed Steps Explanation

1. Model Evaluation which is already provided in the source code

- Importing all the required dependencies

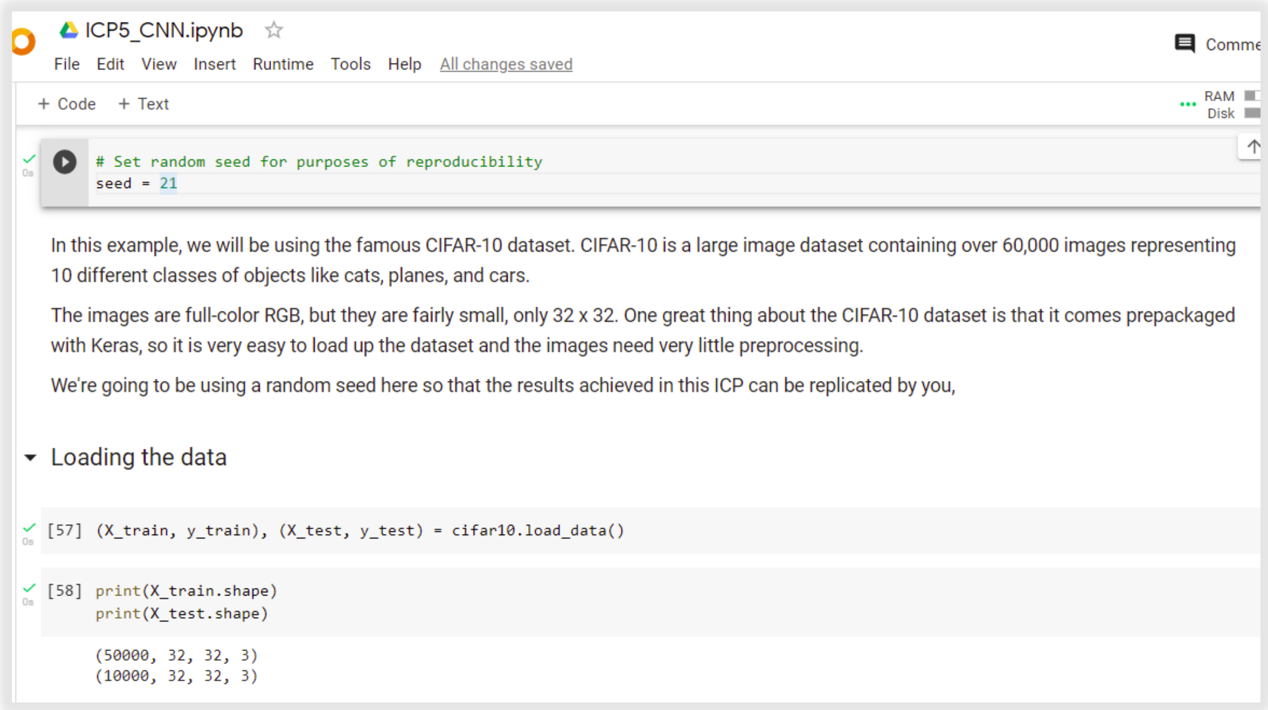
```
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras import datasets, layers, models
from keras import regularizers
from tensorflow.keras import layers
from keras.layers import Dense, Dropout, BatchNormalization
import matplotlib.pyplot as plt
import numpy as np
```

- About dataset

In this example, we will be using the famous CIFAR-10 dataset. CIFAR-10 is a large image dataset containing over 60,000 images representing 10 different classes of objects like cats, planes, and cars.

The images are full-color RGB, but they are small, only 32 x 32. One great thing about the CIFAR-10 dataset is that it comes prepackaged with Keras, so it is very easy to load up the dataset and the images need very little preprocessing.

- Reading the CIFAR-10 dataset and splitting data into training and testing set



```
# Set random seed for purposes of reproducibility
seed = 21
```

In this example, we will be using the famous CIFAR-10 dataset. CIFAR-10 is a large image dataset containing over 60,000 images representing 10 different classes of objects like cats, planes, and cars.

The images are full-color RGB, but they are fairly small, only 32 x 32. One great thing about the CIFAR-10 dataset is that it comes prepackaged with Keras, so it is very easy to load up the dataset and the images need very little preprocessing.

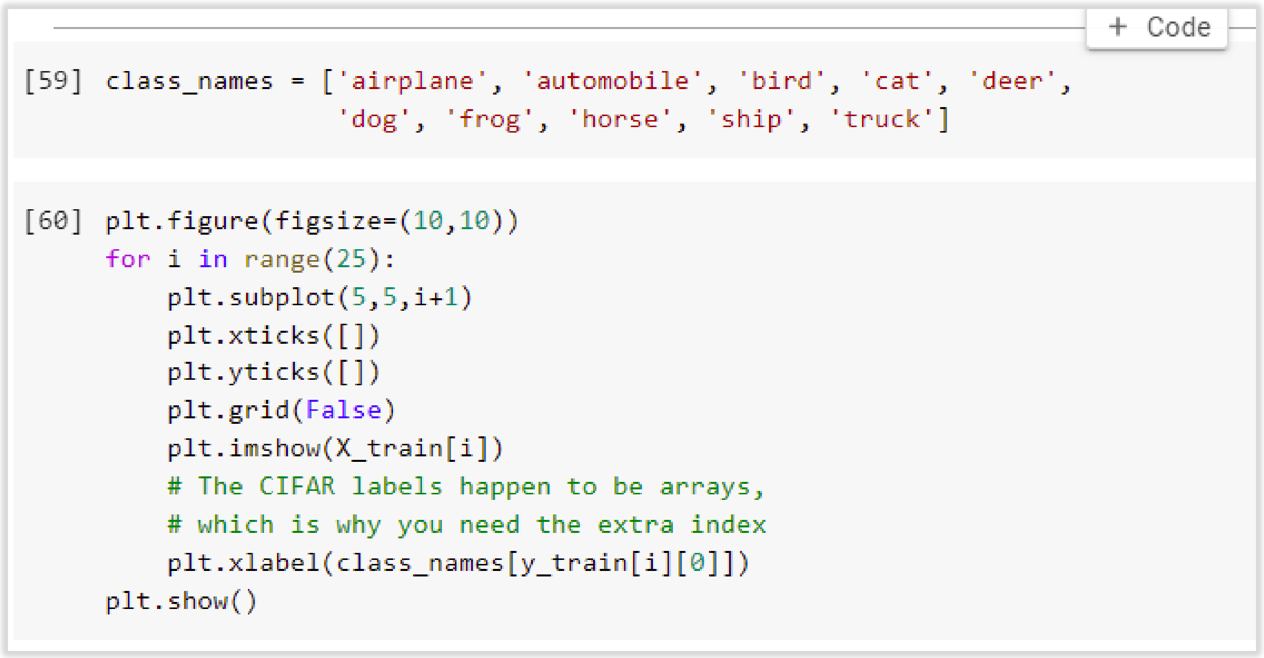
We're going to be using a random seed here so that the results achieved in this ICP can be replicated by you,

Loading the data

```
[57] (X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

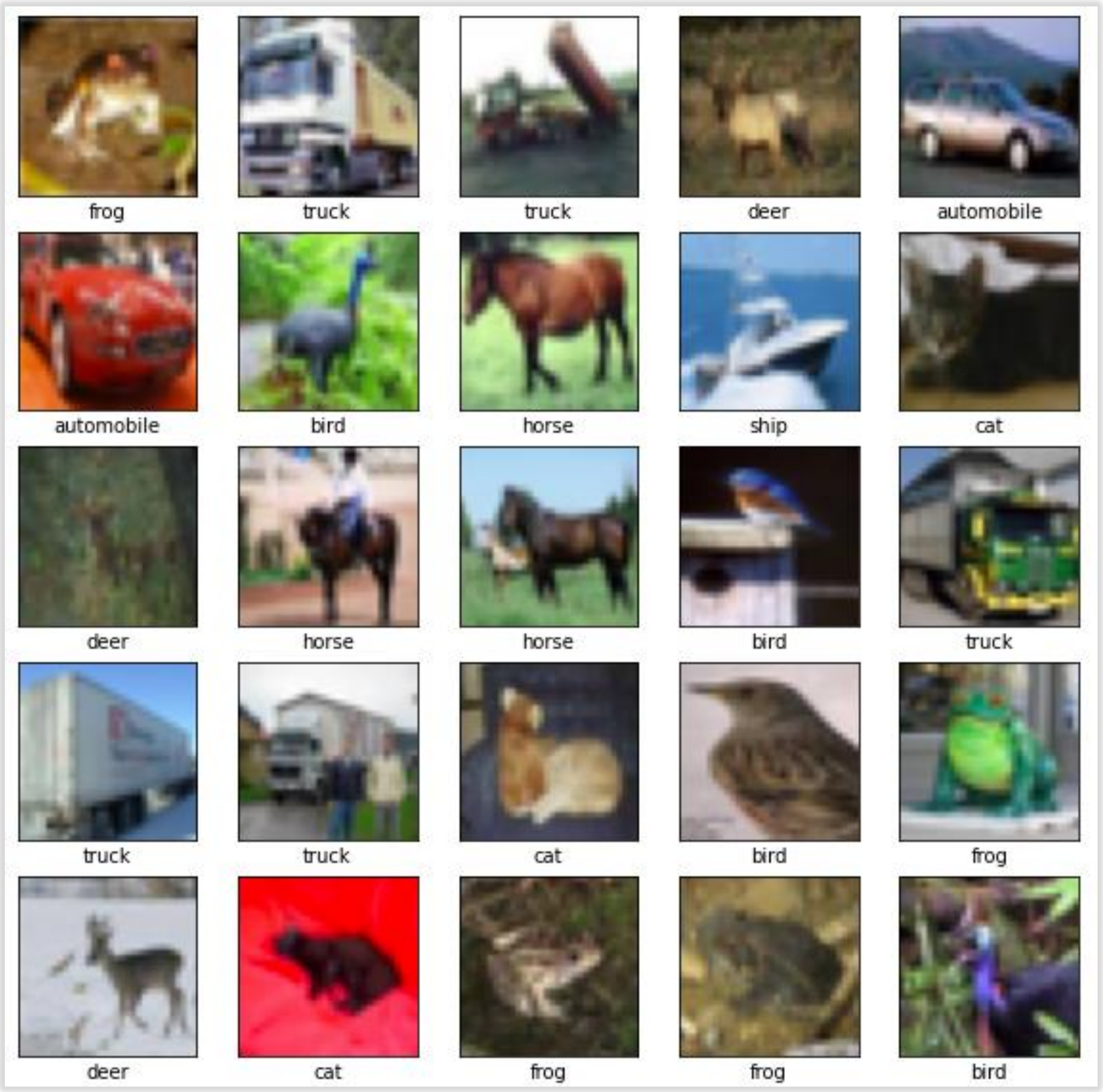
```
[58] print(X_train.shape)
     print(X_test.shape)

     (50000, 32, 32, 3)
     (10000, 32, 32, 3)
```

- Visualize the dataset

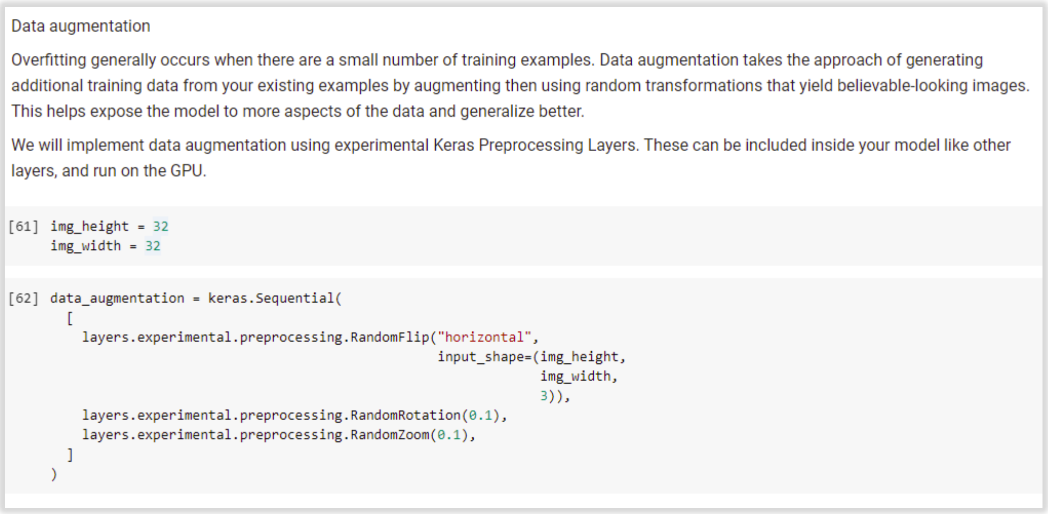


```
[59] class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                    'dog', 'frog', 'horse', 'ship', 'truck']
```

```
[60] plt.figure(figsize=(10,10))
     for i in range(25):
         plt.subplot(5,5,i+1)
         plt.xticks([])
         plt.yticks([])
         plt.grid(False)
         plt.imshow(X_train[i])
         # The CIFAR labels happen to be arrays,
         # which is why you need the extra index
         plt.xlabel(class_names[y_train[i][0]])
     plt.show()
```



- Data Augmentation



Data augmentation

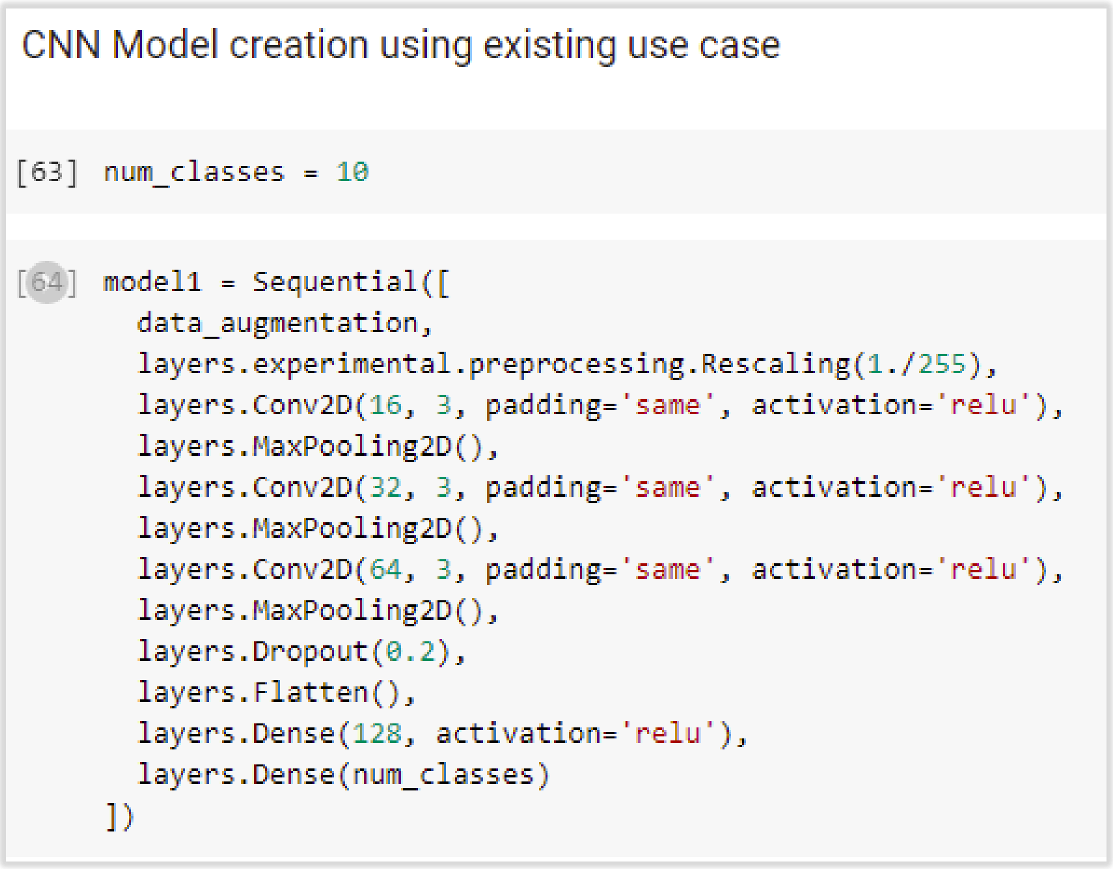
Overfitting generally occurs when there are a small number of training examples. Data augmentation takes the approach of generating additional training data from your existing examples by augmenting then using random transformations that yield believable-looking images. This helps expose the model to more aspects of the data and generalize better.

We will implement data augmentation using experimental Keras Preprocessing Layers. These can be included inside your model like other layers, and run on the GPU.

```
[61] img_height = 32
     img_width = 32
```

```
[62] data_augmentation = keras.Sequential(
       [
         layers.experimental.preprocessing.RandomFlip("horizontal",
                                                      input_shape=(img_height,
                                                                   img_width,
                                                                   3)),
         layers.experimental.preprocessing.RandomRotation(0.1),
         layers.experimental.preprocessing.RandomZoom(0.1),
       ]
     )
```

- CNN Model Creation using Keras by setting up the layers

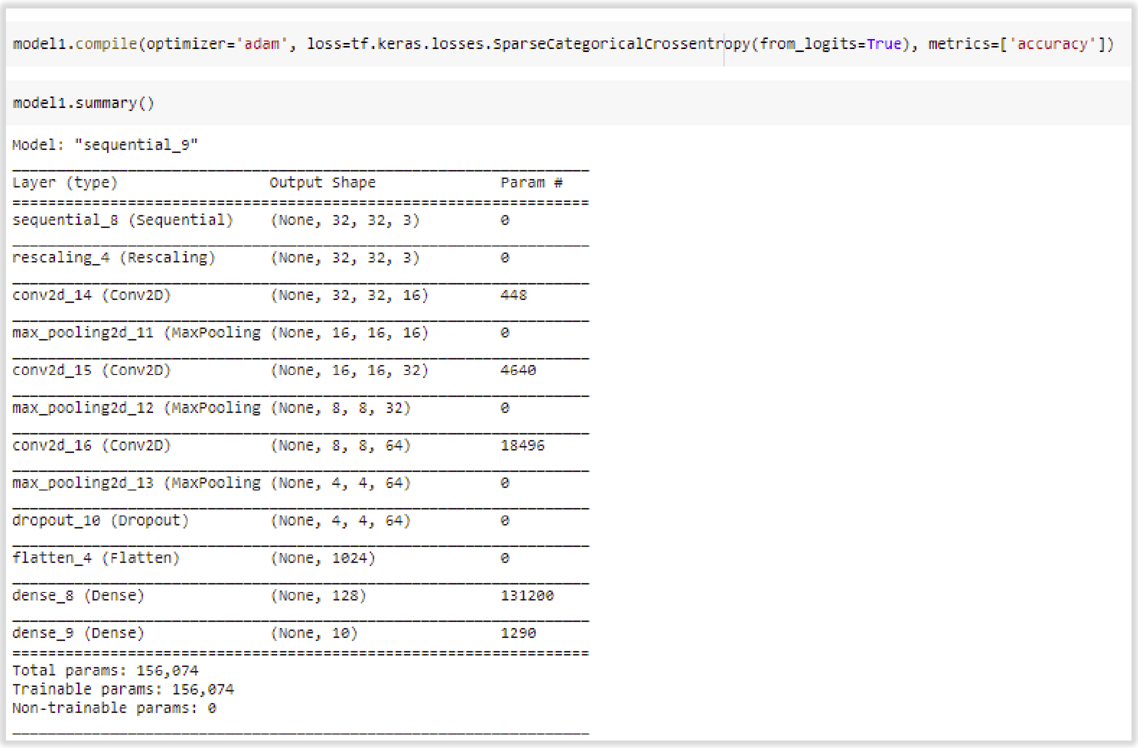


CNN Model creation using existing use case

```
[63] num_classes = 10
```

```
[64] model1 = Sequential([
       data_augmentation,
       layers.experimental.preprocessing.Rescaling(1./255),
       layers.Conv2D(16, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Conv2D(32, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Conv2D(64, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Dropout(0.2),
       layers.Flatten(),
       layers.Dense(128, activation='relu'),
       layers.Dense(num_classes)
     ])
```

- Compiling the model

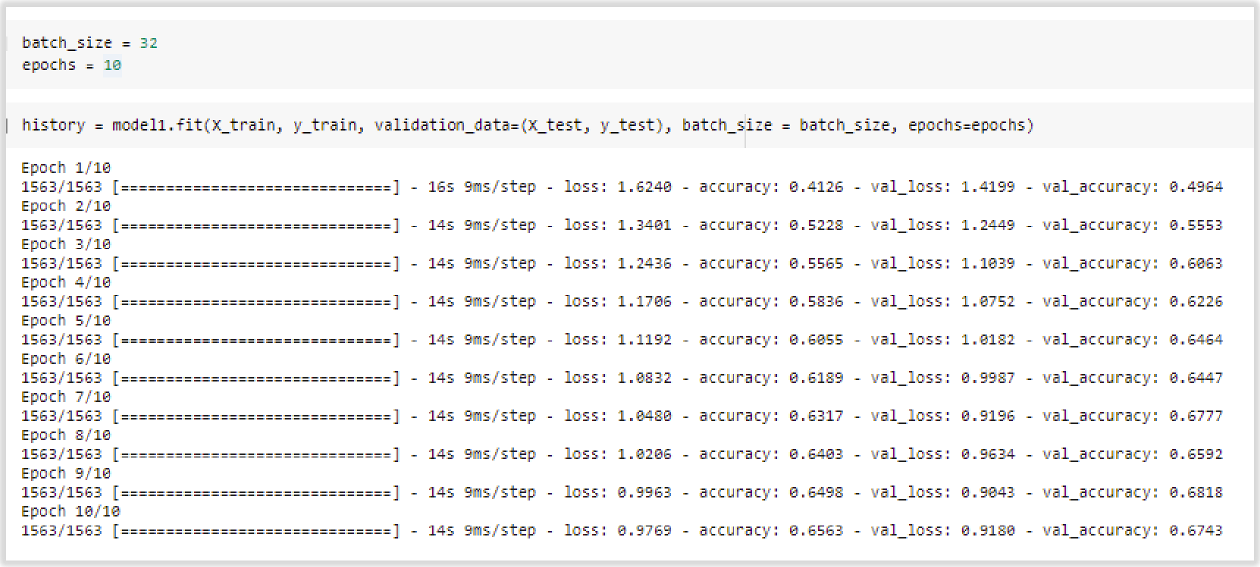


```
model1.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

```
model1.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_8 (Sequential)    (None, 32, 32, 3)         0
_________________________________________________________________
rescaling_4 (Rescaling)      (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 32, 32, 16)        448
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 32)        4640
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
dropout_10 (Dropout)         (None, 4, 4, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               131200
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 156,074
Trainable params: 156,074
Non-trainable params: 0
_________________________________________________________________
```

- Fitting the model

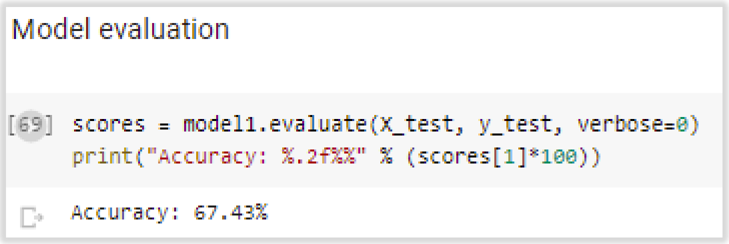


```
batch_size = 32
epochs = 10
```

```
history = model1.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size = batch_size, epochs=epochs)
```

```
Epoch 1/10
1563/1563 [==============================] - 16s 9ms/step - loss: 1.6240 - accuracy: 0.4126 - val_loss: 1.4199 - val_accuracy: 0.4964
Epoch 2/10
1563/1563 [==============================] - 14s 9ms/step - loss: 1.3401 - accuracy: 0.5228 - val_loss: 1.2449 - val_accuracy: 0.5553
Epoch 3/10
1563/1563 [==============================] - 14s 9ms/step - loss: 1.2436 - accuracy: 0.5565 - val_loss: 1.1039 - val_accuracy: 0.6063
Epoch 4/10
1563/1563 [==============================] - 14s 9ms/step - loss: 1.1706 - accuracy: 0.5836 - val_loss: 1.0752 - val_accuracy: 0.6226
Epoch 5/10
1563/1563 [==============================] - 14s 9ms/step - loss: 1.1192 - accuracy: 0.6055 - val_loss: 1.0182 - val_accuracy: 0.6464
Epoch 6/10
1563/1563 [==============================] - 14s 9ms/step - loss: 1.0832 - accuracy: 0.6189 - val_loss: 0.9987 - val_accuracy: 0.6447
Epoch 7/10
1563/1563 [==============================] - 14s 9ms/step - loss: 1.0480 - accuracy: 0.6317 - val_loss: 0.9196 - val_accuracy: 0.6777
Epoch 8/10
1563/1563 [==============================] - 14s 9ms/step - loss: 1.0206 - accuracy: 0.6403 - val_loss: 0.9634 - val_accuracy: 0.6592
Epoch 9/10
1563/1563 [==============================] - 14s 9ms/step - loss: 0.9963 - accuracy: 0.6498 - val_loss: 0.9043 - val_accuracy: 0.6818
Epoch 10/10
1563/1563 [==============================] - 14s 9ms/step - loss: 0.9769 - accuracy: 0.6563 - val_loss: 0.9180 - val_accuracy: 0.6743
```

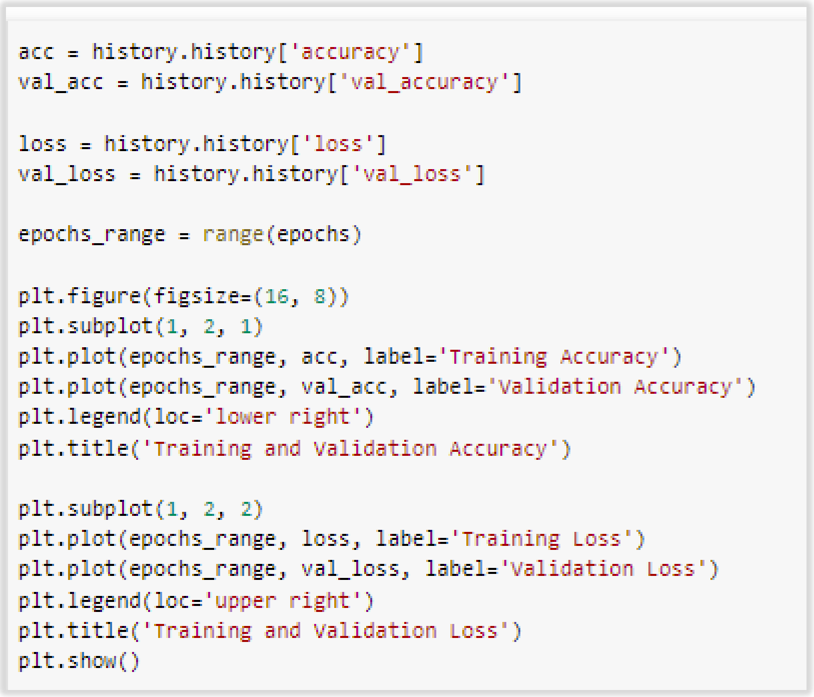
- Model performance evaluation



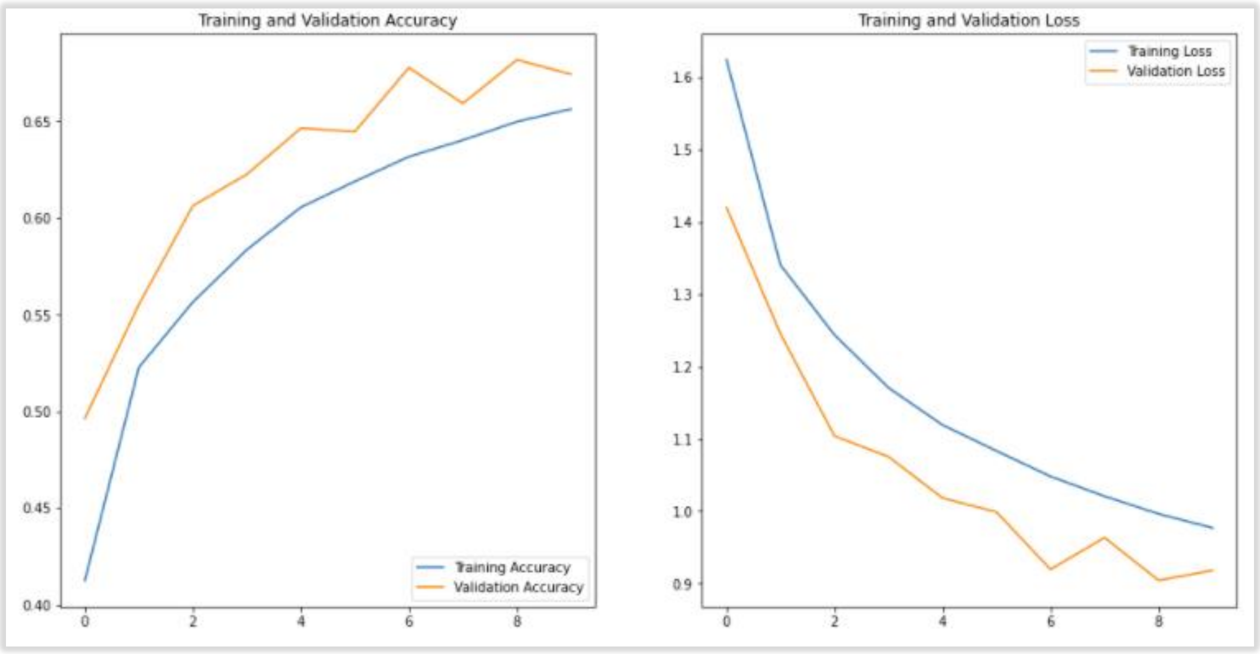
Model evaluation

```
[69] scores = model1.evaluate(X_test, y_test, verbose=0)
     print("Accuracy: %.2f%%" % (scores[1]*100))

Accuracy: 67.43%
```

- Visualize the training and testing accuracy and loss

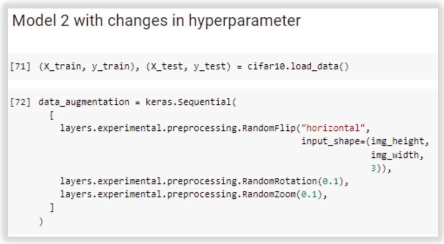


```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



2. Model Evaluation after changing the hyperparameters

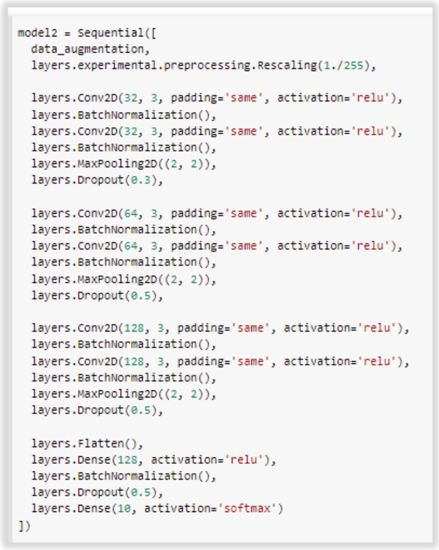
- Reading the CIFAR-10 dataset and splitting data into training and testing set
- Data Augmentation



Model 2 with changes in hyperparameter

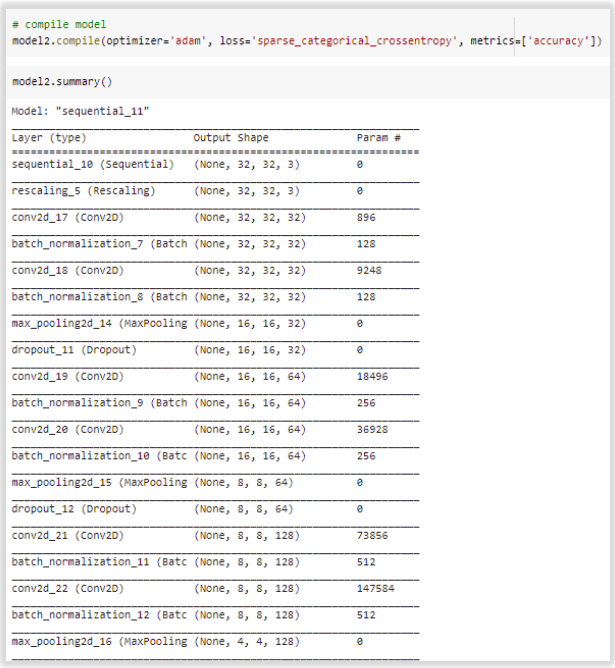
```
[71] (X_train, y_train), (X_test, y_test) = cifar10.load_data()

[72] data_augmentation = keras.Sequential(
       [
         layers.experimental.preprocessing.RandomFlip("horizontal",
                                                      input_shape=(img_height,
                                                                   img_width,
                                                                   3)),
         layers.experimental.preprocessing.RandomRotation(0.1),
         layers.experimental.preprocessing.RandomZoom(0.1),
       ]
     )
```

- CNN Model Creation using Keras by setting up the layers



```
model2 = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),

  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D((2, 2)),
  layers.Dropout(0.3),

  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D((2, 2)),
  layers.Dropout(0.5),

  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.BatchNormalization(),
  layers.MaxPooling2D((2, 2)),
  layers.Dropout(0.5),

  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.BatchNormalization(),
  layers.Dropout(0.5),
  layers.Dense(10, activation='softmax')
])
```

- Compiling the model

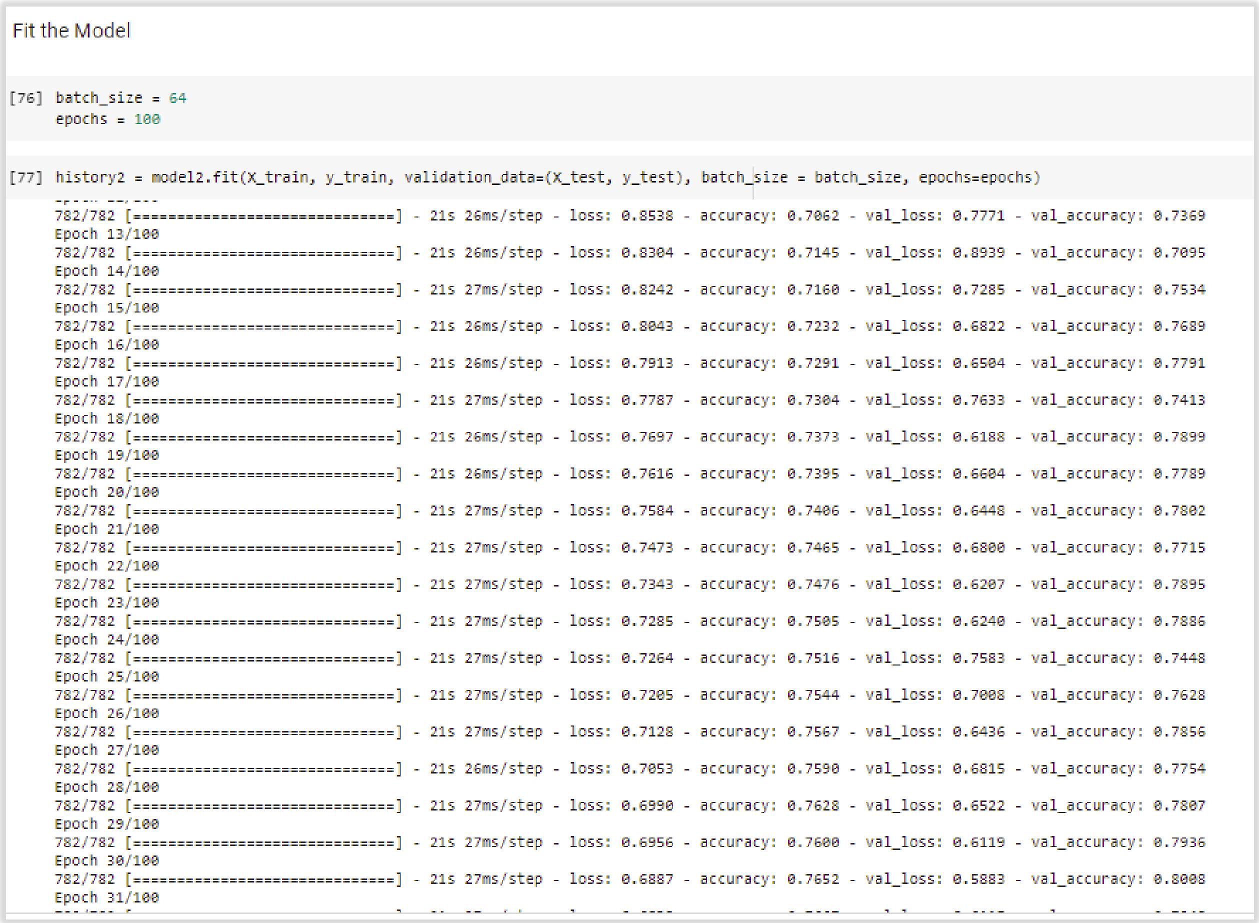


```
# compile model
model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
model2.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
sequential_10 (Sequential)   (None, 32, 32, 3)         0         
_________________________________________________________________
rescaling_5 (Rescaling)      (None, 32, 32, 3)         0         
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 32, 32, 32)        896       
_________________________________________________________________
batch_normalization_7 (Batch (None, 32, 32, 32)        128       
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 32, 32, 32)        9248      
_________________________________________________________________
batch_normalization_8 (Batch (None, 32, 32, 32)        128       
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 16, 16, 32)        0         
_________________________________________________________________
dropout_11 (Dropout)         (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 16, 16, 64)        18496     
_________________________________________________________________
batch_normalization_9 (Batch (None, 16, 16, 64)        256       
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 16, 16, 64)        36928     
_________________________________________________________________
batch_normalization_10 (Batc (None, 16, 16, 64)        256       
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 8, 8, 64)          0         
_________________________________________________________________
dropout_12 (Dropout)         (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 8, 8, 128)         73856     
_________________________________________________________________
batch_normalization_11 (Batc (None, 8, 8, 128)         512       
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 8, 8, 128)         147584    
_________________________________________________________________
batch_normalization_12 (Batc (None, 8, 8, 128)         512       
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 4, 4, 128)         0         
_________________________________________________________________
```

- Fitting the model



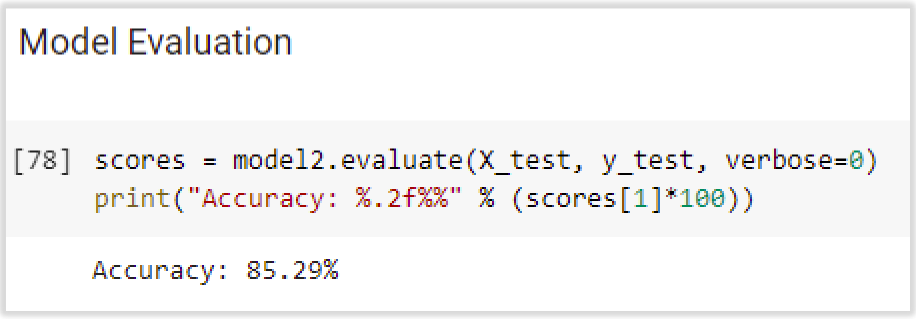
Fit the Model

```
[76] batch_size = 64
     epochs = 100
```

```
[77] history2 = model2.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size = batch_size, epochs=epochs)
```

```
782/782 [==============================] - 21s 26ms/step - loss: 0.8538 - accuracy: 0.7062 - val_loss: 0.7771 - val_accuracy: 0.7369
Epoch 13/100
782/782 [==============================] - 21s 26ms/step - loss: 0.8304 - accuracy: 0.7145 - val_loss: 0.8939 - val_accuracy: 0.7095
Epoch 14/100
782/782 [==============================] - 21s 27ms/step - loss: 0.8242 - accuracy: 0.7160 - val_loss: 0.7285 - val_accuracy: 0.7534
Epoch 15/100
782/782 [==============================] - 21s 26ms/step - loss: 0.8043 - accuracy: 0.7232 - val_loss: 0.6822 - val_accuracy: 0.7689
Epoch 16/100
782/782 [==============================] - 21s 26ms/step - loss: 0.7913 - accuracy: 0.7291 - val_loss: 0.6504 - val_accuracy: 0.7791
Epoch 17/100
782/782 [==============================] - 21s 27ms/step - loss: 0.7787 - accuracy: 0.7304 - val_loss: 0.7633 - val_accuracy: 0.7413
Epoch 18/100
782/782 [==============================] - 21s 26ms/step - loss: 0.7697 - accuracy: 0.7373 - val_loss: 0.6188 - val_accuracy: 0.7899
Epoch 19/100
782/782 [==============================] - 21s 26ms/step - loss: 0.7616 - accuracy: 0.7395 - val_loss: 0.6604 - val_accuracy: 0.7789
Epoch 20/100
782/782 [==============================] - 21s 27ms/step - loss: 0.7584 - accuracy: 0.7406 - val_loss: 0.6448 - val_accuracy: 0.7802
Epoch 21/100
782/782 [==============================] - 21s 27ms/step - loss: 0.7473 - accuracy: 0.7465 - val_loss: 0.6800 - val_accuracy: 0.7715
Epoch 22/100
782/782 [==============================] - 21s 27ms/step - loss: 0.7343 - accuracy: 0.7476 - val_loss: 0.6207 - val_accuracy: 0.7895
Epoch 23/100
782/782 [==============================] - 21s 27ms/step - loss: 0.7285 - accuracy: 0.7505 - val_loss: 0.6240 - val_accuracy: 0.7886
Epoch 24/100
782/782 [==============================] - 21s 27ms/step - loss: 0.7264 - accuracy: 0.7516 - val_loss: 0.7583 - val_accuracy: 0.7448
Epoch 25/100
782/782 [==============================] - 21s 27ms/step - loss: 0.7205 - accuracy: 0.7544 - val_loss: 0.7008 - val_accuracy: 0.7628
Epoch 26/100
782/782 [==============================] - 21s 27ms/step - loss: 0.7128 - accuracy: 0.7567 - val_loss: 0.6436 - val_accuracy: 0.7856
Epoch 27/100
782/782 [==============================] - 21s 26ms/step - loss: 0.7053 - accuracy: 0.7590 - val_loss: 0.6815 - val_accuracy: 0.7754
Epoch 28/100
782/782 [==============================] - 21s 27ms/step - loss: 0.6990 - accuracy: 0.7628 - val_loss: 0.6522 - val_accuracy: 0.7807
Epoch 29/100
782/782 [==============================] - 21s 27ms/step - loss: 0.6956 - accuracy: 0.7600 - val_loss: 0.6119 - val_accuracy: 0.7936
Epoch 30/100
782/782 [==============================] - 21s 27ms/step - loss: 0.6887 - accuracy: 0.7652 - val_loss: 0.5883 - val_accuracy: 0.8008
Epoch 31/100
```

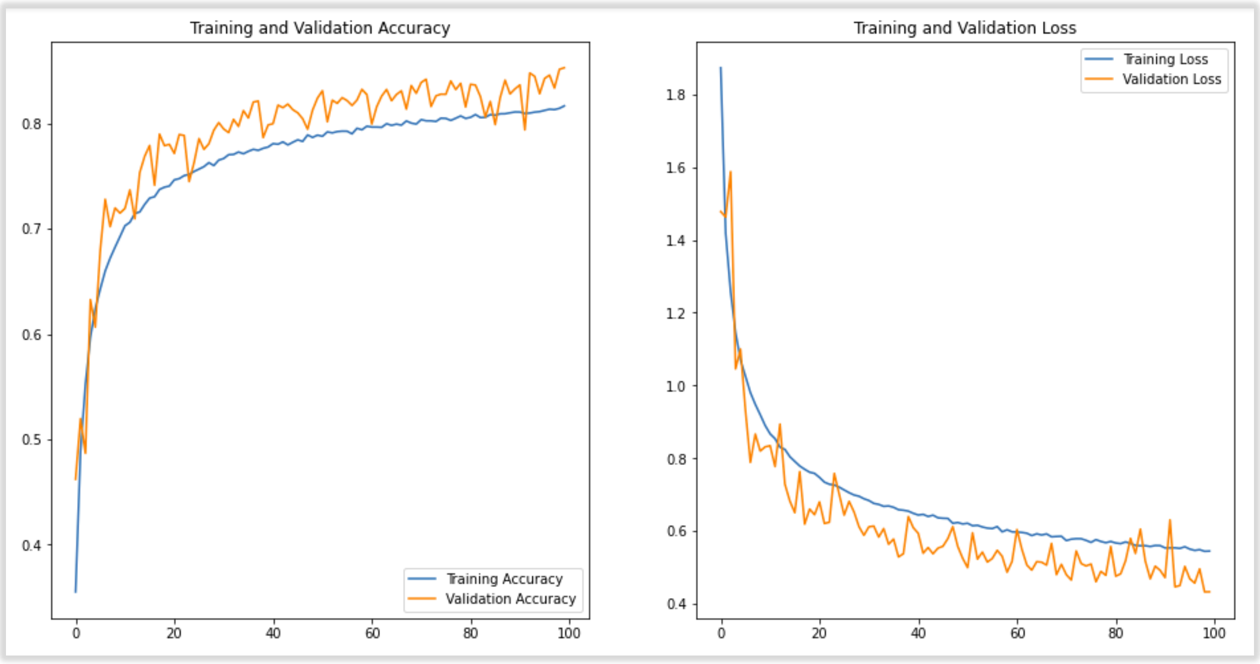
- Model performance evaluation is **85.29%**



Model Evaluation

```
[78] scores = model2.evaluate(X_test, y_test, verbose=0)
     print("Accuracy: %.2f%%" % (scores[1]*100))

     Accuracy: 85.29%
```

- Visualize the training and testing accuracy and loss



3. Final observation comments

We can successfully conclude that our model 2 performed very well as compared to the existing use case model 1. We got significant improvement **from 67.43% to 85.29%.**

The reasons are as below:

1. For building the Model – We used **denser CNN, Maxpooling, and Dense Layers.**
2. For Activation Function – We used ReLU (in CNN layers for handling image pixels) and **Softmax (for final classification) function for the output layer.**
3. For handling Overfitting (Regularizing) – **We used DropOut Layers** to get rid of extra neurons.
4. For normalizing/standardizing the inputs between the layers (within the network) and hence accelerating the training, providing regularization, and reducing the generalization error – **Use of Batch Normalization Layers** proved to be the best fit.

Video Link

- https://youtu.be/WaLDboylKoM

Conclusion

1. Lessons Learnt

- I developed a deep understanding of the Convolutional Neural Networks after doing this ICP.
- The Model summary provides Total params count which is also a very important parameter to be considered when building a CNN model.

2. Challenges Faced

- To improve the accuracy of existing model, use of correct hyperparameters like activation function, convolutional layers, and number of epochs was a challenge. I did build the model a few times and understood which can be a better fit.
- I developed understanding of BatchNormalization function which helps to normalize the data between various convolutional layers.