CS5590/490 - Python and Deep Learning Programming
In Class Programming Report - 11
Class ID 24 - Anurag Thantharate
Date Submitted: 04/19/2020

---

## 1. Follow the instruction below and then report how the performance changed.(apply all at once)

Convolutional input layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
Dropout layer at 20%.
Convolutional layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
Max Pool layer with size 2×2.
Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
Dropout layer at 20%.
Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
Max Pool layer with size 2×2.
Convolutional layer, 128 feature maps with a size of 3×3 and a rectifier activation function.
Dropout layer at 20%.
Convolutional layer,128 feature maps with a size of 3×3 and a rectifier activation function.
Max Pool layer with size 2×2.
Flatten layer.
Dropout layer at 20%.
Fully connected layer with 1024 units and a rectifier activation function.
Dropout layer at 20%.
Fully connected layer with 512 units and a rectifier activation function.
Dropout layer at 20%.
Fully connected output layer with 10 units and a SoftMax activation function

## Did the performance change? 2. Change the previous model into Keras API model.

```
[ ] model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=128)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/15
50000/50000 [==============================] - 30s 600us/step - loss: 1.7572 - accuracy: 0.3615 - val_loss: 1.3732 - val_accuracy: 0.5041
Epoch 2/15
50000/50000 [==============================] - 30s 598us/step - loss: 1.3447 - accuracy: 0.5173 - val_loss: 1.1809 - val_accuracy: 0.5822
Epoch 3/15
50000/50000 [==============================] - 30s 603us/step - loss: 1.2004 - accuracy: 0.5727 - val_loss: 1.0834 - val_accuracy: 0.6140
Epoch 4/15
50000/50000 [==============================] - 30s 604us/step - loss: 1.1176 - accuracy: 0.6047 - val_loss: 1.0183 - val_accuracy: 0.6445
Epoch 5/15
50000/50000 [==============================] - 30s 599us/step - loss: 1.0532 - accuracy: 0.6275 - val_loss: 0.9677 - val_accuracy: 0.6663
Epoch 6/15
50000/50000 [==============================] - 30s 601us/step - loss: 0.9915 - accuracy: 0.6470 - val_loss: 0.9266 - val_accuracy: 0.6774
Epoch 7/15
50000/50000 [==============================] - 30s 600us/step - loss: 0.9479 - accuracy: 0.6629 - val_loss: 0.9158 - val_accuracy: 0.6835
Epoch 8/15
50000/50000 [==============================] - 30s 599us/step - loss: 0.9111 - accuracy: 0.6771 - val_loss: 0.8842 - val_accuracy: 0.6904
Epoch 9/15
50000/50000 [==============================] - 30s 599us/step - loss: 0.8827 - accuracy: 0.6876 - val_loss: 0.8588 - val_accuracy: 0.6995
Epoch 10/15
50000/50000 [==============================] - 30s 600us/step - loss: 0.8641 - accuracy: 0.6932 - val_loss: 0.8763 - val_accuracy: 0.6942
Epoch 11/15
50000/50000 [==============================] - 30s 600us/step - loss: 0.8331 - accuracy: 0.7046 - val_loss: 0.8609 - val_accuracy: 0.6978
Epoch 12/15
50000/50000 [==============================] - 30s 600us/step - loss: 0.8185 - accuracy: 0.7090 - val_loss: 0.8301 - val_accuracy: 0.7102
Epoch 13/15
50000/50000 [==============================] - 30s 601us/step - loss: 0.7913 - accuracy: 0.7209 - val_loss: 0.8436 - val_accuracy: 0.7034
Epoch 14/15
50000/50000 [==============================] - 30s 599us/step - loss: 0.7725 - accuracy: 0.7253 - val_loss: 0.8170 - val_accuracy: 0.7137
Epoch 15/15
50000/50000 [==============================] - 30s 600us/step - loss: 0.7508 - accuracy: 0.7329 - val_loss: 0.8030 - val_accuracy: 0.7181
<keras.callbacks.callbacks.History at 0x7f9d1a593630>
```

Final evaluation of the model

Provided Code

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

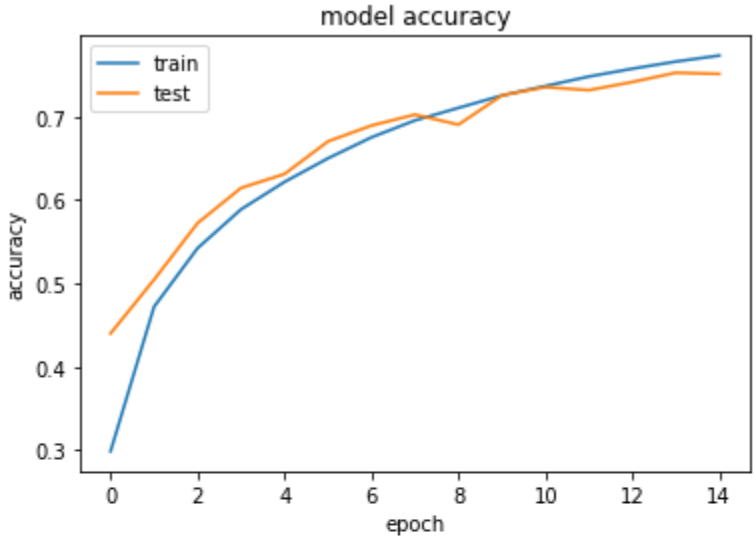
```
Accuracy: 71.81%
```

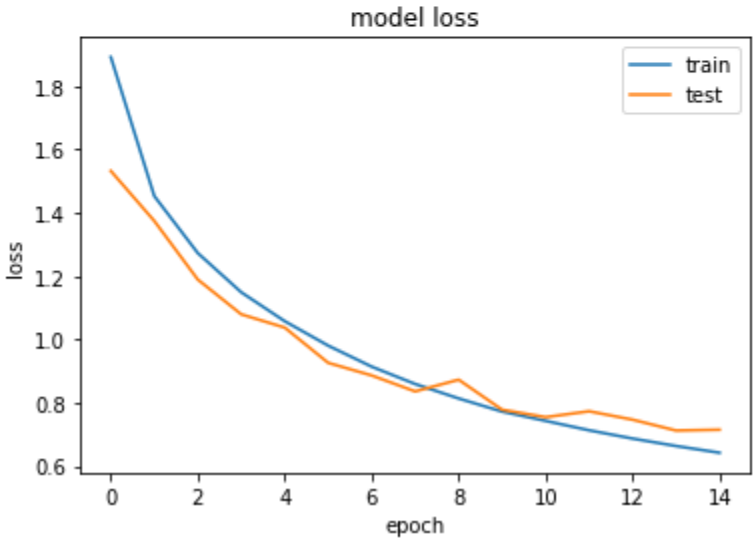
```
10
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
dropout_7 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
dropout_9 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dropout_10 (Dropout)         (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dropout_11 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_12 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                5130
=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
None
Train on 50000 samples, validate on 10000 samples
Epoch 1/15
```

# Adding CNN Layers

```
Non-trainable params: 0
_________________________________________________________________
None
Train on 50000 samples, validate on 10000 samples
Epoch 1/15
50000/50000 [==============================] - 16s 322us/step - loss: 1.8915 - accuracy: 0.2989 - val_loss: 1.5320 - val_accuracy: 0.4401
Epoch 2/15
50000/50000 [==============================] - 16s 311us/step - loss: 1.4525 - accuracy: 0.4721 - val_loss: 1.3740 - val_accuracy: 0.5046
Epoch 3/15
50000/50000 [==============================] - 16s 313us/step - loss: 1.2727 - accuracy: 0.5423 - val_loss: 1.1891 - val_accuracy: 0.5723
Epoch 4/15
50000/50000 [==============================] - 16s 313us/step - loss: 1.1489 - accuracy: 0.5888 - val_loss: 1.0794 - val_accuracy: 0.6145
Epoch 5/15
50000/50000 [==============================] - 16s 315us/step - loss: 1.0576 - accuracy: 0.6217 - val_loss: 1.0374 - val_accuracy: 0.6312
Epoch 6/15
50000/50000 [==============================] - 16s 314us/step - loss: 0.9805 - accuracy: 0.6500 - val_loss: 0.9262 - val_accuracy: 0.6703
Epoch 7/15
50000/50000 [==============================] - 16s 312us/step - loss: 0.9141 - accuracy: 0.6752 - val_loss: 0.8864 - val_accuracy: 0.6893
Epoch 8/15
50000/50000 [==============================] - 16s 316us/step - loss: 0.8594 - accuracy: 0.6954 - val_loss: 0.8361 - val_accuracy: 0.7024
Epoch 9/15
50000/50000 [==============================] - 16s 314us/step - loss: 0.8138 - accuracy: 0.7104 - val_loss: 0.8730 - val_accuracy: 0.6904
Epoch 10/15
50000/50000 [==============================] - 16s 310us/step - loss: 0.7723 - accuracy: 0.7253 - val_loss: 0.7782 - val_accuracy: 0.7254
Epoch 11/15
50000/50000 [==============================] - 15s 308us/step - loss: 0.7430 - accuracy: 0.7367 - val_loss: 0.7548 - val_accuracy: 0.7354
Epoch 12/15
50000/50000 [==============================] - 15s 307us/step - loss: 0.7133 - accuracy: 0.7480 - val_loss: 0.7733 - val_accuracy: 0.7317
Epoch 13/15
50000/50000 [==============================] - 16s 313us/step - loss: 0.6872 - accuracy: 0.7574 - val_loss: 0.7468 - val_accuracy: 0.7415
Epoch 14/15
50000/50000 [==============================] - 15s 310us/step - loss: 0.6637 - accuracy: 0.7660 - val_loss: 0.7125 - val_accuracy: 0.7526
Epoch 15/15
50000/50000 [==============================] - 15s 308us/step - loss: 0.6424 - accuracy: 0.7732 - val_loss: 0.7152 - val_accuracy: 0.7512
Accuracy: 75.12%
(10000, 32, 32, 3)
```

Improved Accuracy



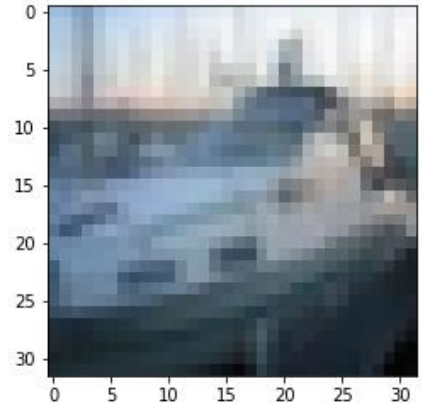
2. Change the previous model into Keras API model.

```
#Functional API
from keras.models import Model
from keras.layers import Input

a = Input(shape=(32,32,3))
x=Conv2D(32,(3,3),padding="same",activation='relu')(a)
x=Dropout(0.2)(x)
x=Conv2D(256, (3, 3), activation='relu', padding='same')(x)
x=MaxPooling2D(pool_size=(2, 2))(x)
x=Dropout(0.5)(x)
x=Flatten()(x)
x=Dense(512, activation='relu', kernel_constraint=maxnorm(3))(x)
x=Dropout(0.3)(x)
x=Dense(num_classes, activation='softmax')(x)
modelapi = Model(inputs=a,output=x)

epochs = 10
lrate = 0.001
decay = lrate/epochs
sgd = Adam(lr=lrate)
modelapi.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
modelapi.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=128)
```

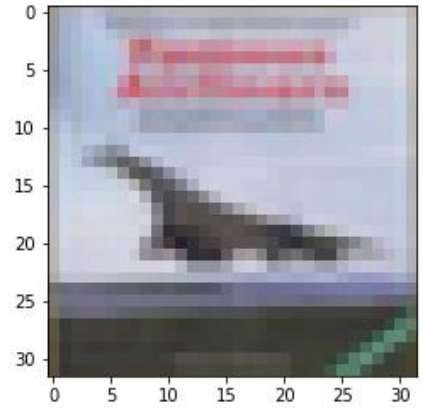
3. predict the first 4 image of the test data. Then, print the actual label for those 4 images (label means the probability associated with them) to check if the model predicted correctly or not

```
# Predicting the first four images of dataset
image=model.predict_classes(X_train[[1],:])
print(image[0])

predicted_image=y_test
import matplotlib.pyplot as plt
for i in range(2,6):
  plt.imshow(X_test[i,:,:])
  plt.show()
  image=model.predict_classes(X_test[[i],:])
  print("actual",predicted_image[i],"predicted",image[0])
```

Index position prediction is correct for the provided dataset and built model.

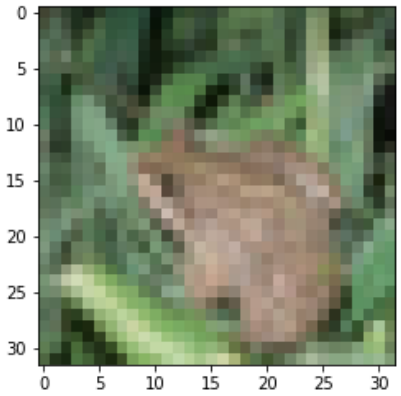


actual [0. 0. 0. 0. 0. 0. 0. 1. 0.] predicted 8

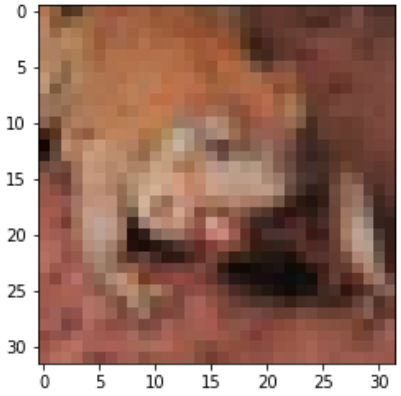


actual [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.] predicted 0

actual [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.] predicted 0



actual [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.] predicted 6



actual [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.] predicted 6