Project Part – 3

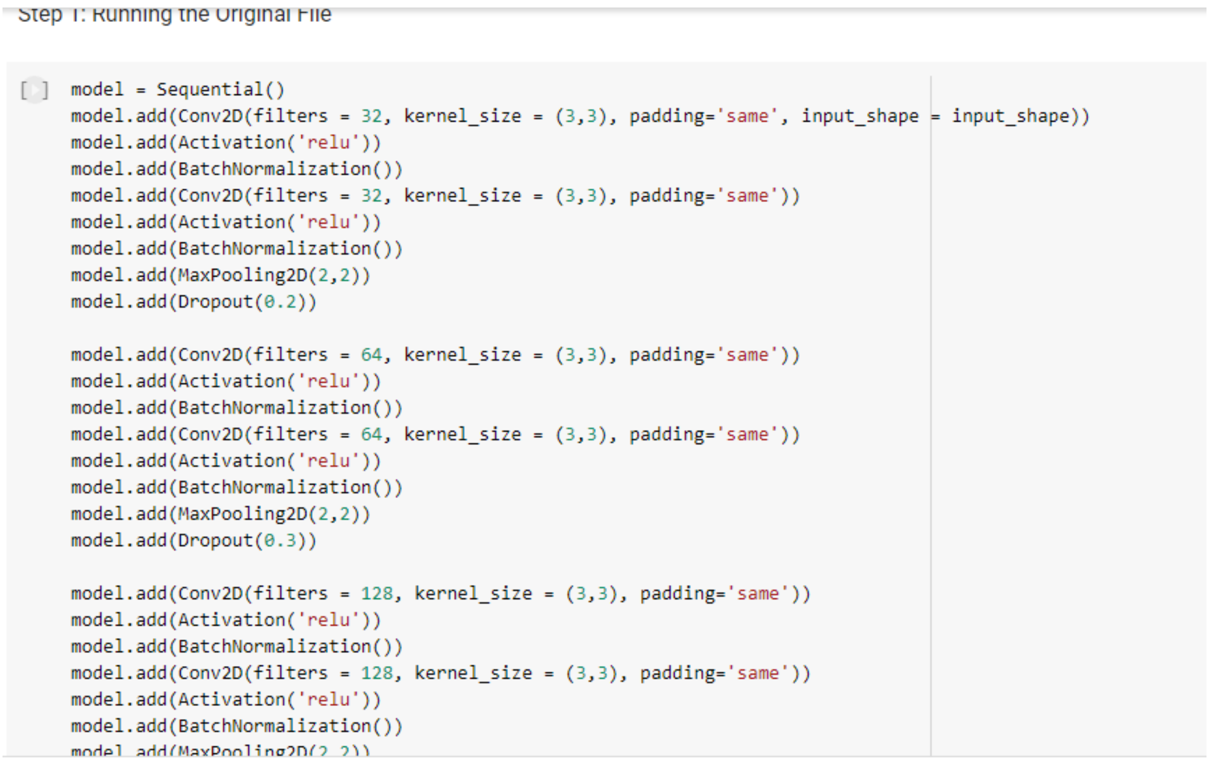
- hshah78@asu.edu

1. Introduction:

In this project we are classifying CIFAR-10 dataset using Convolutional Neural Network. There are 12 hidden layers within the CNN used which have various parameters such as kernel size, Activation Function and Batch Normalization. We are experimenting with different parameters and optimizers and finding which configuration gives the better result.

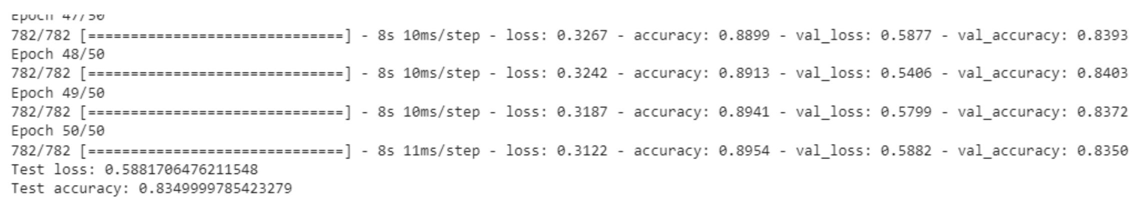
2. Steps and Results:

Running the Baseline code: The initial configuration of the CNN is already provided and we are running based on the given parameters.



Step 1: Running the Original File

```
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same', input_shape = input_shape))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.2))

model.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))

model.add(Conv2D(filters = 128, kernel_size = (3,3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters = 128, kernel_size = (3,3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(2,2))
```



```
Epoch 47/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3267 - accuracy: 0.8899 - val_loss: 0.5877 - val_accuracy: 0.8393
Epoch 48/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3242 - accuracy: 0.8913 - val_loss: 0.5406 - val_accuracy: 0.8403
Epoch 49/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3187 - accuracy: 0.8941 - val_loss: 0.5799 - val_accuracy: 0.8372
Epoch 50/50
782/782 [==============================] - 8s 11ms/step - loss: 0.3122 - accuracy: 0.8954 - val_loss: 0.5882 - val_accuracy: 0.8350
Test loss: 0.5881706476211548
Test accuracy: 0.8349999785423279
```

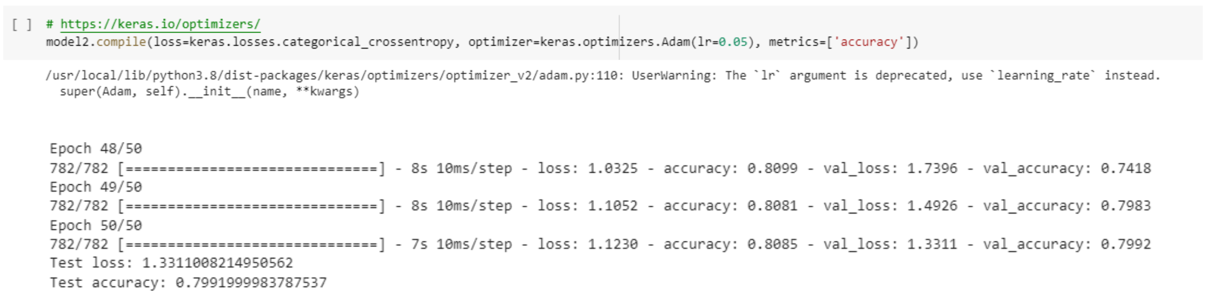
Test Loss: 0.588

Test Accuracy: 0.8349

a. Changing the Learning Rate: The learning rate of the Neural Network should not be too high or too low so that the model converges and not miss the minimum

a. Learning Rate: 0.05

When the learning rate is set to 0.05 it quickly converges but misses the minimum. In this case the Test Loss has increased and Test accuracy has decreased which may mean that the model has missed the minimum and went ahead in the curve.



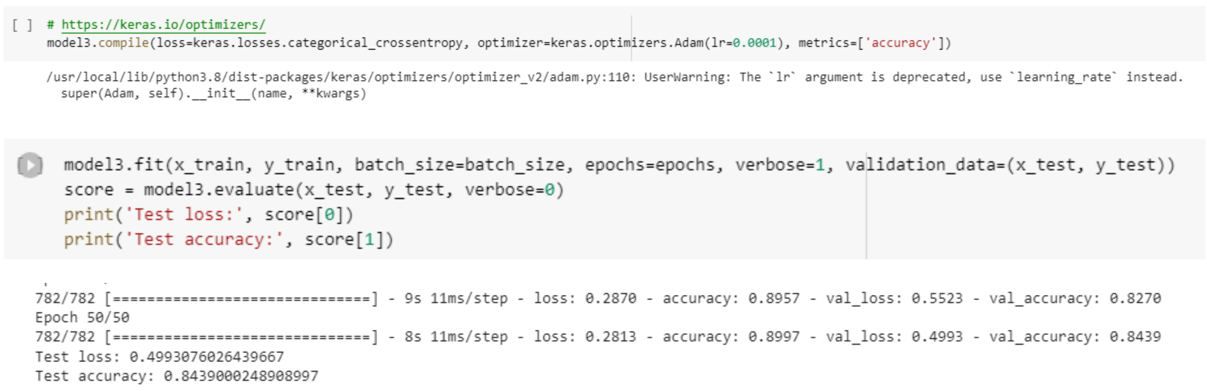
```
[ ] # https://keras.io/optimizers/
    model2.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.05), metrics=['accuracy'])

    /usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
      super(Adam, self).__init__(name, **kwargs)


     Epoch 48/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.0325 - accuracy: 0.8099 - val_loss: 1.7396 - val_accuracy: 0.7418
     Epoch 49/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1052 - accuracy: 0.8081 - val_loss: 1.4926 - val_accuracy: 0.7983
     Epoch 50/50
     782/782 [==============================] - 7s 10ms/step - loss: 1.1230 - accuracy: 0.8085 - val_loss: 1.3311 - val_accuracy: 0.7992
     Test loss: 1.3311008214950562
     Test accuracy: 0.7991999983787537
```

Test Loss: 1.33

Test Accuracy: 0.799

b. Learning Rate: 0.0001

In this case the test Loss has reduced and the test Accuracy has increased, when the learning rate is reduced it takes baby steps and might require more iterations to converge.



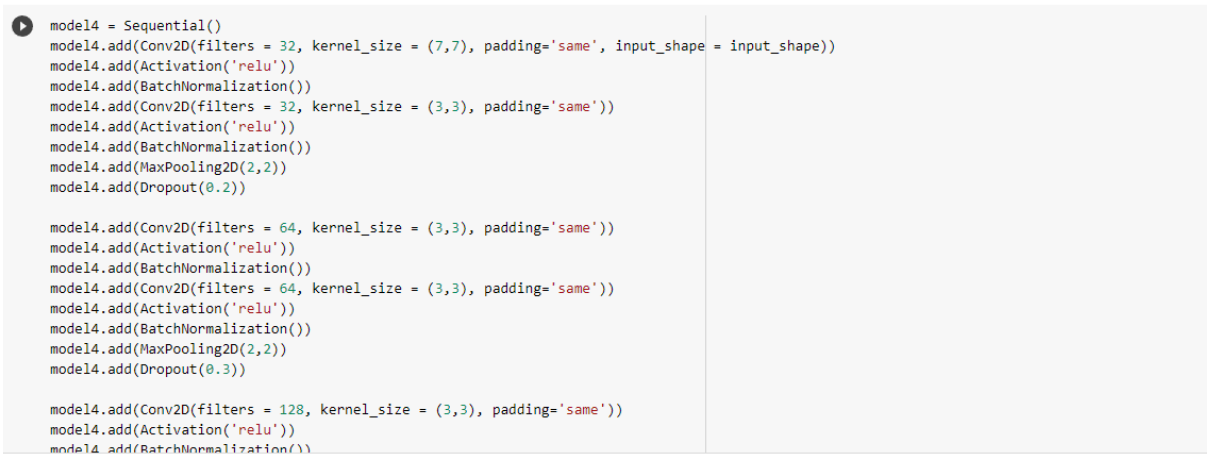
```
[ ] # https://keras.io/optimizers/
    model3.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.0001), metrics=['accuracy'])

    /usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
      super(Adam, self).__init__(name, **kwargs)


    model3.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, y_test))
    score = model3.evaluate(x_test, y_test, verbose=0)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])

   782/782 [==============================] - 9s 11ms/step - loss: 0.2870 - accuracy: 0.8957 - val_loss: 0.5523 - val_accuracy: 0.8270
   Epoch 50/50
   782/782 [==============================] - 8s 11ms/step - loss: 0.2813 - accuracy: 0.8997 - val_loss: 0.4993 - val_accuracy: 0.8439
   Test loss: 0.4993076026439667
   Test accuracy: 0.8439000248908997
```

Test Loss: 0.499

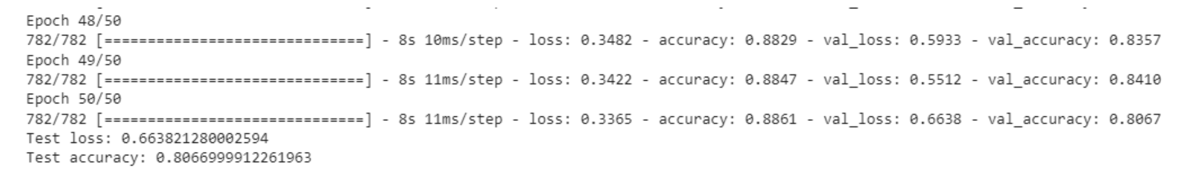
Test Accuracy: 0.843

b. Changing the first Kernel Layer to 7x7

CNN work on locality and map a particular locality in input layer to the next hidden layer and so on. If the kernel size is too big then the CNN will behave as a normal neural network and the convolutional property will vanish. In this case the test loss has increased and the test accuracy has decreased slightly.



```
model4 = Sequential()
model4.add(Conv2D(filters = 32, kernel_size = (7,7), padding='same', input_shape = input_shape))
model4.add(Activation('relu'))
model4.add(BatchNormalization())
model4.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same'))
model4.add(Activation('relu'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(2,2))
model4.add(Dropout(0.2))

model4.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
model4.add(Activation('relu'))
model4.add(BatchNormalization())
model4.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
model4.add(Activation('relu'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(2,2))
model4.add(Dropout(0.3))

model4.add(Conv2D(filters = 128, kernel_size = (3,3), padding='same'))
model4.add(Activation('relu'))
model4.add(BatchNormalization())
```



```
Epoch 48/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3482 - accuracy: 0.8829 - val_loss: 0.5933 - val_accuracy: 0.8357
Epoch 49/50
782/782 [==============================] - 8s 11ms/step - loss: 0.3422 - accuracy: 0.8847 - val_loss: 0.5512 - val_accuracy: 0.8410
Epoch 50/50
782/782 [==============================] - 8s 11ms/step - loss: 0.3365 - accuracy: 0.8861 - val_loss: 0.6638 - val_accuracy: 0.8067
Test loss: 0.663821280002594
Test accuracy: 0.8066999912261963
```

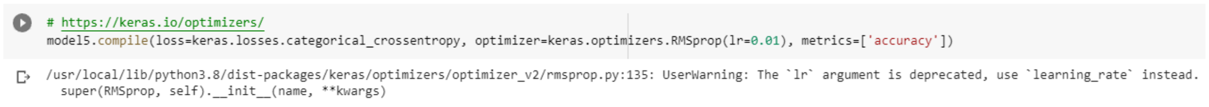
Test Loss: 0.663

Test Accuracy: 0.806

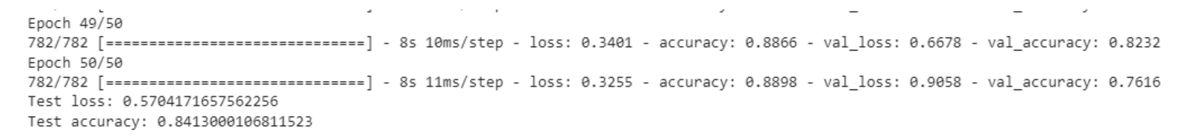
c. Changing the optimizer to RMSprop

The gist of RMSprop is to:

- Maintain a moving average of the square of gradients
- Divide the gradient by the root of this average



```
# https://keras.io/optimizers/
model5.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.RMSprop(lr=0.01), metrics=['accuracy'])

/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/rmsprop.py:135: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(RMSprop, self).__init__(name, **kwargs)
```



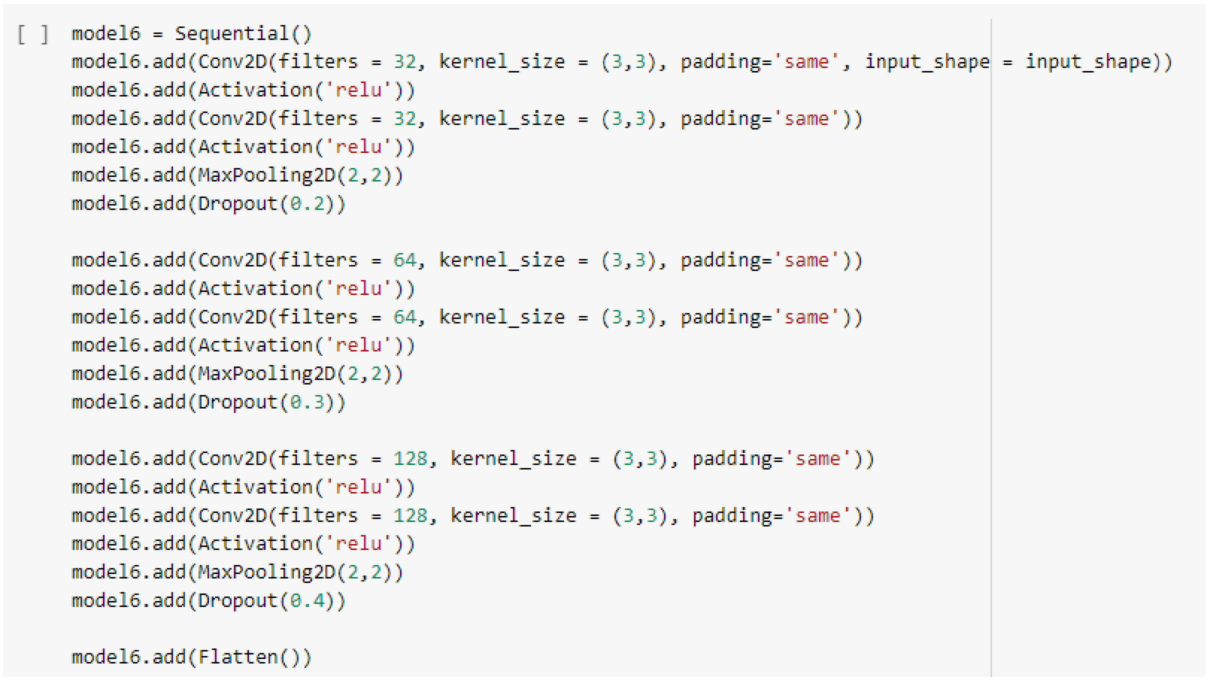
```
Epoch 49/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3401 - accuracy: 0.8866 - val_loss: 0.6678 - val_accuracy: 0.8232
Epoch 50/50
782/782 [==============================] - 8s 11ms/step - loss: 0.3255 - accuracy: 0.8898 - val_loss: 0.9058 - val_accuracy: 0.7616
Test loss: 0.5704171657562256
Test accuracy: 0.8413000106811523
```

Test Loss: 0.5704

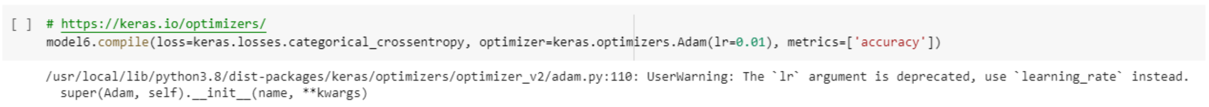
Test Accuracy: 0.8413

d. Remove all Batch normalization Layers

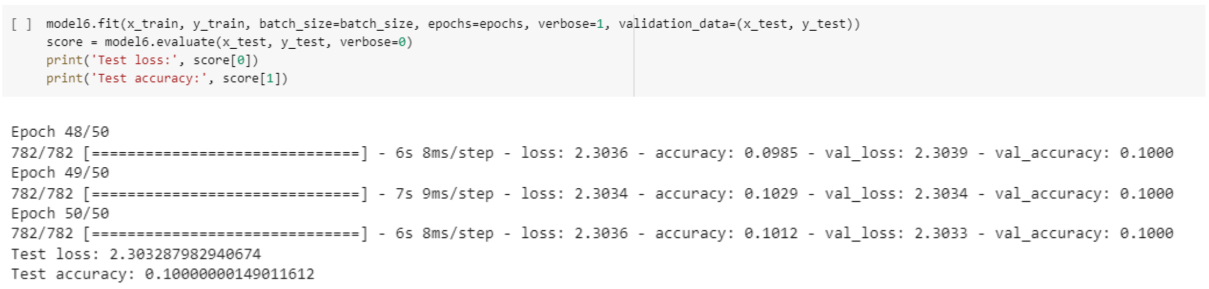
Batch Normalization Layers normalizes the whole data stream by adding some noise to the parameters. This makes the learning from the data more efficient and avoid more learning. By removing the batch normalization layers the Test Loss has increased exponentially and test accuracy has also decreased.



```
[ ]  model6 = Sequential()
     model6.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same', input_shape = input_shape))
     model6.add(Activation('relu'))
     model6.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same'))
     model6.add(Activation('relu'))
     model6.add(MaxPooling2D(2,2))
     model6.add(Dropout(0.2))

     model6.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
     model6.add(Activation('relu'))
     model6.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
     model6.add(Activation('relu'))
     model6.add(MaxPooling2D(2,2))
     model6.add(Dropout(0.3))

     model6.add(Conv2D(filters = 128, kernel_size = (3,3), padding='same'))
     model6.add(Activation('relu'))
     model6.add(Conv2D(filters = 128, kernel_size = (3,3), padding='same'))
     model6.add(Activation('relu'))
     model6.add(MaxPooling2D(2,2))
     model6.add(Dropout(0.4))

     model6.add(Flatten())
```



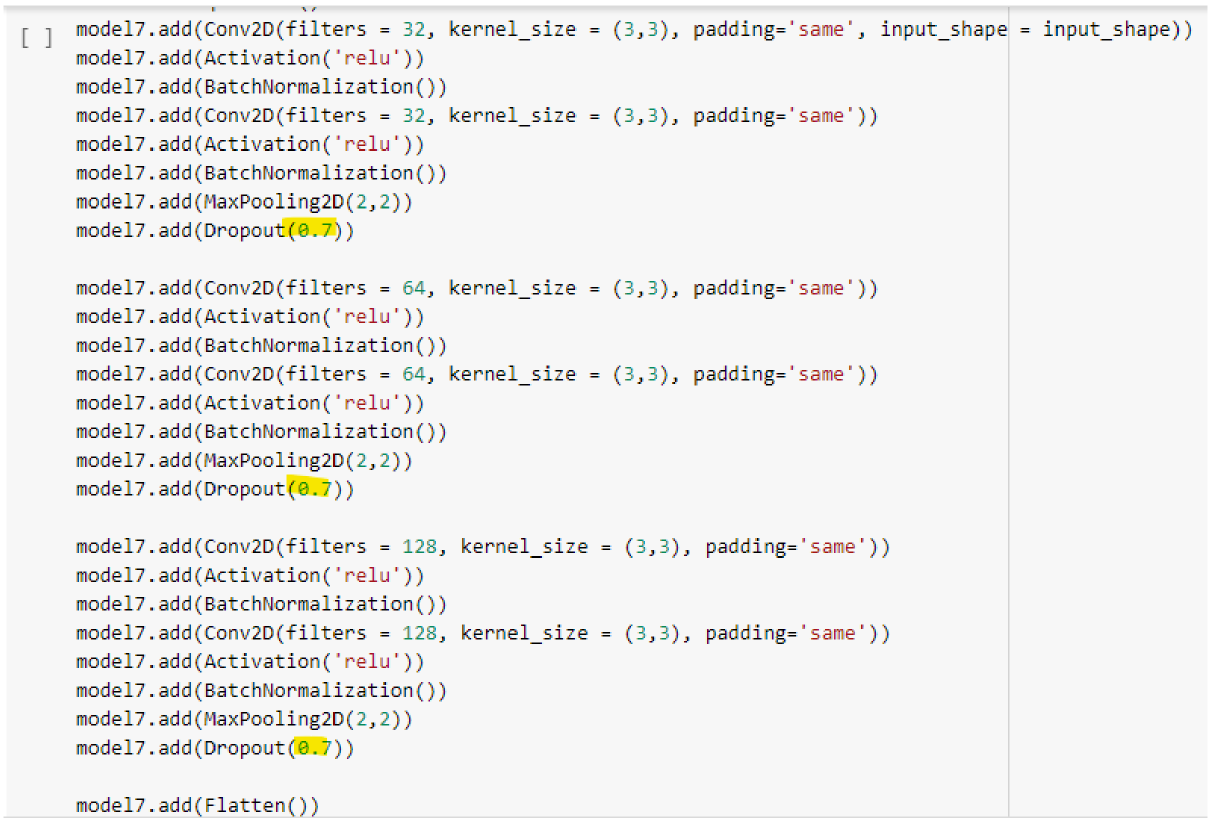
```
[ ]  # https://keras.io/optimizers/
     model6.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.01), metrics=['accuracy'])

     /usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
       super(Adam, self).__init__(name, **kwargs)
```



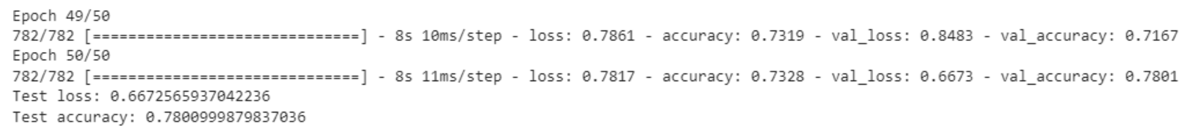
```
[ ]  model6.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, y_test))
     score = model6.evaluate(x_test, y_test, verbose=0)
     print('Test loss:', score[0])
     print('Test accuracy:', score[1])

Epoch 48/50
782/782 [==============================] - 6s 8ms/step - loss: 2.3036 - accuracy: 0.0985 - val_loss: 2.3039 - val_accuracy: 0.1000
Epoch 49/50
782/782 [==============================] - 7s 9ms/step - loss: 2.3034 - accuracy: 0.1029 - val_loss: 2.3034 - val_accuracy: 0.1000
Epoch 50/50
782/782 [==============================] - 6s 8ms/step - loss: 2.3036 - accuracy: 0.1012 - val_loss: 2.3033 - val_accuracy: 0.1000
Test loss: 2.303287982940674
Test accuracy: 0.10000000149011612
```

e. Change dropout values to 0.7

Dropout values are used to randomly close some nodes within the layer so that we don't overfit the model to the input data. A dropout value of 0.7 means that 7 nodes in 10 would be randomly turned off and won't send any data to the next layer. This increased the test loss and decreases the test accuracy.



```
[ ]  model7.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same', input_shape = input_shape))
     model7.add(Activation('relu'))
     model7.add(BatchNormalization())
     model7.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same'))
     model7.add(Activation('relu'))
     model7.add(BatchNormalization())
     model7.add(MaxPooling2D(2,2))
     model7.add(Dropout(0.7))

     model7.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
     model7.add(Activation('relu'))
     model7.add(BatchNormalization())
     model7.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
     model7.add(Activation('relu'))
     model7.add(BatchNormalization())
     model7.add(MaxPooling2D(2,2))
     model7.add(Dropout(0.7))

     model7.add(Conv2D(filters = 128, kernel_size = (3,3), padding='same'))
     model7.add(Activation('relu'))
     model7.add(BatchNormalization())
     model7.add(Conv2D(filters = 128, kernel_size = (3,3), padding='same'))
     model7.add(Activation('relu'))
     model7.add(BatchNormalization())
     model7.add(MaxPooling2D(2,2))
     model7.add(Dropout(0.7))

     model7.add(Flatten())
```



```
Epoch 49/50
782/782 [==============================] - 8s 10ms/step - loss: 0.7861 - accuracy: 0.7319 - val_loss: 0.8483 - val_accuracy: 0.7167
Epoch 50/50
782/782 [==============================] - 8s 11ms/step - loss: 0.7817 - accuracy: 0.7328 - val_loss: 0.6673 - val_accuracy: 0.7801
Test loss: 0.6672565937042236
Test accuracy: 0.7800999879837036
```

Test Loss: 0.667

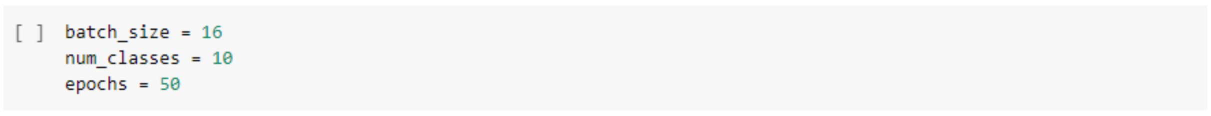
Test Accuracy: 0.7800

f. Changing the batch size

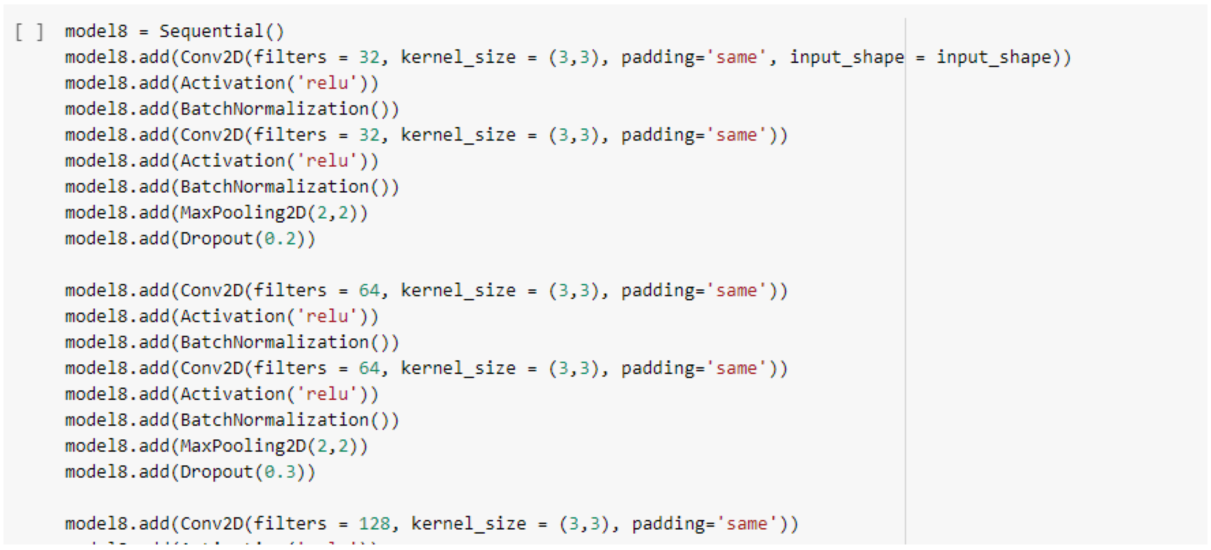
According to the batch size specified the model is trained with that many input samples.

a. Batch size: 16

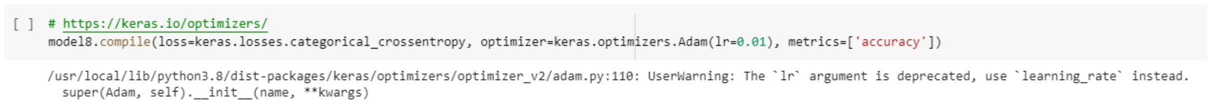
If the batch size is too small that the weights can be trained faster but it would take a longer time as only small samples are being used to train. Thus, a larger batch size is preferred that would not decrease the accuracy of the model but also converge faster.



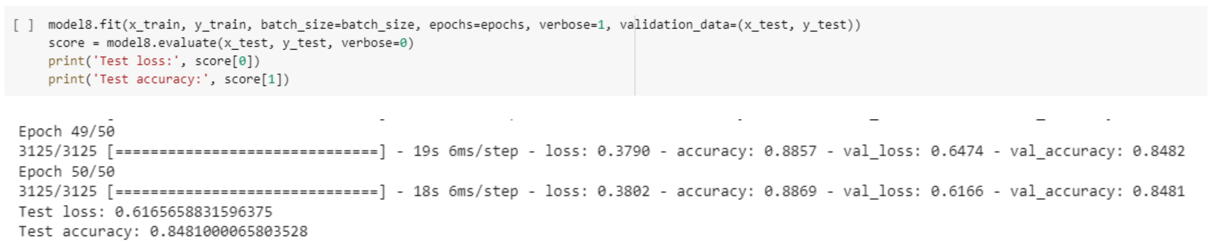
```
[ ]  batch_size = 16
     num_classes = 10
     epochs = 50
```



```
[ ]  model8 = Sequential()
     model8.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same', input_shape = input_shape))
     model8.add(Activation('relu'))
     model8.add(BatchNormalization())
     model8.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same'))
     model8.add(Activation('relu'))
     model8.add(BatchNormalization())
     model8.add(MaxPooling2D(2,2))
     model8.add(Dropout(0.2))

     model8.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
     model8.add(Activation('relu'))
     model8.add(BatchNormalization())
     model8.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
     model8.add(Activation('relu'))
     model8.add(BatchNormalization())
     model8.add(MaxPooling2D(2,2))
     model8.add(Dropout(0.3))

     model8.add(Conv2D(filters = 128, kernel_size = (3,3), padding='same'))
```



```
[ ]  # https://keras.io/optimizers/
     model8.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.01), metrics=['accuracy'])

     /usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
       super(Adam, self).__init__(name, **kwargs)
```



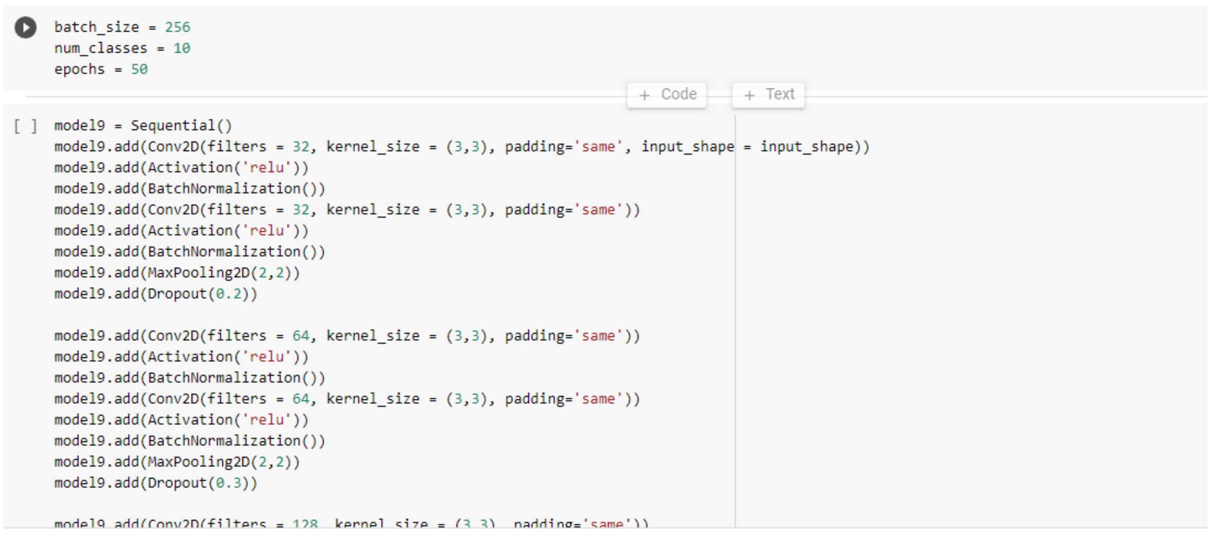
```
[ ]  model8.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, y_test))
     score = model8.evaluate(x_test, y_test, verbose=0)
     print('Test loss:', score[0])
     print('Test accuracy:', score[1])

Epoch 49/50
3125/3125 [==============================] - 19s 6ms/step - loss: 0.3790 - accuracy: 0.8857 - val_loss: 0.6474 - val_accuracy: 0.8482
Epoch 50/50
3125/3125 [==============================] - 18s 6ms/step - loss: 0.3802 - accuracy: 0.8869 - val_loss: 0.6166 - val_accuracy: 0.8481
Test loss: 0.6165658831596375
Test accuracy: 0.8481000065803528
```

Test Loss: 0.616

Test Accuracy: 0.848

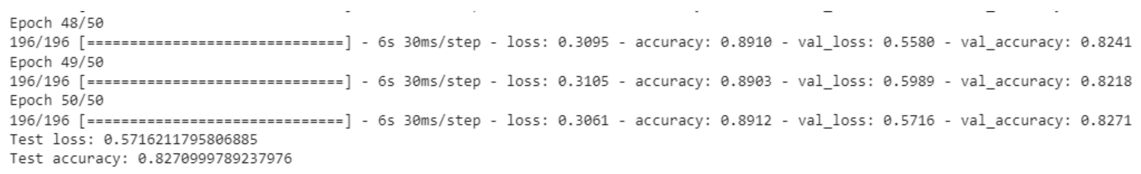
b. Batch Size: 256

If the batch size is too big then while the model is faster to train, it might not be able to capture all the patterns from the input data and thus the accuracy of the model would decrease.



```
batch_size = 256
num_classes = 10
epochs = 50
```

```
model9 = Sequential()
model9.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same', input_shape = input_shape))
model9.add(Activation('relu'))
model9.add(BatchNormalization())
model9.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same'))
model9.add(Activation('relu'))
model9.add(BatchNormalization())
model9.add(MaxPooling2D(2,2))
model9.add(Dropout(0.2))

model9.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
model9.add(Activation('relu'))
model9.add(BatchNormalization())
model9.add(Conv2D(filters = 64, kernel_size = (3,3), padding='same'))
model9.add(Activation('relu'))
model9.add(BatchNormalization())
model9.add(MaxPooling2D(2,2))
model9.add(Dropout(0.3))

model9.add(Conv2D(filters = 128, kernel_size = (3,3), padding='same'))
```



```
Epoch 48/50
196/196 [==============================] - 6s 30ms/step - loss: 0.3095 - accuracy: 0.8910 - val_loss: 0.5580 - val_accuracy: 0.8241
Epoch 49/50
196/196 [==============================] - 6s 30ms/step - loss: 0.3105 - accuracy: 0.8903 - val_loss: 0.5989 - val_accuracy: 0.8218
Epoch 50/50
196/196 [==============================] - 6s 30ms/step - loss: 0.3061 - accuracy: 0.8912 - val_loss: 0.5716 - val_accuracy: 0.8271
Test loss: 0.5716211795806885
Test accuracy: 0.8270999789237976
```

Test Loss: 0.571

Test Accuracy: 0.8270

Conclusion –

Thus, from all the simulations we have ran we find that the learning should not be too high or too low so that the model converges in the specified iterations without affecting the accuracy. Also, the batch size should also be just right so that we are able to capture most of the patterns within the dataset. Batch Normalization is required within the models so that we get better accuracy and also the dropout values should not be to high so that we train the model properly without overfitting to the training data.