Fundamentals of Statistical Learning and Pattern Recognition

CSE 569

Project Report on Deep Learning with CNN

Submitted To

Prof. Dr. Baoxin Li

Submitted By

Sai Vikhyath Kudhroli

ASU Id: 1225432689

Date: 08 December 2022

INTRODUCTION

Problem Statement: To perform hyperparameter tuning in a Convolutional Neural Network used to classify CIFAR-10 dataset. Convolutional Neural Network is used to classify the CIFAR-10 dataset into appropriate classes.

Dataset Description: The dataset used is the CIFAR-10 dataset which is included in the Keras framework's datasets. It consists of 60000 colored images, each of the size 32 x 32 which implies the size of the dataset when read into NumPy array would be of the shape 32 x 32 x 3, where 3 represents the RGB channel of each image. The dataset consists of 10 classes. Out of the 60000 images in the dataset, 50000 are training images and the rest of the 10000 are testing images.

PARAMETER DESCRIPTION

Learning Rate: Learning Rate is a parameter that can be tuned in an optimization algorithm that determines the amount by which the weights of the neural network must be varied with respect to the minimum loss function.

If the learning rate of the algorithm is extremely low, the algorithm takes an extremely long time to converge to the global optimum. Whereas if the learning rate is too big, then the algorithm may overshoot the global optimum and may converge at some other point other than global optimum. So, a learning rate must be carefully defined for the algorithm to converge at global optimum within reasonable time.

Kernel Size: Kernel size is a two-dimensional matrix with each cell containing some value, which is used to perform convolution operation with the image. Kernel size specifies the window size for the convolution operation.

Optimizer: An optimizer is an algorithm that modifies the attributes of the neural network. Typically, an optimizer reduces the overall loss resulting in enhancing the accuracy of the neural network. Optimizer modifies the parameters such as weights to improve the accuracy of the model. Some of the prominent optimizers used are Adam, RMSprop (Root Mean Square propagation), SGD (Stochastic Gradient Descent).

Batch Normalization: Batch Normalization is a technique used to expedite the training process of the network by normalizing the layers' inputs by using some normalization technique. Batch Normalization in general, re-centers and re-scales the data which ensures faster computation and faster training. Some of the popular normalization techniques are Min-Max Normalization, Log Scaling, Decimal Scaling, Feature Clipping, Z-Score.

Dropout: Dropout is a regularization technique that nullifies the contribution of some nodes that have least contribution to learning process. A percent of the nodes that contributes least to the training are cut off from the network by removing the forward and backward edges that pass through those nodes. Dropout is used to avoid overfitting of the model. When the model over fits, it performs well on training data and performs poorly on test data and when some of the least contributing nodes are removed from the network, it generates a more regularized function that fits well to the test data.

Batch Size: Batch size defines the number of training samples that will be propagated through the network. Once the samples of one batch are propagated through the network, the model is updated, hence controlling the accuracy of the error of the gradient when training the neural network.

RESULTS AND OBSERVATIONS

Results with out any modifications: Without any modifications to the network, the model performs well both during training and on test data. During training, it can be observed that the training accuracy, training loss, validation accuracy and validation loss are improving with each epoch. The following are the test accuracy and test loss obtained:

Test Accuracy: 85.49%

Test Loss: 0.5943

```
[38] # https://keras.io/optimizers/
     model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.01), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.05), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.0001), metrics=['accuracy'])
```

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

```
Epoch 23/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3107 - accuracy: 0.8956 - val_loss: 0.5582 - val_accuracy: 0.8502
Epoch 24/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2755 - accuracy: 0.9064 - val_loss: 0.5652 - val_accuracy: 0.8551
Epoch 25/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2674 - accuracy: 0.9108 - val_loss: 0.5550 - val_accuracy: 0.8474
Epoch 26/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2710 - accuracy: 0.9101 - val_loss: 0.5704 - val_accuracy: 0.8491
Epoch 27/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2590 - accuracy: 0.9153 - val_loss: 0.6748 - val_accuracy: 0.8285
Epoch 28/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2665 - accuracy: 0.9117 - val_loss: 0.6163 - val_accuracy: 0.8441
Epoch 29/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2678 - accuracy: 0.9109 - val_loss: 0.6090 - val_accuracy: 0.8533
Epoch 30/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2720 - accuracy: 0.9109 - val_loss: 0.6389 - val_accuracy: 0.8403
Epoch 31/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2677 - accuracy: 0.9112 - val_loss: 0.5691 - val_accuracy: 0.8509
Epoch 32/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2667 - accuracy: 0.9110 - val_loss: 0.5453 - val_accuracy: 0.8544
Epoch 33/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2626 - accuracy: 0.9126 - val_loss: 0.5644 - val_accuracy: 0.8475
Epoch 34/50
782/782 [==============================] - 8s 11ms/step - loss: 0.2675 - accuracy: 0.9116 - val_loss: 0.6220 - val_accuracy: 0.8491
Epoch 35/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2597 - accuracy: 0.9137 - val_loss: 0.6041 - val_accuracy: 0.8494
Epoch 36/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2593 - accuracy: 0.9142 - val_loss: 0.5739 - val_accuracy: 0.8440
Epoch 37/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2567 - accuracy: 0.9141 - val_loss: 0.5735 - val_accuracy: 0.8478
Epoch 38/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2561 - accuracy: 0.9150 - val_loss: 0.5929 - val_accuracy: 0.8506
Epoch 39/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2667 - accuracy: 0.9124 - val_loss: 0.5745 - val_accuracy: 0.8533
Epoch 40/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2545 - accuracy: 0.9158 - val_loss: 0.6850 - val_accuracy: 0.8428
Epoch 41/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2536 - accuracy: 0.9158 - val_loss: 0.6095 - val_accuracy: 0.8521
Epoch 42/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2569 - accuracy: 0.9137 - val_loss: 0.6140 - val_accuracy: 0.8498
Epoch 43/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2496 - accuracy: 0.9174 - val_loss: 0.5964 - val_accuracy: 0.8553
Epoch 44/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2568 - accuracy: 0.9150 - val_loss: 0.5597 - val_accuracy: 0.8488
Epoch 45/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3242 - accuracy: 0.8934 - val_loss: 0.5525 - val_accuracy: 0.8456
Epoch 46/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2659 - accuracy: 0.9110 - val_loss: 0.5679 - val_accuracy: 0.8509
Epoch 47/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2473 - accuracy: 0.9175 - val_loss: 0.5799 - val_accuracy: 0.8515
Epoch 48/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2404 - accuracy: 0.9212 - val_loss: 0.5976 - val_accuracy: 0.8541
Epoch 49/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2458 - accuracy: 0.9197 - val_loss: 0.5881 - val_accuracy: 0.8548
Epoch 50/50
782/782 [==============================] - 8s 10ms/step - loss: 0.2356 - accuracy: 0.9224 - val_loss: 0.5943 - val_accuracy: 0.8549
Test loss: 0.5942748188972473
Test accuracy: 0.8549000024795532
```

Results with 0.05 learning rate: The model performs decently with 0.05 as the learning rate but the accuracy reported is less than that of with 0.01 learning rate. When the learning rate is increased to 0.05, the algorithm takes bigger steps to update weights to reach global optimum, but with high learning rate there is always a possibility that the algorithm may overshoot the global optima which could have happened in this case. The following are results reported:

Test Accuracy: 83.29%

Test Loss: 1.5227

```
[40] # https://keras.io/optimizers/
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.01), metrics=['accuracy'])
     model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.05), metrics=['accuracy'])
     # model.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)

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

     Epoch 23/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.0472 - accuracy: 0.8072 - val_loss: 1.0519 - val_accuracy: 0.7999
     Epoch 24/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1367 - accuracy: 0.8007 - val_loss: 1.9035 - val_accuracy: 0.7385
     Epoch 25/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1848 - accuracy: 0.8048 - val_loss: 1.4002 - val_accuracy: 0.8007
     Epoch 26/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.0403 - accuracy: 0.8149 - val_loss: 1.2354 - val_accuracy: 0.7991
     Epoch 27/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1002 - accuracy: 0.8129 - val_loss: 1.3263 - val_accuracy: 0.7923
     Epoch 28/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1040 - accuracy: 0.8147 - val_loss: 1.2042 - val_accuracy: 0.8044
     Epoch 29/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1924 - accuracy: 0.8123 - val_loss: 1.0428 - val_accuracy: 0.8261
     Epoch 30/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.0132 - accuracy: 0.8255 - val_loss: 1.2017 - val_accuracy: 0.8050
     Epoch 31/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.0579 - accuracy: 0.8244 - val_loss: 1.8948 - val_accuracy: 0.7846
     Epoch 32/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.0885 - accuracy: 0.8232 - val_loss: 1.8146 - val_accuracy: 0.7796
     Epoch 33/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1400 - accuracy: 0.8238 - val_loss: 2.1018 - val_accuracy: 0.7618
     Epoch 34/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.0662 - accuracy: 0.8277 - val_loss: 1.4394 - val_accuracy: 0.7903
     Epoch 35/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1429 - accuracy: 0.8272 - val_loss: 1.4326 - val_accuracy: 0.8102
     Epoch 36/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1227 - accuracy: 0.8296 - val_loss: 1.4159 - val_accuracy: 0.8109
     Epoch 37/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.0883 - accuracy: 0.8351 - val_loss: 1.3297 - val_accuracy: 0.8138
     Epoch 38/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1016 - accuracy: 0.8341 - val_loss: 1.2145 - val_accuracy: 0.8278
     Epoch 39/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1653 - accuracy: 0.8321 - val_loss: 1.3809 - val_accuracy: 0.8074
     Epoch 40/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1084 - accuracy: 0.8366 - val_loss: 1.4029 - val_accuracy: 0.8097
     Epoch 41/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1258 - accuracy: 0.8369 - val_loss: 2.1614 - val_accuracy: 0.7875
     Epoch 42/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.2022 - accuracy: 0.8362 - val_loss: 1.5389 - val_accuracy: 0.7990
     Epoch 43/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.0821 - accuracy: 0.8420 - val_loss: 2.0380 - val_accuracy: 0.7634
     Epoch 44/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1520 - accuracy: 0.8395 - val_loss: 1.6068 - val_accuracy: 0.8026
     Epoch 45/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1772 - accuracy: 0.8431 - val_loss: 1.6018 - val_accuracy: 0.8018
     Epoch 46/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1259 - accuracy: 0.8443 - val_loss: 1.4089 - val_accuracy: 0.8196
     Epoch 47/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1429 - accuracy: 0.8445 - val_loss: 1.4067 - val_accuracy: 0.8342
     Epoch 48/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1894 - accuracy: 0.8461 - val_loss: 1.5194 - val_accuracy: 0.8177
     Epoch 49/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.2173 - accuracy: 0.8445 - val_loss: 1.5041 - val_accuracy: 0.8278
     Epoch 50/50
     782/782 [==============================] - 8s 10ms/step - loss: 1.1817 - accuracy: 0.8483 - val_loss: 1.5228 - val_accuracy: 0.8329
     Test loss: 1.5227998495101929
     Test accuracy: 0.8328999876976013
```

Results with 0.0001 learning rate: The model performs better when the learning rate is set to 0.0001 than when set to 0.01 and 0.05. When the learning rate is set to a lower value, the algorithm takes smaller steps in updating weights to reach the global optima. Thus, reducing the learning rate to 0.0001 may have reduced the step size of reaching global optima and may have appropriately reached global optima. However, setting the learning rate to a low value may not always be a good approach because the algorithm would take a lot of time to converge to global optimum. The following are the results obtained:

Test Accuracy: 86.74%

Test Loss: 0.6290

```
[42] # https://keras.io/optimizers/
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.01), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.05), metrics=['accuracy'])
     model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.0001), metrics=['accuracy'])

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

     Epoch 23/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3760 - accuracy: 0.9108 - val_loss: 0.8262 - val_accuracy: 0.8645
     Epoch 24/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3991 - accuracy: 0.9135 - val_loss: 0.8179 - val_accuracy: 0.8654
     Epoch 25/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3692 - accuracy: 0.9131 - val_loss: 0.8050 - val_accuracy: 0.8649
     Epoch 26/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3653 - accuracy: 0.9117 - val_loss: 0.7998 - val_accuracy: 0.8658
     Epoch 27/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3568 - accuracy: 0.9130 - val_loss: 0.7893 - val_accuracy: 0.8650
     Epoch 28/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3519 - accuracy: 0.9134 - val_loss: 0.7793 - val_accuracy: 0.8655
     Epoch 29/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3498 - accuracy: 0.9119 - val_loss: 0.7680 - val_accuracy: 0.8664
     Epoch 30/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3489 - accuracy: 0.9125 - val_loss: 0.7597 - val_accuracy: 0.8652
     Epoch 31/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3409 - accuracy: 0.9140 - val_loss: 0.7549 - val_accuracy: 0.8655
     Epoch 32/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3361 - accuracy: 0.9127 - val_loss: 0.7440 - val_accuracy: 0.8658
     Epoch 33/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3397 - accuracy: 0.9122 - val_loss: 0.7369 - val_accuracy: 0.8662
     Epoch 34/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3224 - accuracy: 0.9168 - val_loss: 0.7317 - val_accuracy: 0.8667
     Epoch 35/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3202 - accuracy: 0.9164 - val_loss: 0.7227 - val_accuracy: 0.8676
     Epoch 36/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3487 - accuracy: 0.9151 - val_loss: 0.7170 - val_accuracy: 0.8662
     Epoch 37/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3193 - accuracy: 0.9154 - val_loss: 0.7101 - val_accuracy: 0.8671
     Epoch 38/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3215 - accuracy: 0.9139 - val_loss: 0.7027 - val_accuracy: 0.8655
     Epoch 39/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3062 - accuracy: 0.9167 - val_loss: 0.6951 - val_accuracy: 0.8666
     Epoch 40/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3388 - accuracy: 0.9159 - val_loss: 0.6925 - val_accuracy: 0.8662
     Epoch 41/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.2990 - accuracy: 0.9178 - val_loss: 0.6863 - val_accuracy: 0.8669
     Epoch 42/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3027 - accuracy: 0.9159 - val_loss: 0.6783 - val_accuracy: 0.8668
     Epoch 43/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3011 - accuracy: 0.9167 - val_loss: 0.6715 - val_accuracy: 0.8673
     Epoch 44/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3078 - accuracy: 0.9154 - val_loss: 0.6653 - val_accuracy: 0.8671
     Epoch 45/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.2916 - accuracy: 0.9172 - val_loss: 0.6579 - val_accuracy: 0.8667
     Epoch 46/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.2938 - accuracy: 0.9171 - val_loss: 0.6506 - val_accuracy: 0.8669
     Epoch 47/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.2868 - accuracy: 0.9176 - val_loss: 0.6463 - val_accuracy: 0.8675
     Epoch 48/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.2896 - accuracy: 0.9175 - val_loss: 0.6433 - val_accuracy: 0.8667
     Epoch 49/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.2906 - accuracy: 0.9166 - val_loss: 0.6356 - val_accuracy: 0.8669
     Epoch 50/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3185 - accuracy: 0.9155 - val_loss: 0.6291 - val_accuracy: 0.8674
     Test loss: 0.6290989518165588
     Test accuracy: 0.8673999905586243
```

Results with 7x7 kernel size: Model's accuracy dropped when a kernel of 7x7 size was used. Increasing the kernel size increases the window of the convolution operation. Increasing the kernel size can increase the complexity of the model, however, with a large kernel size, more area of the image is convolved at once leading to important and unimportant features of the image to be convolved at the same time which can dampen the efficiency of the model. With a lower kernel size, small area of the image is convolved leading to important features to be convolved to a single cell in next layer. The following results were obtained:

Test Accuracy: 80.54%

Test Loss: 0.6462

```
[44] model = Sequential()
     # model.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same', input_shape = input_shape))
     model.add(Conv2D(filters = 32, kernel_size = (7,7), 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))
     model.add(Dropout(0.4))

     model.add(Flatten())

     model.add(Dense(units = 10, activation = 'softmax'))
```

```
[45] # https://keras.io/optimizers/
     model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.01), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.05), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.0001), metrics=['accuracy'])
```

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

```
Epoch 23/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4688 - accuracy: 0.8384 - val_loss: 0.5886 - val_accuracy: 0.8127
Epoch 24/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4626 - accuracy: 0.8407 - val_loss: 0.6294 - val_accuracy: 0.8012
Epoch 25/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4539 - accuracy: 0.8446 - val_loss: 0.6098 - val_accuracy: 0.8137
Epoch 26/50
782/782 [==============================] - 8s 11ms/step - loss: 0.4436 - accuracy: 0.8476 - val_loss: 0.6084 - val_accuracy: 0.8186
Epoch 27/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4361 - accuracy: 0.8516 - val_loss: 0.6394 - val_accuracy: 0.7948
Epoch 28/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4309 - accuracy: 0.8544 - val_loss: 0.5882 - val_accuracy: 0.8177
Epoch 29/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4299 - accuracy: 0.8523 - val_loss: 0.6413 - val_accuracy: 0.8094
Epoch 30/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4184 - accuracy: 0.8557 - val_loss: 0.6215 - val_accuracy: 0.8114
Epoch 31/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4141 - accuracy: 0.8589 - val_loss: 0.5558 - val_accuracy: 0.8253
Epoch 32/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4074 - accuracy: 0.8602 - val_loss: 0.5924 - val_accuracy: 0.8217
Epoch 33/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3980 - accuracy: 0.8632 - val_loss: 0.5415 - val_accuracy: 0.8335
Epoch 34/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3985 - accuracy: 0.8644 - val_loss: 0.5865 - val_accuracy: 0.8241
Epoch 35/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3909 - accuracy: 0.8663 - val_loss: 0.5601 - val_accuracy: 0.8245
Epoch 36/50
782/782 [==============================] - 8s 11ms/step - loss: 0.3848 - accuracy: 0.8685 - val_loss: 0.6056 - val_accuracy: 0.8096
Epoch 37/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3823 - accuracy: 0.8689 - val_loss: 0.5648 - val_accuracy: 0.8283
Epoch 38/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3742 - accuracy: 0.8706 - val_loss: 0.5589 - val_accuracy: 0.8279
Epoch 39/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3723 - accuracy: 0.8736 - val_loss: 0.6134 - val_accuracy: 0.8242
Epoch 40/50
782/782 [==============================] - 9s 11ms/step - loss: 0.3708 - accuracy: 0.8731 - val_loss: 0.5570 - val_accuracy: 0.8317
Epoch 41/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3657 - accuracy: 0.8754 - val_loss: 0.5686 - val_accuracy: 0.8293
Epoch 42/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3584 - accuracy: 0.8772 - val_loss: 0.5594 - val_accuracy: 0.8351
Epoch 43/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3562 - accuracy: 0.8782 - val_loss: 0.6078 - val_accuracy: 0.8270
Epoch 44/50
782/782 [==============================] - 9s 12ms/step - loss: 0.3553 - accuracy: 0.8784 - val_loss: 0.6077 - val_accuracy: 0.8321
Epoch 45/50
782/782 [==============================] - 8s 11ms/step - loss: 0.3488 - accuracy: 0.8801 - val_loss: 0.6296 - val_accuracy: 0.8157
Epoch 46/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3483 - accuracy: 0.8817 - val_loss: 0.5808 - val_accuracy: 0.8264
Epoch 47/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3463 - accuracy: 0.8807 - val_loss: 0.5798 - val_accuracy: 0.8268
Epoch 48/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3450 - accuracy: 0.8814 - val_loss: 0.5802 - val_accuracy: 0.8280
Epoch 49/50
782/782 [==============================] - 8s 10ms/step - loss: 0.3353 - accuracy: 0.8848 - val_loss: 0.6019 - val_accuracy: 0.8324
Epoch 50/50
782/782 [==============================] - 8s 11ms/step - loss: 0.3463 - accuracy: 0.8839 - val_loss: 0.6462 - val_accuracy: 0.8054
Test loss: 0.646163284778595
Test accuracy: 0.805400013923645
```

Results with RMSprop optimizer: The results obtained with Root Mean Square propagation optimizer are close to the results obtained with Adam's optimizer with a minimal loss of accuracy. Both Adam and RMSprop are adaptive optimizers. However, RMSprop uses a decaying average of partial gradients to update the parameters, which simply is a way to update weights by considering some history of the gradient descent. While Adam adapts the parameters based on the mean and the variance of the past few gradients. So, Adam in most cases performs better than other optimizers. The following results were obtained:

Test Accuracy: 84.62%

Test Loss: 0.5240

```
[47] model = Sequential()
     model.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same', input_shape = input_shape))
     # model.add(Conv2D(filters = 32, kernel_size = (7,7), 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))
     model.add(Dropout(0.4))

     model.add(Flatten())

     model.add(Dense(units = 10, activation = 'softmax'))

[48] # https://keras.io/optimizers/
     model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.RMSprop(lr=0.01), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.05), metrics=['accuracy'])
     # model.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/rmsprop.py:135: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
       super(RMSprop, self).__init__(name, **kwargs)

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

     Epoch 23/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.4379 - accuracy: 0.8499 - val_loss: 0.6437 - val_accuracy: 0.8065
     Epoch 24/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.4274 - accuracy: 0.8531 - val_loss: 0.6363 - val_accuracy: 0.8005
     Epoch 25/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.4187 - accuracy: 0.8583 - val_loss: 0.7447 - val_accuracy: 0.7789
     Epoch 26/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.4165 - accuracy: 0.8570 - val_loss: 0.5743 - val_accuracy: 0.8151
     Epoch 27/50
     782/782 [==============================] - 9s 12ms/step - loss: 0.4101 - accuracy: 0.8595 - val_loss: 0.5500 - val_accuracy: 0.8189
     Epoch 28/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.4043 - accuracy: 0.8628 - val_loss: 0.6114 - val_accuracy: 0.8092
     Epoch 29/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3969 - accuracy: 0.8640 - val_loss: 0.6804 - val_accuracy: 0.7969
     Epoch 30/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3974 - accuracy: 0.8654 - val_loss: 0.6897 - val_accuracy: 0.8021
     Epoch 31/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3858 - accuracy: 0.8682 - val_loss: 0.4953 - val_accuracy: 0.8433
     Epoch 32/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3860 - accuracy: 0.8668 - val_loss: 0.5910 - val_accuracy: 0.8135
     Epoch 33/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3753 - accuracy: 0.8710 - val_loss: 0.7235 - val_accuracy: 0.7972
     Epoch 34/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3771 - accuracy: 0.8707 - val_loss: 0.6698 - val_accuracy: 0.7925
     Epoch 35/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3652 - accuracy: 0.8748 - val_loss: 0.6250 - val_accuracy: 0.8114
     Epoch 36/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3602 - accuracy: 0.8778 - val_loss: 0.6758 - val_accuracy: 0.7970
     Epoch 37/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3562 - accuracy: 0.8791 - val_loss: 0.6457 - val_accuracy: 0.8132
     Epoch 38/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3532 - accuracy: 0.8798 - val_loss: 0.5811 - val_accuracy: 0.8246
     Epoch 39/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3525 - accuracy: 0.8791 - val_loss: 0.7227 - val_accuracy: 0.8083
     Epoch 40/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3518 - accuracy: 0.8786 - val_loss: 0.5805 - val_accuracy: 0.8330
     Epoch 41/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3474 - accuracy: 0.8814 - val_loss: 0.5285 - val_accuracy: 0.8377
     Epoch 42/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3412 - accuracy: 0.8840 - val_loss: 0.7034 - val_accuracy: 0.8036
     Epoch 43/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3365 - accuracy: 0.8856 - val_loss: 0.5493 - val_accuracy: 0.8334
     Epoch 44/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3312 - accuracy: 0.8879 - val_loss: 0.5573 - val_accuracy: 0.8371
     Epoch 45/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3337 - accuracy: 0.8872 - val_loss: 0.6696 - val_accuracy: 0.8059
     Epoch 46/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3323 - accuracy: 0.8869 - val_loss: 0.5910 - val_accuracy: 0.8327
     Epoch 47/50
     782/782 [==============================] - 8s 11ms/step - loss: 0.3241 - accuracy: 0.8904 - val_loss: 0.6448 - val_accuracy: 0.8183
     Epoch 48/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3214 - accuracy: 0.8910 - val_loss: 0.5392 - val_accuracy: 0.8389
     Epoch 49/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3182 - accuracy: 0.8911 - val_loss: 0.7017 - val_accuracy: 0.7934
     Epoch 50/50
     782/782 [==============================] - 8s 10ms/step - loss: 0.3198 - accuracy: 0.8902 - val_loss: 0.5240 - val_accuracy: 0.8462
     Test loss: 0.5240421891212463
     Test accuracy: 0.8461999893188477
```

Results with batch normalization removed: The results obtained after removing batch normalization are extremely poor. Removing batch normalization will not normalize the inputs at each layer resulting in the inputs being of very large values and would require an extremely low learning rate to converge to global optimum. Because the inputs are not normalized and the learning rate not modified, the model performs poorly. The following results are obtained:

Test Accuracy: 10%

Test Loss: 2.3034

```
[50] model = Sequential()
     model.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same', input_shape = input_shape))
     # model.add(Conv2D(filters = 32, kernel_size = (7,7), 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))
     model.add(Dropout(0.4))

     model.add(Flatten())

     model.add(Dense(units = 10, activation = 'softmax'))

[51] # https://keras.io/optimizers/
     model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.01), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.RMSprop(lr=0.01), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.05), metrics=['accuracy'])
     # model.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)

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

     Epoch 23/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3036 - accuracy: 0.0978 - val_loss: 2.3032 - val_accuracy: 0.1000
     Epoch 24/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3035 - accuracy: 0.0988 - val_loss: 2.3031 - val_accuracy: 0.1000
     Epoch 25/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3037 - accuracy: 0.0979 - val_loss: 2.3033 - val_accuracy: 0.1000
     Epoch 26/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3036 - accuracy: 0.0981 - val_loss: 2.3035 - val_accuracy: 0.1000
     Epoch 27/50
     782/782 [==============================] - 7s 9ms/step - loss: 2.3037 - accuracy: 0.0977 - val_loss: 2.3033 - val_accuracy: 0.1000
     Epoch 28/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3036 - accuracy: 0.1023 - val_loss: 2.3032 - val_accuracy: 0.1000
     Epoch 29/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3037 - accuracy: 0.0981 - val_loss: 2.3032 - val_accuracy: 0.1000
     Epoch 30/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3038 - accuracy: 0.0976 - val_loss: 2.3037 - val_accuracy: 0.1000
     Epoch 31/50
     782/782 [==============================] - 7s 9ms/step - loss: 2.3037 - accuracy: 0.0973 - val_loss: 2.3038 - val_accuracy: 0.1000
     Epoch 32/50
     782/782 [==============================] - 7s 9ms/step - loss: 2.3035 - accuracy: 0.1017 - val_loss: 2.3032 - val_accuracy: 0.1000
     Epoch 33/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3036 - accuracy: 0.0980 - val_loss: 2.3034 - val_accuracy: 0.1000
     Epoch 34/50
     782/782 [==============================] - 6s 8ms/step - loss: 2.3036 - accuracy: 0.1003 - val_loss: 2.3034 - val_accuracy: 0.1000
     Epoch 35/50
     782/782 [==============================] - 6s 8ms/step - loss: 2.3035 - accuracy: 0.1015 - val_loss: 2.3038 - val_accuracy: 0.1000
     Epoch 36/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3035 - accuracy: 0.1007 - val_loss: 2.3033 - val_accuracy: 0.1000
     Epoch 37/50
     782/782 [==============================] - 7s 9ms/step - loss: 2.3036 - accuracy: 0.0980 - val_loss: 2.3037 - val_accuracy: 0.1000
     Epoch 38/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3037 - accuracy: 0.1004 - val_loss: 2.3031 - val_accuracy: 0.1000
     Epoch 39/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3036 - accuracy: 0.0990 - val_loss: 2.3035 - val_accuracy: 0.1000
     Epoch 40/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3035 - accuracy: 0.1022 - val_loss: 2.3033 - val_accuracy: 0.1000
     Epoch 41/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3038 - accuracy: 0.0981 - val_loss: 2.3029 - val_accuracy: 0.1000
     Epoch 42/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3035 - accuracy: 0.0997 - val_loss: 2.3038 - val_accuracy: 0.1000
     Epoch 43/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3036 - accuracy: 0.1004 - val_loss: 2.3030 - val_accuracy: 0.1000
     Epoch 44/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3035 - accuracy: 0.1005 - val_loss: 2.3033 - val_accuracy: 0.1000
     Epoch 45/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3035 - accuracy: 0.1000 - val_loss: 2.3039 - val_accuracy: 0.1000
     Epoch 46/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3036 - accuracy: 0.1007 - val_loss: 2.3033 - val_accuracy: 0.1000
     Epoch 47/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3034 - accuracy: 0.1005 - val_loss: 2.3042 - val_accuracy: 0.1000
     Epoch 48/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3038 - accuracy: 0.0991 - val_loss: 2.3031 - val_accuracy: 0.1000
     Epoch 49/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3037 - accuracy: 0.0985 - val_loss: 2.3037 - val_accuracy: 0.1000
     Epoch 50/50
     782/782 [==============================] - 7s 8ms/step - loss: 2.3036 - accuracy: 0.1019 - val_loss: 2.3034 - val_accuracy: 0.1000
     Test loss: 2.3034255504608154
     Test accuracy: 0.10000000149011612
```

Results with 0.7 dropout: The results obtained with 0.7 dropout are not as good as the one without any modifications. Dropout is used for regularization. When the model performs well on training data and does not perform well on test data, dropout is used to regularize by dropping out least contributing nodes from the network. But the percent of nodes dropped has an impact on the performance of the model. When more nodes are dropped, the network becomes sparse and may perform poorly.

Test Accuracy: 74.48%

Test Loss: 0.7895

```
[53] model = Sequential()
     model.add(Conv2D(filters = 32, kernel_size = (3,3), padding='same', input_shape = input_shape))
     # model.add(Conv2D(filters = 32, kernel_size = (7,7), 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(Dropout(0.7))

     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(Dropout(0.7))

     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))
     # model.add(Dropout(0.4))
     model.add(Dropout(0.7))

     model.add(Flatten())

     model.add(Dense(units = 10, activation = 'softmax'))
```

```
[54] # https://keras.io/optimizers/
     model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.01), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.RMSprop(lr=0.01), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.05), metrics=['accuracy'])
     # model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.0001), metrics=['accuracy'])
```

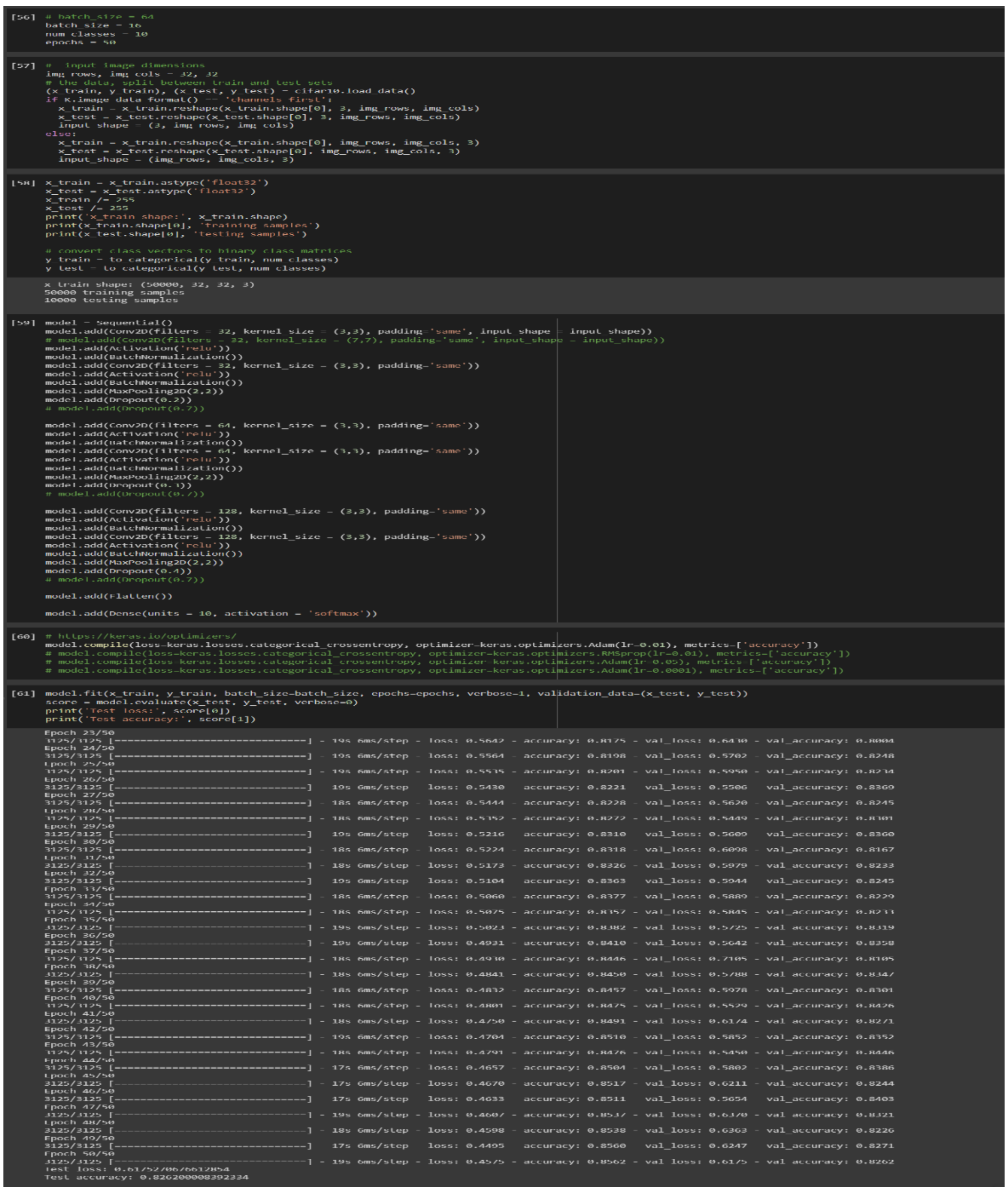
```
[55] model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, y_test))
     score = model.evaluate(x_test, y_test, verbose=0)
     print('Test loss:', score[0])
     print('Test accuracy:', score[1])
```

```
Epoch 23/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8887 - accuracy: 0.6954 - val_loss: 0.7781 - val_accuracy: 0.7384
Epoch 24/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8729 - accuracy: 0.6982 - val_loss: 0.7682 - val_accuracy: 0.7435
Epoch 25/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8705 - accuracy: 0.6987 - val_loss: 0.9734 - val_accuracy: 0.6902
Epoch 26/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8589 - accuracy: 0.7034 - val_loss: 0.6708 - val_accuracy: 0.7707
Epoch 27/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8607 - accuracy: 0.7029 - val_loss: 0.6667 - val_accuracy: 0.7718
Epoch 28/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8566 - accuracy: 0.7042 - val_loss: 0.9124 - val_accuracy: 0.6964
Epoch 29/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8437 - accuracy: 0.7090 - val_loss: 0.8672 - val_accuracy: 0.7086
Epoch 30/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8392 - accuracy: 0.7103 - val_loss: 0.9366 - val_accuracy: 0.7019
Epoch 31/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8424 - accuracy: 0.7087 - val_loss: 1.0572 - val_accuracy: 0.6695
Epoch 32/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8339 - accuracy: 0.7139 - val_loss: 0.6269 - val_accuracy: 0.7774
Epoch 33/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8249 - accuracy: 0.7154 - val_loss: 0.7559 - val_accuracy: 0.7421
Epoch 34/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8212 - accuracy: 0.7170 - val_loss: 0.7322 - val_accuracy: 0.7652
Epoch 35/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8203 - accuracy: 0.7159 - val_loss: 0.6736 - val_accuracy: 0.7731
Epoch 36/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8122 - accuracy: 0.7168 - val_loss: 0.7035 - val_accuracy: 0.7571
Epoch 37/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8119 - accuracy: 0.7190 - val_loss: 0.7681 - val_accuracy: 0.7362
Epoch 38/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8110 - accuracy: 0.7208 - val_loss: 0.6677 - val_accuracy: 0.7793
Epoch 39/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8067 - accuracy: 0.7226 - val_loss: 0.7138 - val_accuracy: 0.7646
Epoch 40/50
782/782 [==============================] - 8s 10ms/step - loss: 0.7955 - accuracy: 0.7246 - val_loss: 0.6804 - val_accuracy: 0.7664
Epoch 41/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8003 - accuracy: 0.7242 - val_loss: 0.7772 - val_accuracy: 0.7469
Epoch 42/50
782/782 [==============================] - 8s 10ms/step - loss: 0.8035 - accuracy: 0.7248 - val_loss: 0.6394 - val_accuracy: 0.7824
Epoch 43/50
782/782 [==============================] - 8s 10ms/step - loss: 0.7888 - accuracy: 0.7269 - val_loss: 1.0327 - val_accuracy: 0.6610
Epoch 44/50
782/782 [==============================] - 8s 10ms/step - loss: 0.7910 - accuracy: 0.7277 - val_loss: 0.6738 - val_accuracy: 0.7769
Epoch 45/50
782/782 [==============================] - 8s 10ms/step - loss: 0.7865 - accuracy: 0.7287 - val_loss: 0.6369 - val_accuracy: 0.7837
Epoch 46/50
782/782 [==============================] - 8s 11ms/step - loss: 0.7888 - accuracy: 0.7300 - val_loss: 0.7020 - val_accuracy: 0.7663
Epoch 47/50
782/782 [==============================] - 8s 10ms/step - loss: 0.7801 - accuracy: 0.7324 - val_loss: 0.9720 - val_accuracy: 0.6883
Epoch 48/50
782/782 [==============================] - 8s 10ms/step - loss: 0.7782 - accuracy: 0.7323 - val_loss: 0.6557 - val_accuracy: 0.7800
Epoch 49/50
782/782 [==============================] - 8s 10ms/step - loss: 0.7811 - accuracy: 0.7323 - val_loss: 0.6618 - val_accuracy: 0.7726
Epoch 50/50
782/782 [==============================] - 8s 10ms/step - loss: 0.7736 - accuracy: 0.7347 - val_loss: 0.7895 - val_accuracy: 0.7448
Test loss: 0.7894716262817383
Test accuracy: 0.7447999715805054
```

Results with 16 batch size: Batch size has an impact on the accuracy of the model by controlling the estimate of error in gradient descent. Lower batch size implies the weights will be updated several times in each epoch. A smaller batch size with a low learning rate works better on most of the scenarios. The results obtained with the batch size of 16 were as follows:

Test Accuracy: 82.62%

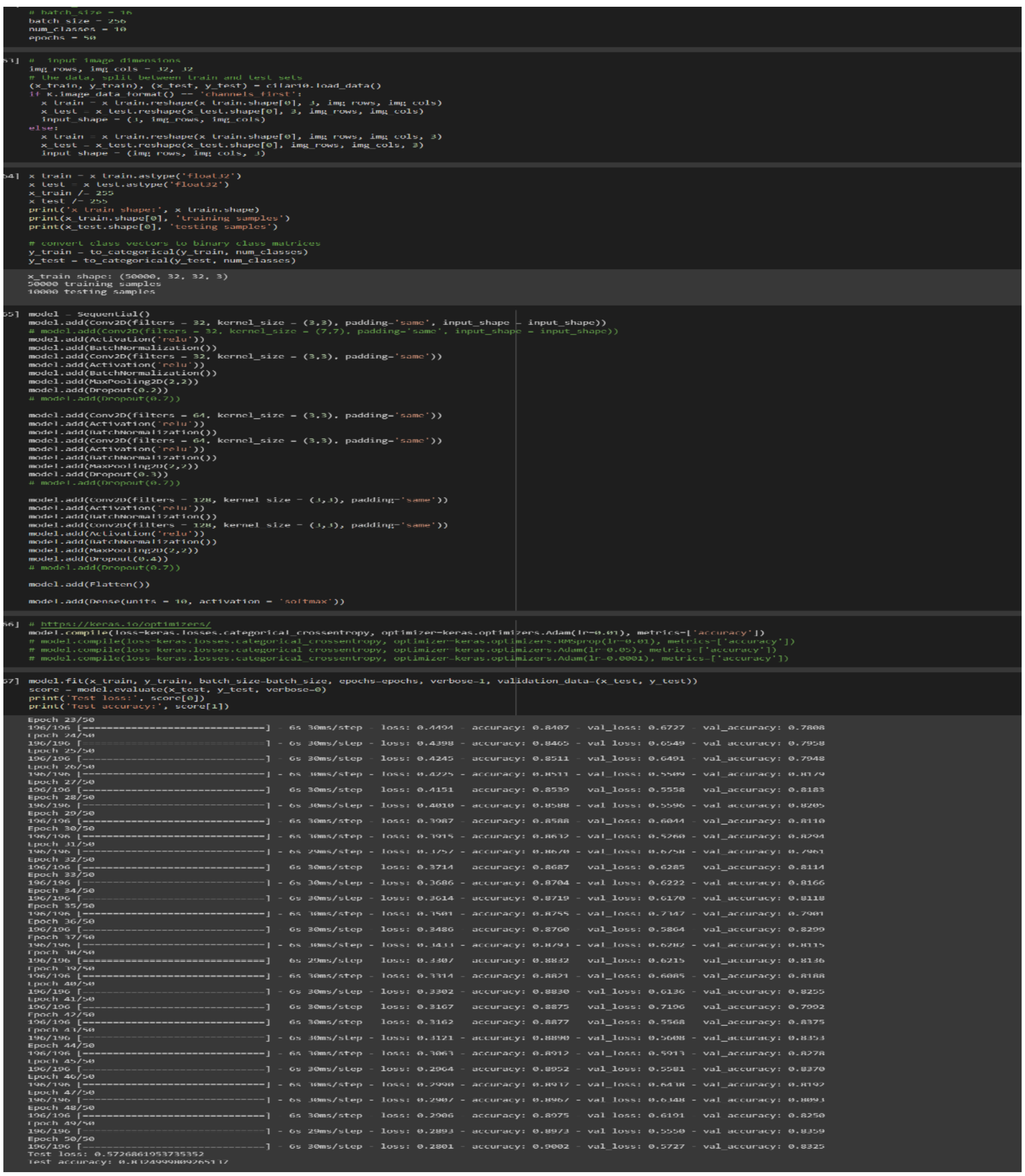
Test Loss: 0.6175



Results with 256 batch size: Batch size has an impact on the accuracy of the model by controlling the estimate of error in gradient descent. Higher batch size implies the weights will be updated lesser number of times in each epoch. A larger batch size with a high learning rate generally works together. The results obtained with the batch size of 256 were as follows:

Test Accuracy: 83.25%

Test Loss: 0.5727



CONCLUSION

This project was aimed at learning about Convolutional Neural Network and Hyperparameter tuning of the neural network to enhance accuracy. Key learning from the project was on how to build a convolutional neural network to perform image classification using Keras framework. It was observed on how the model's accuracy varies with various parameters. It was also learnt what parameters to be used in a particular use case and how to tune those parameters to achieve better results.