CS577 Assignment 1: Sample Report

Shivi Srivastava
Department Of Computer Science
Illinois Institute of Technology
March 25, 2021

Problem 2 –

1. Problem statement

This problem is for CIFAR10 where we need to get the subset of it as three classes and prepare a model which can identify the images within these three classes by using the training validation and testing set.

We need to try out different optimizers, regularization methods, loss functions and then draw the conclusion based on the observation.

2. Proposed solution

- data in cypher 10 is divided into 10 classes each class having 5000 training images and 1000 testing images so for our solution we will have total 15,000 training set plus validation set and 3000 testing images.
- We will start by downloading the data from the source and preprocessing it, making it in a format which can then easily be fed to the neural network.
- We will use combination of different optimizers, loss functions and regularization techniques as we tune the model.
- As this is the fully connected neural network using only the dense layers.
- As this is multiclass classification problem we are going to use below loss functions-
  Multi-Class Cross-Entropy Loss
  Sparse Multiclass Cross-Entropy Loss
  Kullback Leibler Divergence Loss

3. Implementation details

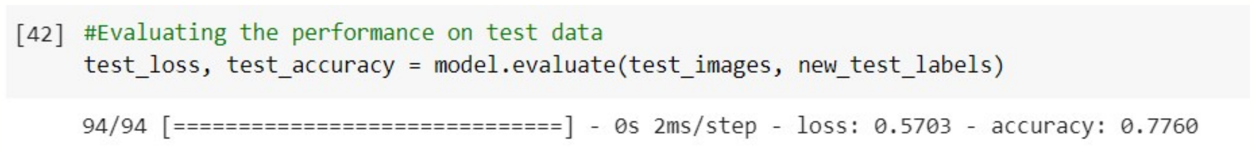
Model 1 -

```
opt=keras.optimizers.Adagrad(
    learning_rate=0.007,
    #initial_accumulator_value=,
    epsilon=1e-07,
    name="Adagrad"

)
model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accu
racy'])
```

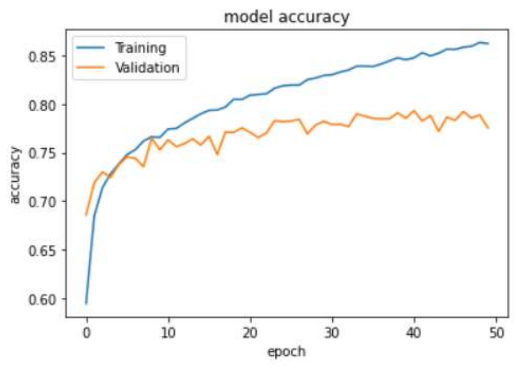
```
model= Sequential()
model.add(Dense(units=50,activation='relu',input_shape=(32*32*3,)))
#model.add(Dropout(0.6))
model.add(Dense(units=20))
#model.add(Dropout(0.5))
#model.add(Dense(units=100))
#model.add(Dense(20))
model.add(Dense(units=10,activation='softmax'))
model.summary()
```

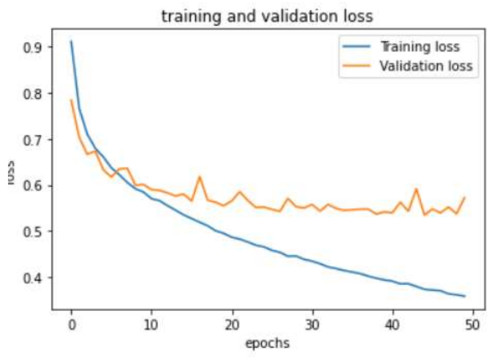
<u>**Outcome-**</u>

Looking at the below graph and evaluation performance we can see there is some amount of overfitting with the above configuration.



```
[42] #Evaluating the performance on test data
     test_loss, test_accuracy = model.evaluate(test_images, new_test_labels)

     94/94 [==============================] - 0s 2ms/step - loss: 0.5703 - accuracy: 0.7760
```





<u>Model 2 –</u>

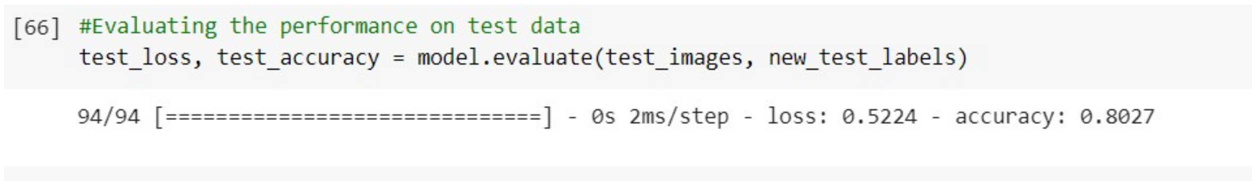
<u>Adding a dropout layer on 2nd hidden layer-</u>

```
model= Sequential()
model.add(Dense(units=50,activation='relu',input_shape=(32*32*3,)))
model.add(Dense(units=20))
model.add(Dropout(0.5))
model.add(Dense(units=10,activation='softmax'))
model.summary()
```

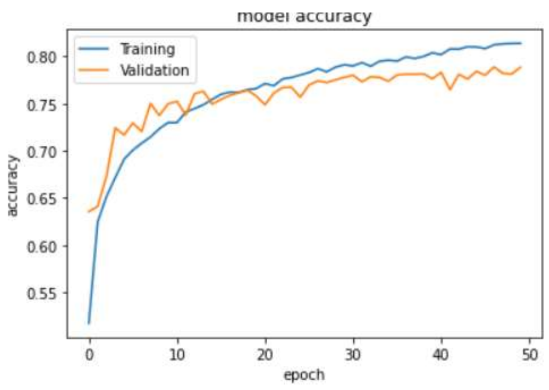
<u>Outcome-</u>

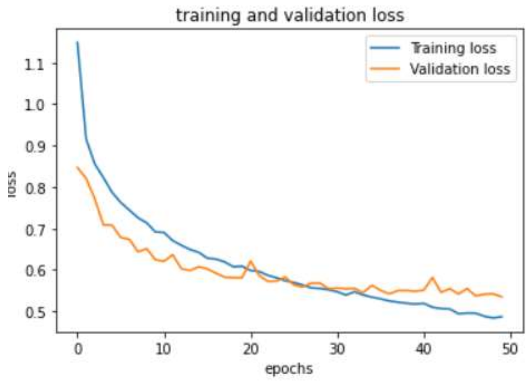
We can observe that the accuracy increased and overfitting reduced to a certain amount.

Increasing the learning rate from 0.007 → 0.01 at this point is increasing the overfitting.



```
[66] #Evaluating the performance on test data
     test_loss, test_accuracy = model.evaluate(test_images, new_test_labels)

     94/94 [==============================] - 0s 2ms/step - loss: 0.5224 - accuracy: 0.8027
```





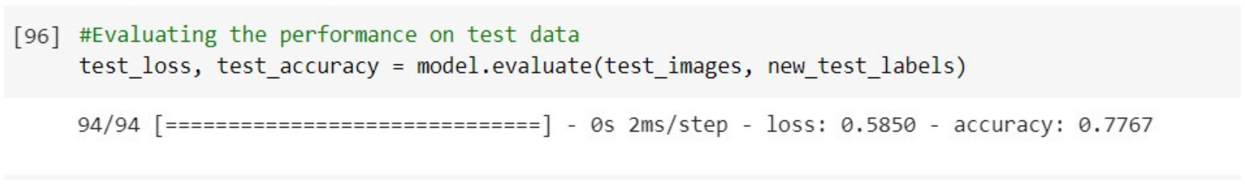
Model 3 –

Adding a dropout layer on 1nd input layer-

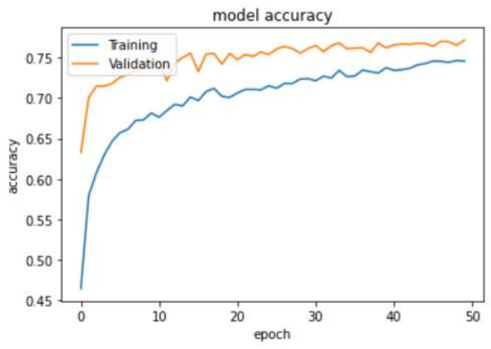
```
model= Sequential()
model.add(Dense(units=50,activation='relu',input_shape=(32*32*3,)))
model.add(Dropout(0.5))
model.add(Dense(units=20))
#model.add(Dropout(0.5))
#model.add(Dense(units=100))
#model.add(Dense(20))
model.add(Dense(units=10,activation='softmax'))
model.summary()
```

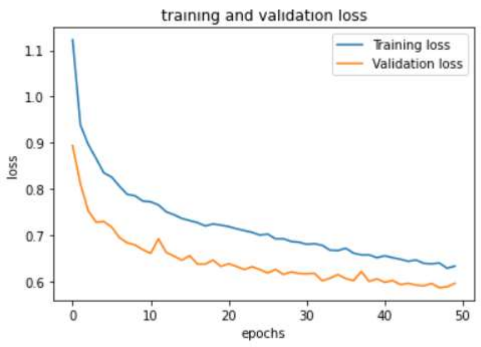
Outcome-

Though the accuracy is not that great, overfitting is reduced significantly.



```
[96] #Evaluating the performance on test data
     test_loss, test_accuracy = model.evaluate(test_images, new_test_labels)

     94/94 [==============================] - 0s 2ms/step - loss: 0.5850 - accuracy: 0.7767
```





Model 4-

Adding L1 and L2 Regularizer (weight Decay)

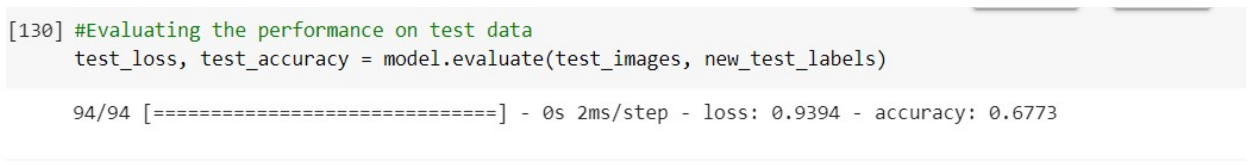
For L1 Regularizer-

```
model= Sequential()
model.add(Dense(units=50,activation='relu',input_shape=(32*32*3,), kernel_
regularizer=l1(0.01), bias_regularizer=l1(0.01)))
#model.add(Dropout(0.5))
model.add(Dense(units=20))
#model.add(Dropout(0.5))
#model.add(Dense(units=100))
#model.add(Dense(20))
model.add(Dense(units=10,activation='softmax'))
model.summary()
```

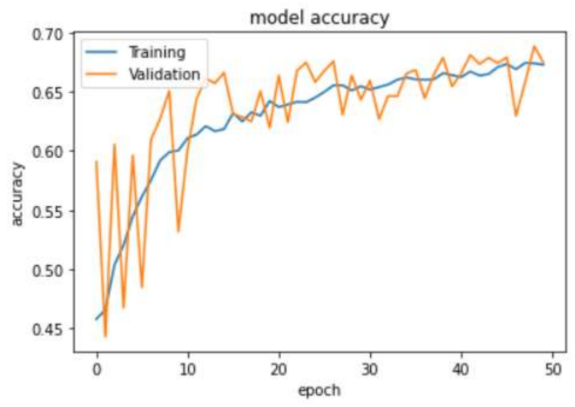
Outcome-

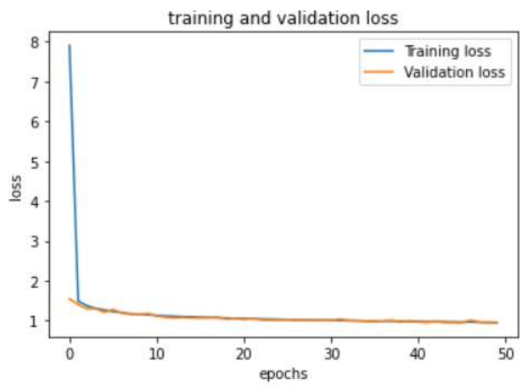
The accuracy graph measure spiked a lot and the training los and validation loss didn't have very good start with training loss starting from 16.5% for the first epoch.

Total all loss does not look good for testing set.



```
[130] #Evaluating the performance on test data
      test_loss, test_accuracy = model.evaluate(test_images, new_test_labels)

      94/94 [==============================] - 0s 2ms/step - loss: 0.9394 - accuracy: 0.6773
```



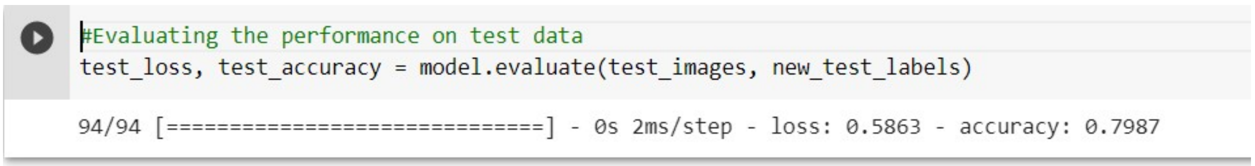


For L2 Regularizer-

```
model= Sequential()
model.add(Dense(units=50,activation='relu',input_shape=(32*32*3,), kernel_
regularizer=l2(0.01), bias_regularizer=l2(0.01)))
#model.add(Dropout(0.5))
model.add(Dense(units=20))
#model.add(Dropout(0.5))
#model.add(Dense(units=100))
#model.add(Dense(20))
model.add(Dense(units=10,activation='softmax'))
model.summary()
```

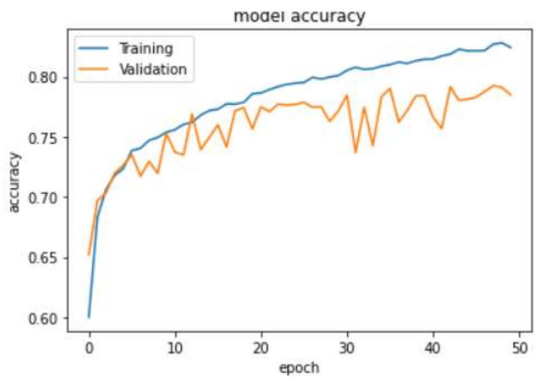
Outcome-

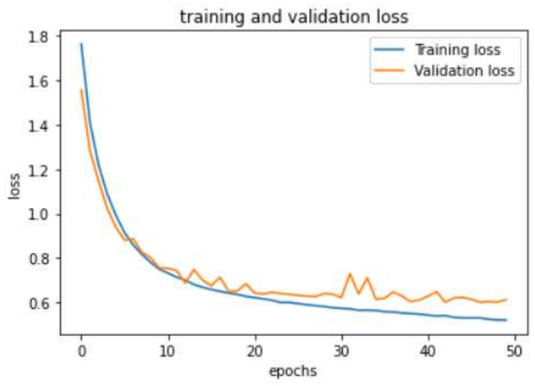
The graph is comparitively smoother than for L1 regularizer and acuuracy is good though there are still fluctuations when compared to dropouts.



```
#Evaluating the performance on test data
test_loss, test_accuracy = model.evaluate(test_images, new_test_labels)
```

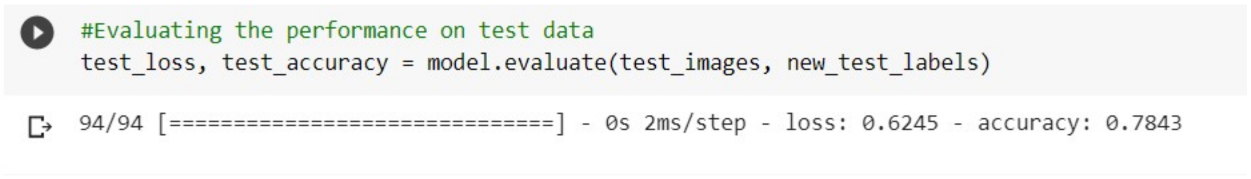
```
94/94 [==============================] - 0s 2ms/step - loss: 0.5863 - accuracy: 0.7987
```





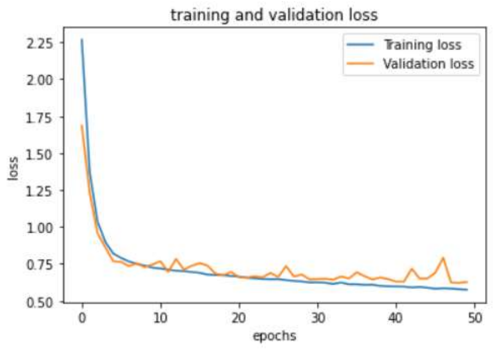
Note that increasing and decreasing the learning rates here is not helpful in increaing the accuracy.

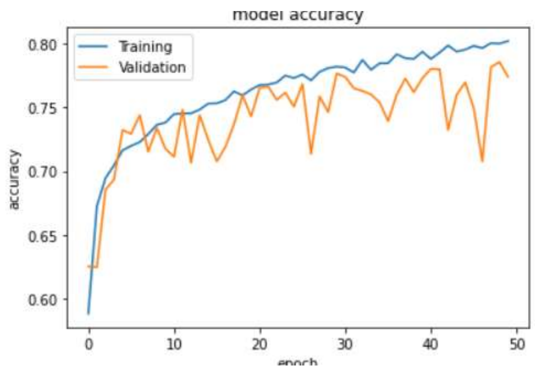
Also increaing the regularization value is not helping the model to converge faster as below (0.01→0.03)



```
#Evaluating the performance on test data
test_loss, test_accuracy = model.evaluate(test_images, new_test_labels)
```

```
94/94 [==============================] - 0s 2ms/step - loss: 0.6245 - accuracy: 0.7843
```





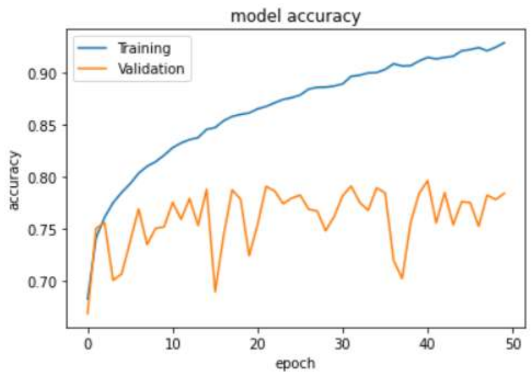
Model 5 –

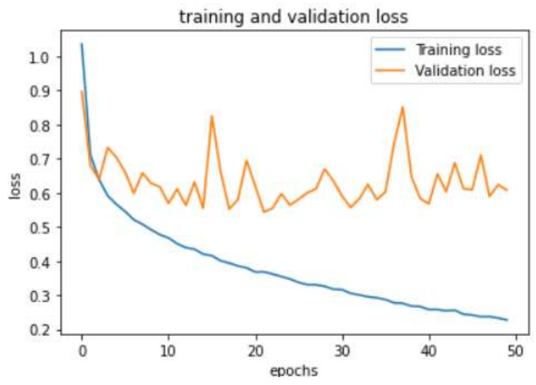
Adding Batch Normalization without dropout-

```
model= Sequential()
model.add(Dense(units=50,input_shape=(32*32*3,)))
model.add(BatchNormalization(axis=-1, momentum=0.99, epsilon=1e-
07, center=True, scale=True, beta_initializer='zeros', gamma_initializer='
ones', moving_mean_initializer='zeros', moving_variance_initializer='ones'
, beta_regularizer=None, gamma_regularizer=None, beta_constraint=None, gam
ma_constraint=None))
model.add(Activation('relu'))
model.add(Dense(units=20))
#model.add(Dropout(0.5))
#model.add(Dense(units=100))
#model.add(Dense(20))
model.add(Dense(units=10,activation='softmax'))
model.summary()
```

Output-

The model does not converge effectively and there are some spikes which are flattening out once we add dropout with BatchNormalization-



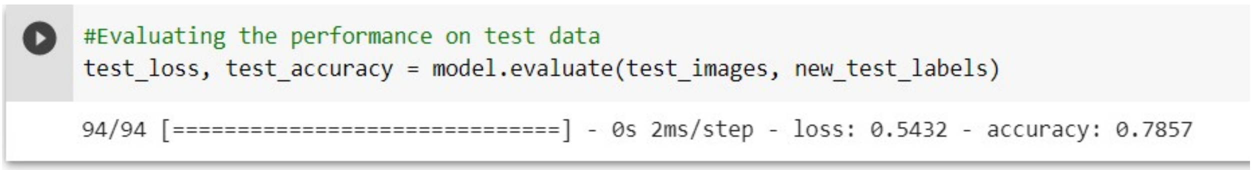


Adding Batch Normalization with dropout-

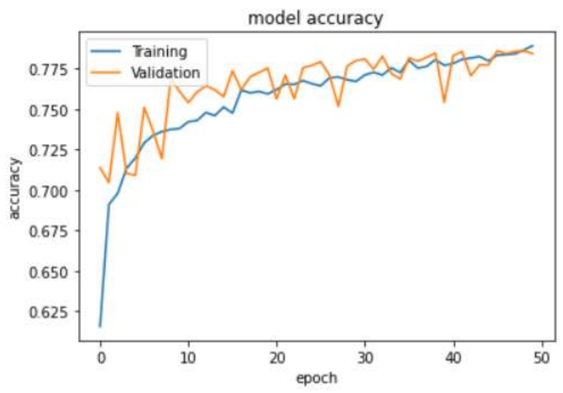
```
model= Sequential()
model.add(Dense(units=50,input_shape=(32*32*3,)))
model.add(BatchNormalization(axis=-1, momentum=0.99, epsilon=1e-
07, center=True, scale=True, beta_initializer='zeros', gamma_initializer='
ones', moving_mean_initializer='zeros', moving_variance_initializer='ones'
, beta_regularizer=None, gamma_regularizer=None, beta_constraint=None, gam
ma_constraint=None))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(units=20))
#model.add(Dropout(0.5))
#model.add(Dense(units=100))
#model.add(Dense(20))
model.add(Dense(units=10,activation='softmax'))
model.summary()
```

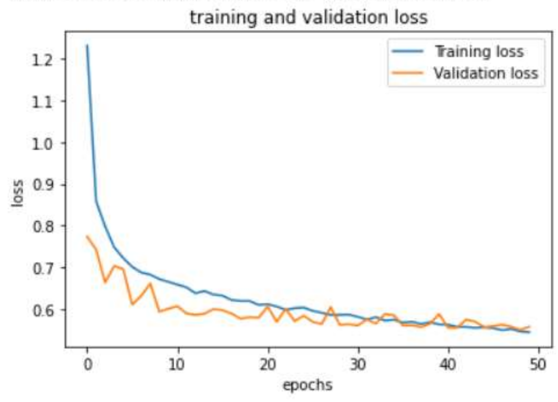
Output-

The model converges and we can see there is no overfitting and both training and validation sets have similar accuracy and loss value.



```
#Evaluating the performance on test data
test_loss, test_accuracy = model.evaluate(test_images, new_test_labels)

94/94 [==============================] - 0s 2ms/step - loss: 0.5432 - accuracy: 0.7857
```



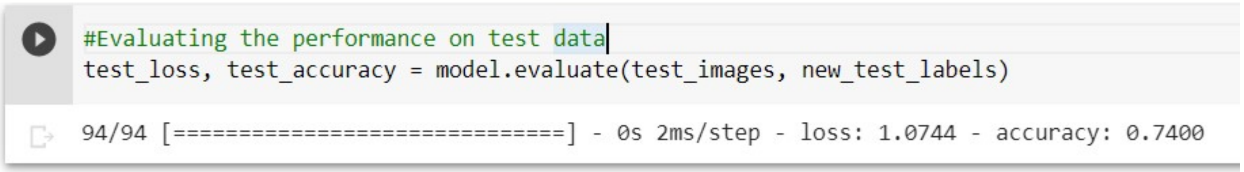


Model 6 –

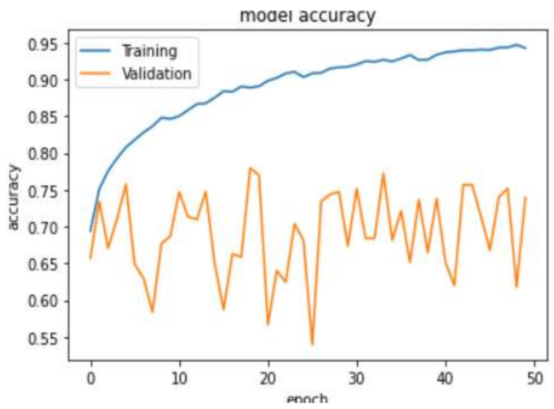
Using RMSProp optimizer –

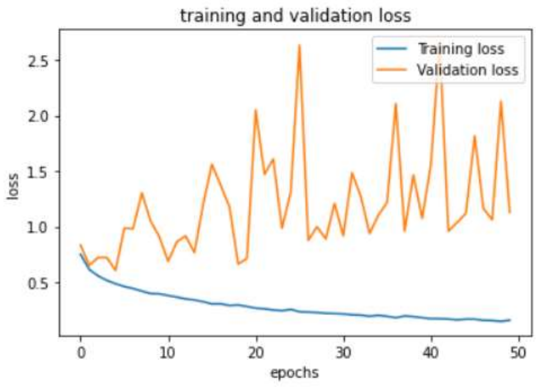
```
opt1 =keras.optimizers.RMSprop(
    learning_rate=0.007,
    epsilon=1e-07,
    name="RMSprop")
model.compile(loss='categorical_crossentropy',optimizer=opt1,metrics=['acc
uracy'])
```

Output -
The model does not converge well with RMSProp and even adjusting the learning rate and dense layer units is not helping to converge the model.



```
#Evaluating the performance on test data
test_loss, test_accuracy = model.evaluate(test_images, new_test_labels)

94/94 [==============================] - 0s 2ms/step - loss: 1.0744 - accuracy: 0.7400
```





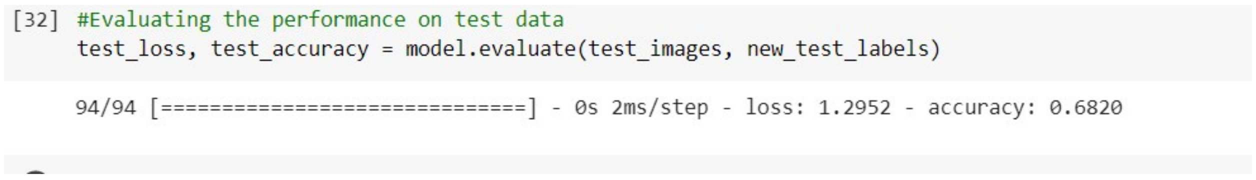
<u>Model 7 –</u>

<u>Using Adam optimizer –</u>

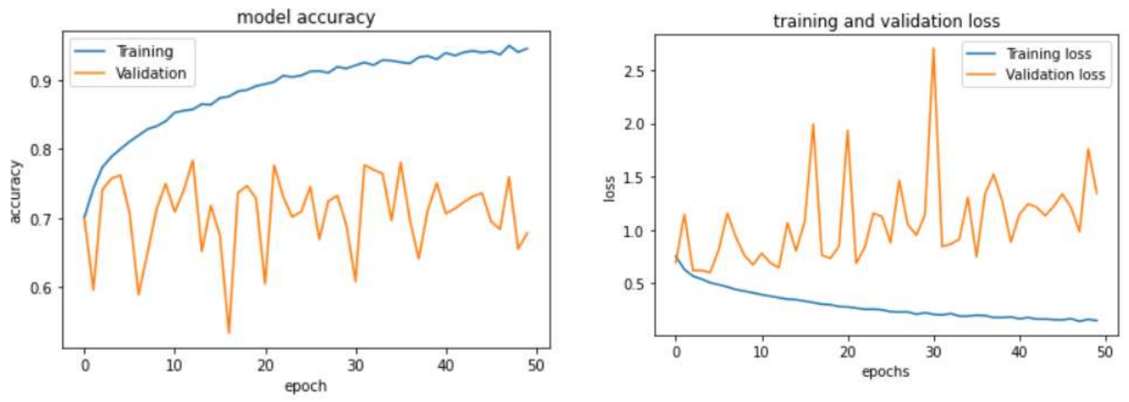
```
opt2=keras.optimizers.Adam(
    learning_rate=0.003,
    #initial_accumulator_value=,
    epsilon=1e-07,
    name="Adam"
)
model.compile(loss='categorical_crossentropy',optimizer=opt2,metrics=['acc
uracy'])
```

<u>Output -</u>

The model does not converge properly and the we can see the overfitting and testing loss more than validation loss. The hyperparameters adjustment is not helping much in this case-



```
[32] #Evaluating the performance on test data
     test_loss, test_accuracy = model.evaluate(test_images, new_test_labels)

     94/94 [==============================] - 0s 2ms/step - loss: 1.2952 - accuracy: 0.6820
```



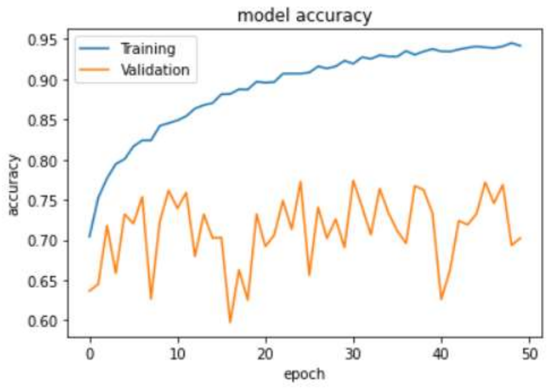
Model 8 –

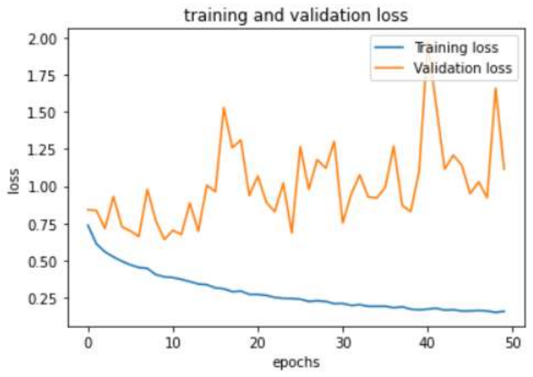
Using SGD optimizer and sparse categorical crossentropy –

```
opt3 = keras.optimizers.SGD(lr=0.01, momentum=0.9)
model.compile(loss='sparse_categorical_crossentropy', optimizer=opt3, metr
ics=['accuracy'])
```

Output-

The model is not converging as expected and there are many spikes and overfitting-





4. Results and Discussions

- The best model selected were model 2 and model 3. Even model 4 with L2 regularizer was smooth and gave proper results after some adjustments to the weights.
- The major problem occurred while creating the best combinations of loss, regularization methods and optimizers along with which hyperparameter needs to be adjusted.
- In most of the cases adjusting learning rate and dense layer was helping in converging the graph.

1. **Problem statement-**
   Given the communities and crime dataset. The goal is to predict ViolentCrimesPerPop.
   This is a regression problem.

   We need to try out different optimizers, regularization methods, loss functions and then draw the conclusion based on the observation.

2. **Proposed solution-**
   - We will use the pandas library to read the CSV file and make any changes to the dataframe.
   - Load_crime_data is the method used for downloading, cleaning, normalizing and distributing the data
   - Downloading the data using wget so that we don't have to maintain the CSV file externally.
   - We will be using a combination of relu and linear activation functions on the dense layers.

   - As this is a regression problem we are going to use the following loss functions-
     Mean Squared Error Loss
     Mean Squared Logarithmic Error Loss
     Mean Absolute Error Loss

3. **Implementation details**

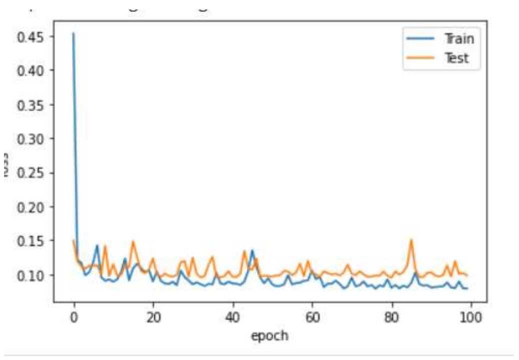
   - Load_ crime_ data method is used to download the data set.
   - Any missing values as '?' is replaced by NAN, to be replaced by zero or mean if required.
   - Normalization is done using min max
   - Any missing value is replaced by 0

Model 1-

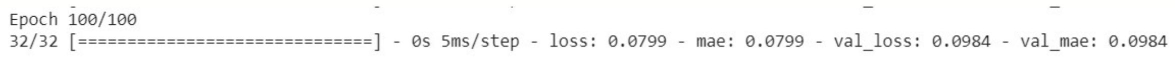
```
model.add(tf.keras.layers.Dense(300, input_shape=[122,]))
  model.add(tf.keras.layers.Dense(200, activation='relu'))
  model.add(tf.keras.layers.Dense(25,activation='relu'))
  model.add(tf.keras.layers.Dense(1))
  model.summary()


opt=keras.optimizers.Adam(learning_rate=0.003,epsilon=1e-07,name="Adam")
model.compile(optimizer=opt, loss=tf.keras.losses.MeanAbsoluteError(),metr
ics=['mae'])
model.fit(x_training, y_training, epochs=100)
```

Output-

Graph is overfitting-



Also the training data loss and test data loss have little difference, which could be taken care of using some regularizers –



```
Epoch 100/100
32/32 [==============================] - 0s 5ms/step - loss: 0.0799 - mae: 0.0799 - val_loss: 0.0984 - val_mae: 0.0984
```

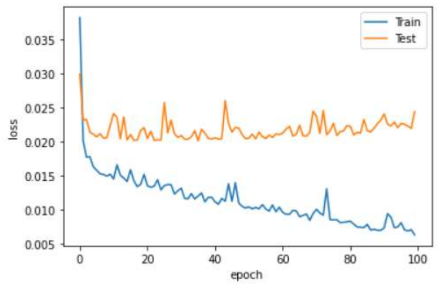
Model 2-

Using MSE loss and SGD – momentum optimizer

```
opt1 = keras.optimizers.SGD(lr=0.01, momentum=0.9)
model.compile(loss='mean_squared_error', optimizer=opt1, metrics=['mse'])
```

Output-

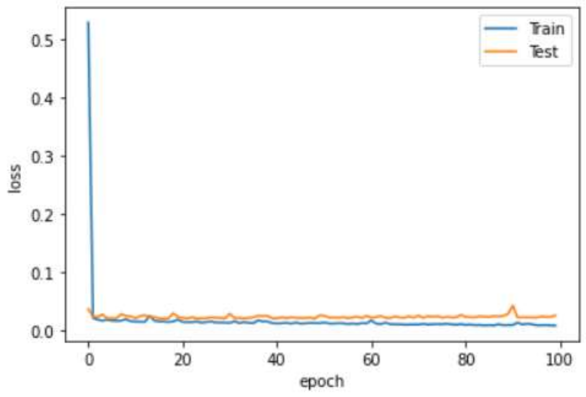
There is a huge overfitting-



Modifying the dense layers and replacing SGD with Adam –

```
model = Sequential()
model.add(tf.keras.layers.Dense(300, input_shape=[122,],activation='relu')
)
#model.add(tf.keras.layers.Dense(200, activation='relu'))
model.add(tf.keras.layers.Dense(25,activation='relu'))
model.add(tf.keras.layers.Dense(1))
model.summary()
```

Output-
We get a smoother graph-

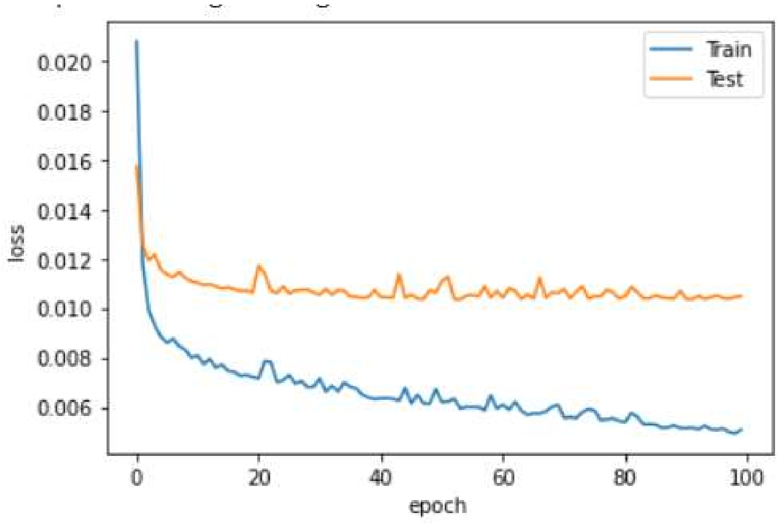


Model 3-

Mean Squared Logarithmic Error with SGD- momentum

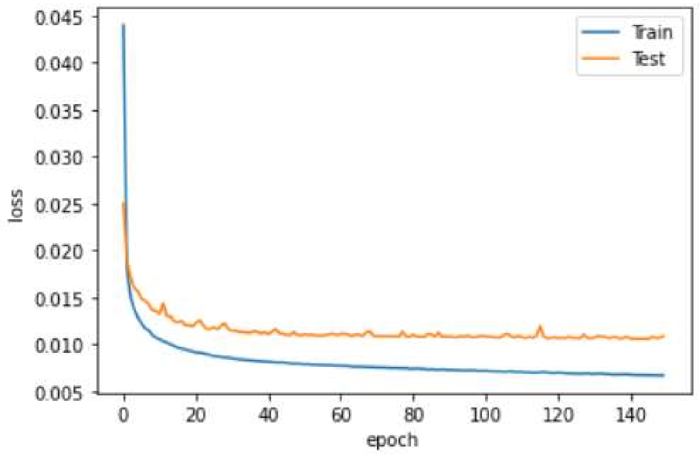
```
opt1 = keras.optimizers.SGD(lr=0.01, momentum=0.9)
#model.compile(optimizer=opt, loss=tf.keras.losses.MeanAbsoluteError(),met
rics=['mae'])
#model.compile(loss='mean_squared_error', optimizer=opt, metrics=['mse'])
model.compile(loss='mean_squared_logarithmic_error', optimizer=opt1, metri
cs=['mse'])
history=model.fit(x_training, y_training,validation_data=(x_test, y_test),
 epochs=100)
```

Output-
It shows a high level of overfitting-



Decreasing the momentum from 0.9→0.1 and increasing the number of epochs 100→150

The graph smoothes out and overfitting is reduced –

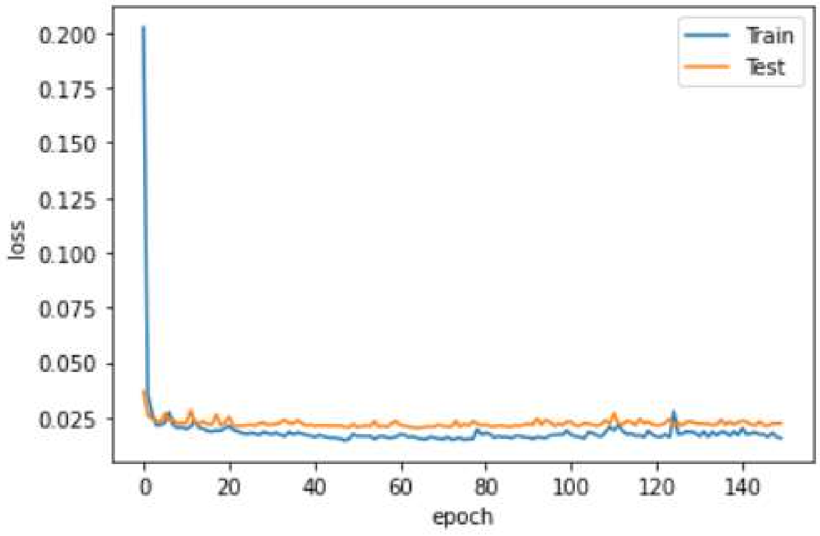


Model 4 –

Adding dropout layer +MSE+ Adam

```
model = Sequential()
model.add(tf.keras.layers.Dense(300, input_shape=[122,],activation='relu')
)
model.add(Dropout(0.5))
#model.add(tf.keras.layers.Dense(200, activation='relu'))
model.add(tf.keras.layers.Dense(25,activation='relu'))

model.add(tf.keras.layers.Dense(1))
model.summary()
```

Output-
The model is converging with very little overfitting.



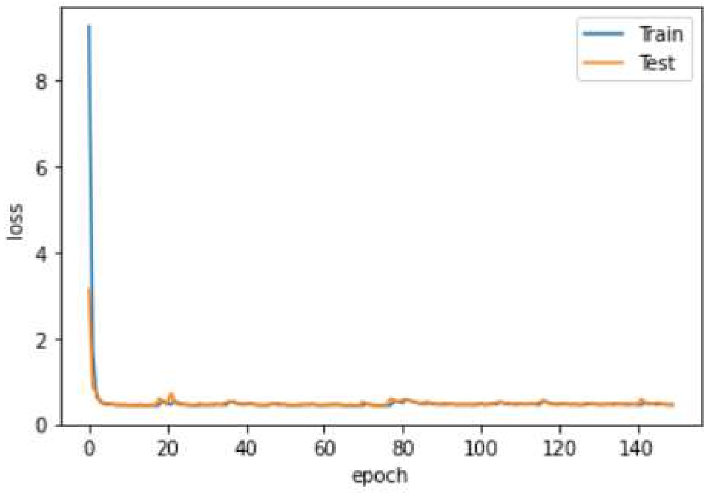
Model 5-

Adding L1 and L2 Regularizer (weight Decay)

For L1 Regularizer-

```
model = Sequential()
model.add(tf.keras.layers.Dense(300, input_shape=[122,],activation='relu',
kernel_regularizer=l1(0.01), bias_regularizer=l1(0.01)))
#model.add(Dropout(0.5))
#model.add(tf.keras.layers.Dense(200, activation='relu'))
model.add(tf.keras.layers.Dense(25,activation='relu'))

model.add(tf.keras.layers.Dense(1))
model.summary()
```

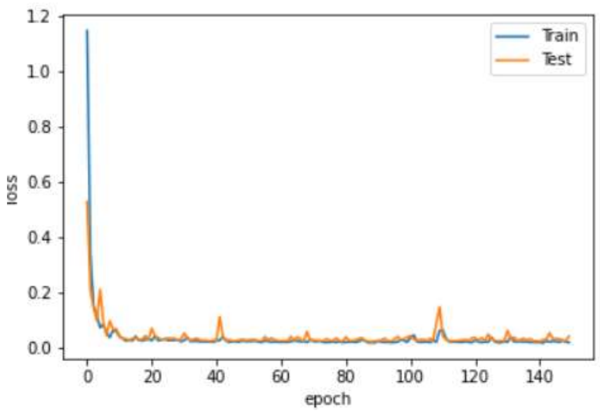
Output-
The model converges nearly smooth with L1 regularizer, with no overfitting-



For L2 Regularizer-

```
model = Sequential()
model.add(tf.keras.layers.Dense(300, input_shape=[122,],activation='relu',
kernel_regularizer=l2(0.01), bias_regularizer=l2(0.01)))
#model.add(Dropout(0.5))
#model.add(tf.keras.layers.Dense(200, activation='relu'))
model.add(tf.keras.layers.Dense(25,activation='relu'))
model.add(tf.keras.layers.Dense(1))
model.summary()
```

The model converges but has some spikes and is not as smooth as L1.

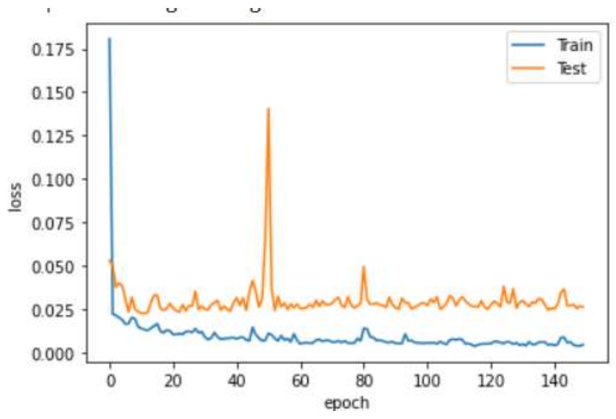


Model 6-

Adding Batch Normalization without dropout-

```
model = Sequential()
model.add(tf.keras.layers.Dense(300, input_shape=[122,]))
model.add(BatchNormalization(axis=-1, momentum=0.99, epsilon=1e-
07, center=True, scale=True, beta_initializer='zeros', gamma_initializer='
ones', moving_mean_initializer='zeros', moving_variance_initializer='ones'
, beta_regularizer=None, gamma_regularizer=None, beta_constraint=None, gam
ma_constraint=None))#model.add(Dropout(0.5))
model.add(Activation('relu'))
#model.add(tf.keras.layers.Dense(200, activation='relu'))
model.add(tf.keras.layers.Dense(25,activation='relu'))

model.add(tf.keras.layers.Dense(1))
model.summary()
```

Output-
Model shows overfitting and sharp spikes, for testing dataset-



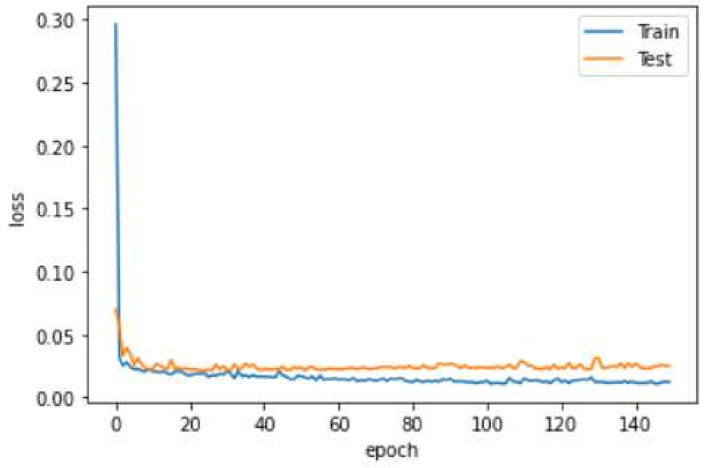
Adding Batch Normalization with dropout-

```
model = Sequential()
model.add(tf.keras.layers.Dense(300, input_shape=[122,]))
model.add(BatchNormalization(axis=-1, momentum=0.99, epsilon=1e-
07, center=True, scale=True, beta_initializer='zeros', gamma_initializer='
ones', moving_mean_initializer='zeros', moving_variance_initializer='ones'
, beta_regularizer=None, gamma_regularizer=None, beta_constraint=None, gam
ma_constraint=None))
model.add(Activation('relu'))
model.add(Dropout(0.5))
#model.add(tf.keras.layers.Dense(200, activation='relu'))
model.add(tf.keras.layers.Dense(25,activation='relu'))

model.add(tf.keras.layers.Dense(1))
model.summary()
```

Output-

The model converges better with low overfitting -

```
<matplotlib.legend.Legend at 0x7fd847c1a290>
```



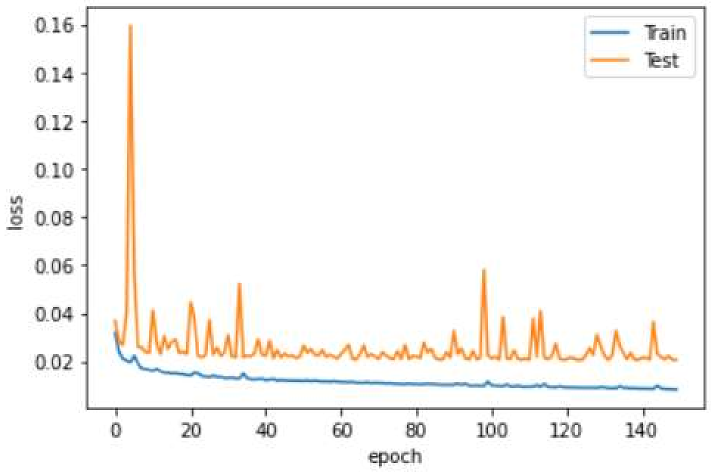
Model 7 –

Using Adagrad Optimizer with MSE loss

```
opt2 = keras.optimizers.Adagrad(lr=0.01, epsilon=1e-06)
#model.compile(optimizer=opt, loss=tf.keras.losses.MeanAbsoluteError(),met
rics=['mae'])
model.compile(loss='mean_squared_error', optimizer=opt2, metrics=['mse'])
#model.compile(loss='mean_squared_logarithmic_error', optimizer=opt1, metr
ics=['mse'])
history=model.fit(x_training, y_training,validation_data=(x_test, y_test),
 epochs=150)
```

Output-

The graph converges but not smoothly we can see many shoot out points while testing –



5. Results and Discussions

- The best model selected were model 2,model 3 converges properly after some hyperparameter adjustments. Even L1 regularizer was smooth and gave proper results without any dropouts.
- The major problem occurred while creating the best combinations of loss, regularization methods and optimizers along with which hyperparameter needs to be adjusted.
- In most of the cases adjusting learning rate and dense layer was helping in converging the graph.