CS 577 Assignment-1
Vatsal Ketan Patel
A20458061
Spring 2020

TASK – 1 (CIFAR10 Dataset):

Problem Statement

Creating a subset of Cifar10 Database with just 3 classes instead of 10 and categorically encode the vectorized images by dividing the test batch into testing set and validation set. Creating a fully connected network to perform multi-class classification on this data.

Implementation Details

- *Program Design Issues and their solutions*

1. Extracting a subset of 3 classes from Cifar10(an inbuilt database in Keras with 10 classes) as there is no in-built method to extract a subset of the database.
    - Extracted the index positions of the classes which I used for the task i.e. (Automobile, Horse, Ship) -> (1,7,8) and created a train, validation and test case only for the above-mentioned classes.

```
30 #Extracting the subset of 3 classes(Automobile, horse, ship) from the cifar10 Dataset
31 def subset(subset_labels):
32   arr = []
33   for index,label in enumerate(subset_labels):
34     if label in [1,7,8]:
35       arr.append(index)
36   return arr
```

2. Converting the extracted labels of the classes or else there will be an anomaly in the shape of the array while categorically encoding the label indices.
    - Converted the index labels of the extracted classes i.e. (1,7,8) to (0,1,2).

```
51 #Converting extracted labels 1,7,8 to 0,1,2
52 def label_convert(new_labels):
53   for index,value in enumerate(new_labels):
54     if value == 1:
55       new_labels[index] = 0
56     elif value == 7:
57       new_labels[index] = 1
58     else:
59       new_labels[index] = 2
60   return new_labels
61
62 y_train = label_convert(y_train)
63 y_test = label_convert(y_test)
```

Proposed solution

Converting the dataset of (50000,10000) train and test images into (15000,3000) train and test images respectively according to the class labels chosen(1,7,8 for me). Further dividing the test images into training and validation images with test_size=75% i.e. 11250 test images and 3750 validation images with each image shape of(32,32,3). Vectorizing all the images for Fully connected neural network we have images of the shape 3072. As it's a multi-class classification problem the labels are one-hot encoded. All the images are normalized by dividing them by 255.

Model Design:

1. Implementing a Sequential neural network model as we want an input x an output y, but don't need previous inputs to predict outputs.
2. The sequential model contains 3 layers, with 2 hidden layers with 256 and 128 nodes respectively, for achieving higher accuracy then using a single hidden layer.
3. The hidden layers have been using 'ReLU' as activation function because it does not activate all the neurons at the same time i.e. more computationally efficient then tanh and sigmoid function.
4. The final layer uses 'SoftMax' as activation function as it gives probabilities of our classes with a combined probability of 1. The output class is the one with the maximum probability among all the classes.
5. Dropout is also set to 0.35 i.e. 35% of the neurons will be deactivated so as to avoid overfitting of the model.
6. The model is compiled using 'sgd'(stochastic gradient descent) over gradient descent because the training time is a lot faster and sgd gives us the global minima rather than local minima.
7. The learning rate is defined so as to define the adjustment of weights with accordance to finding the local minima and momentum for adjusting the right speed to by-pass the local minima, so we achieve global minima.
8. The loss function used to compile the model is 'categorical_crossentropy' because the target variable has 3 classes.
9. The metrics used is 'accuracy' as we want to check how much accurately our model is performing in predicting the classes.

```
#Creating fully connected Artificail Neural Network
model = Sequential()
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Activation('relu'))
model.add(Dense(3, activation='softmax'))

sgd = SGD(lr=0.02, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='sgd')
history = model.fit(x_train, y_train, batch_size=64, epochs=24, validation_data =(x_val,y_val))
```

Results and Discussion

1. README

   Tuned the parameters such as batch size and epochs for finding the best fit. The one that worked well for me was batch-size:64, epochs between (24 to 32). Split the dataset into 75% test batch and 25% validation batch to obtain better results. Edited the number of neurons in the dense layers (Fully connected ANNs) and tweaked a little with dropout cases to prevent overfitting. 1st dense layer: 256, 2nd dense layer: 128, dropout rate: 35%, 3rd dense layer:3.

```
In [66]: model.summary()
Model: "sequential_34"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_100 (Dense)            (None, 256)               786688
_________________________________________________________________
dropout_13 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_101 (Dense)            (None, 256)               65792
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
activation_7 (Activation)    (None, 256)               0
_________________________________________________________________
dense_102 (Dense)            (None, 3)                 771
=================================================================
Total params: 853,251
Trainable params: 853,251
Non-trainable params: 0
_________________________________________________________________
```

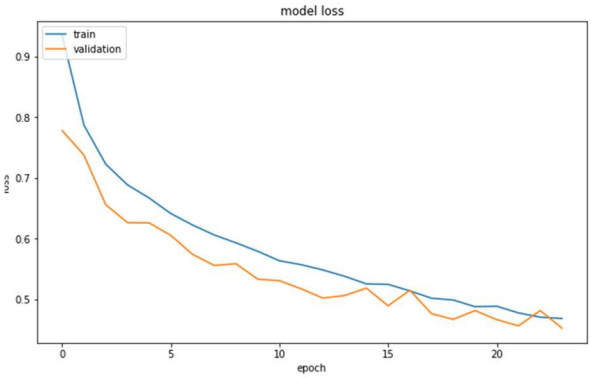
2. 11250 images were trained and validated on 3750 samples for 24 epochs.
   Epoch 1: Val_accuracy > Test_accuracy because of initial epoch & high loss on trainset.
   Epoch 15: Test_accuracy>Val_accuracy and looks like it will start to overfit.
   Epoch 24: Test_accuracy ≈ Val_accuracy.

```
Train on 11250 samples, validate on 3750 samples
Epoch 1/24
11250/11250 [==============================] - 3s 262us/step - loss: 0.9349 -
accuracy: 0.5534 - val_loss: 0.7781 - val_accuracy: 0.6576

Epoch 15/24
11250/11250 [==============================] - 10s 922us/step - loss: 0.5254 -
accuracy: 0.7975 - val_loss: 0.5185 - val_accuracy: 0.7875

Epoch 24/24
11250/11250 [==============================] - 1s 123us/step - loss: 0.4524 -
accuracy: 0.8309 - val_loss: 0.4424 - val_accuracy: 0.8363
```

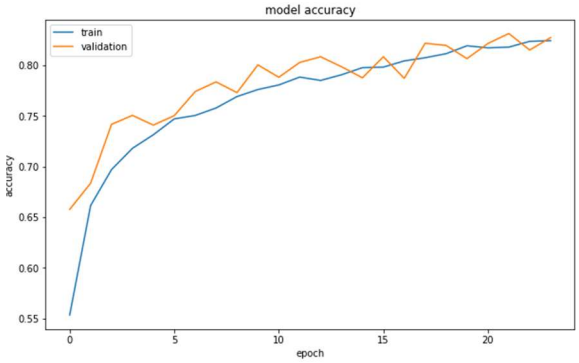
3. After evaluating the model with the test case, we achieve an accuracy of 83.366%

```
3000/3000 [==============================] - 0s 56us/step
Accuracy: 83.36666822433472 2
```

4. This graph shows the training and validation losses as a function of the number of epochs with the best validation loss of 44.24%.



5. This graph shows the training and validation accuracy as a function of the number of epochs with the best validation accuracy of 83.63%.



References

http://home.mit.bme.hu/~hadhazi/Oktatas/NN18/dem3/html_demo/CIFAR-10Demo.html

https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

TASK – 2 (UCI Spambase):

Problem Statement

Creating a neural network to perform binary classification whether an email is spam or not using UCI repository "Spambase".

Implementation Details

- *Program Design Issues and their solutions*
  1. Splitting the dataset into test, train and validation sets randomly since the data contains only 4500 rows out of which the data in the last column has only 2 values 0 and 1, the approx. first 2500 rows contain only 1 and the rest contain only 0 indicating that it requires shuffling or random distribution of the data.
     - Using the sklearn's: train_test_split function and assigning the random state to each split being a crucial part of the process.

```
19  df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data', header=Non
20  x = df.iloc[:, :57].values
21  y = df.iloc[:, 57].values
22  x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)
23  x_train,x_val, y_train,y_val = train_test_split(x_train,y_train , test_size = 0.2, random_state = 42)
```

Proposed solution

After loading the data, the first 57 columns are assigned to x and the last column to y. The data is split into training, validation and testing sets using sci-kit learn train test split method with test_split of 20% each. The data is normalized and scaled to obtain the data in similar range by sklearn's:'StandardScaler' function.

Model Design:

1. Implementing a Sequential neural network model as we want an input x an output y, but don't need previous inputs to predict outputs.
2. The sequential model contains 3 layers, with 2 hidden layers with 32 and 16 nodes respectively, for achieving higher accuracy then using a single hidden layer.
3. The hidden layers have been using 'ReLU' as activation function because it does not activate all the neurons at the same time i.e. more computationally efficient then tanh and sigmoid function.
4. The final layer uses 'sigmoid' as activation function as it gives probabilities of our classes between 0 and 1.
5. The model is compiled using 'rmsprop' over gradient descent because the training time is a lot faster when dividing the dataset into batches.
6. The loss function used to compile the model is 'binary_crossentropy' because the target variable has only 2 classes.
7. The metrics used is 'accuracy' as we want to check how much accurately our model is performing in predicting the classes.

```
#Builiding a sequential model
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(57,)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(x=x_train, y=y_train, batch_size=32, epochs=24, validation_data=(x_val,y_val))
```

Results and Discussion

1. README

   Tuned the parameters such as batch size and epochs for finding the best fit. The one that worked well for me was batch-size:64, epochs between (24 to 30). Split the dataset into 80% test batch and 20% validation batch to obtain better results. Edited the number of neurons in the dense layers (Fully connected ANNs) and tweaked a little with dropout cases to prevent overfitting. 1st dense layer: 32, 2nd dense layer: 32, 3rd dense layer: 1

```
Model: "sequential_36"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_106 (Dense)            (None, 32)                1856
_________________________________________________________________
dense_107 (Dense)            (None, 32)                1056
_________________________________________________________________
dense_108 (Dense)            (None, 1)                 33
=================================================================
Total params: 2,945
Trainable params: 2,945
Non-trainable params: 0
_________________________________________________________________
```

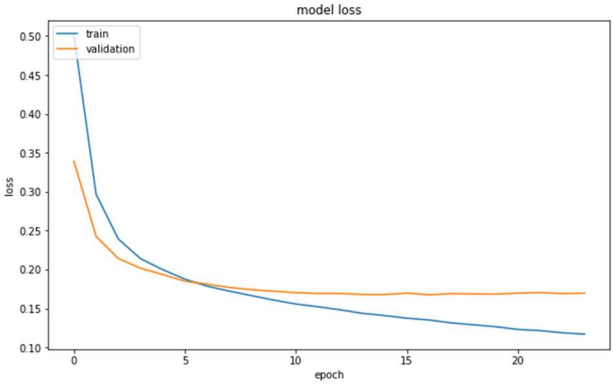
2. Training on 2944 rows and validating on 736 rows for 24 epochs.
   Epoch 1: Val_accuracy > Test_accuracy because of initial epoch & high loss on trainset.
   Epoch 15: Test_accuracy>Val_accuracy and looks like it will start to overfit.
   Epoch 24: Test_accuracy ≈ Val_accuracy

```
Train on 2944 samples, validate on 736 samples
Epoch 1/24
2944/2944 [==============================] - 1s 170us/step - loss: 0.5011 - accuracy:
0.8057 - val_loss: 0.3391 - val_accuracy: 0.9022

Epoch 5/24
2944/2944 [==============================] - 0s 31us/step - loss: 0.1998 - accuracy:
0.9324 - val_loss: 0.1936 - val_accuracy: 0.9280

Epoch 24/24
2944/2944 [==============================] - 0s 32us/step - loss: 0.1170 - accuracy:
0.9555 - val_loss: 0.1695 - val_accuracy: 0.9375
```

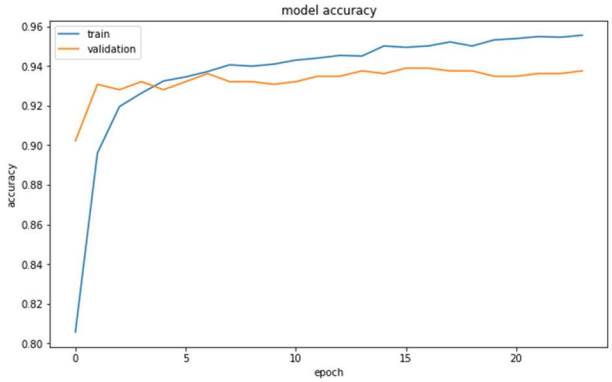
3. After evaluating the model with the test case we achieve an accuracy of 93.75%

```
921/921 [==============================] - 0s 19us/step
Accuracy: 94.46253776550293 2
```

4. This graph shows the training and validation losses as a function of the number of epochs with best validation loss of 16.95%.



5. This graph shows the training and validation accuracy with respect to the number of epochs with the best validation accuracy of 93.75%.



References

https://www.udemy.com/course/machinelearning/learn/lecture/7675934?start=600#overview

https://keras.io/optimizers/

TASK – 3 (UCI Communities and Crime):

Problem Statement

Creating a neural network to predict the ViolentCrimesPerPop using K-Fold Cross Verification using UCI repository “Communities and Crime”.

Implementation Details

- *Program Design Issues and their solutions*

1. Removing the first 5 columns of the dataset because they are irrelevant in predicting the target column as there are more than 75% of the values that are missing from those columns.
   - Storing the new dataset in another object of the load model by taking the columns starting from index number 5.

```
24      new_data = data.iloc[:,5:].values
```

2. Replacing the missing values from the dataset since it will result in error and wrong predictions.
   - This is done by sklearn’s:’SimpleImputer’ function which replaces the missing values in all the columns by the means of their respective columns.

```
22  data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/communities/communities.data',head
23  data = data.replace('?',np.nan)

26      missingvalues = SimpleImputer(missing_values = np.nan , strategy = 'mean', verbose=0)
27      missingvalues = missingvalues.fit(new_data[:,5:])
28      new_data[:,5:]= missingvalues.transform(new_data[:,5:])
```

Proposed solution

After loading the data, the missing values are replaced by ‘np.nan’. The first 5 columns are not related to the prediction since more than 75% values in those columns are missing. The rest of the missing values in other columns are replaced by the means of the respective columns by using SimpleImputer function from sklearn. 122 columns are assigned to x and the last column to y. The data is split into training and testing sets using sci-kit learn train test split method with test_split of 25%.

Model Design:

1. Implementing a Sequential neural network model as we want an input x an output y, but don’t need previous inputs to predict outputs.
2. The sequential model contains 3 layers, with 2 hidden layers with 64 and 64 nodes respectively, for achieving higher accuracy then using a single hidden layer.
3. The hidden layers have been using ‘ReLU’ as activation function because it does not activate all the neurons at the same time i.e. more computationally efficient then tanh and sigmoid function.
4. The final layer uses ‘linear’ as activation function as it gives numeric output.

5. The model is compiled using 'Adam' over gradient descent because it is computationally efficient and is well suited that are large in terms of data and parameters.
6. The loss function used to compile the model is 'mean_squared_error' because it reduces the difference between actual value of target data and predicted value.
7. The metrics used is 'MAE' as we want to get the absolute difference between predicted and actual target values.

```
#Function for builiding the model
def build_model():

    model = Sequential()
    model.add(Dense(64,activation='relu',input_shape=(122,)))
    model.add(Dense(64,activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='Adam', loss='mean_squared_error',metrics=['mean_absolute_error'])
    return model
```

Results and Discussion

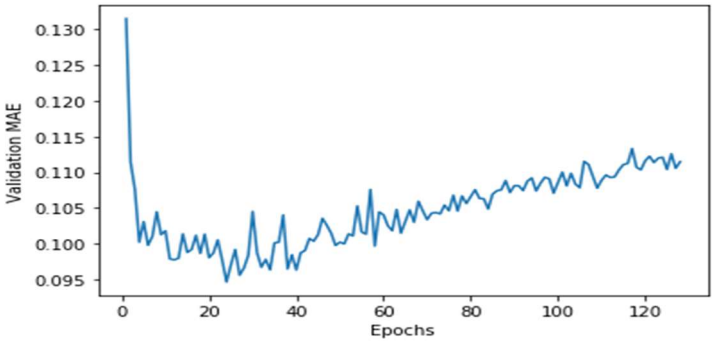
1. README
   Tuned the parameters such as batch size and epochs for finding the best fit. The one that worked well for me was batch-size:64, epochs between (24 to 30). Split the dataset into 80% test batch and 20% validation batch to obtain better results. Edited the number of neurons in the dense layers (Fully connected ANNs) and tweaked a little with dropout cases to prevent overfitting. 1st dense layer: 64, 2nd dense layer: 64, 3rd dense layer: 1

```
Model: "sequential_48"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_142 (Dense)            (None, 64)                7872
_________________________________________________________________
dense_143 (Dense)            (None, 64)                4160
_________________________________________________________________
dense_144 (Dense)            (None, 1)                 65
=================================================================
Total params: 12,097
Trainable params: 12,097
Non-trainable params: 0
```

2. k-fold cross validation(where k=5) has been implemented in this task with 128 epochs and a batch size of 64.

```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
processing fold # 4
```

3. This graph shows the validation mean_absolute_error as a function of the number of epochs with the lowest Validation MAE at epoch number 22 with a value of 0.0942.



4. After evaluating the model with the test case, we have the Mean_squared_error: 0.01875 and Mean_absolute_error:0.0946435

```
499/499 [==============================] - 0s 604us/step

 MSE SCORE:  0.01875420246564434

 MAE SCORE:  0.09464356303215027
```

References

https://www.udemy.com/course/machinelearning/learn/lecture/7675934?start=600#overview

https://keras.io/optimizers/

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/3.7-predicting-house-prices.ipynb