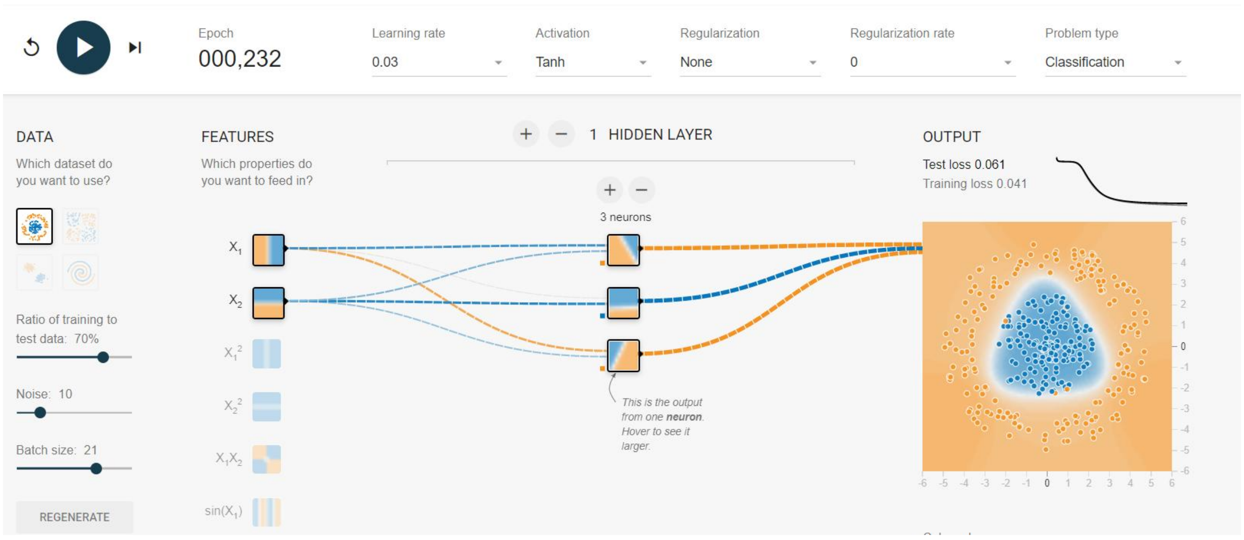
TASK 1 CIRCLE:


Epoch 000,232
Learning rate 0.03
Activation Tanh
Regularization None
Regularization rate 0
Problem type Classification

Ratio of training to test data: 70%
Noise: 10
Batch size: 21

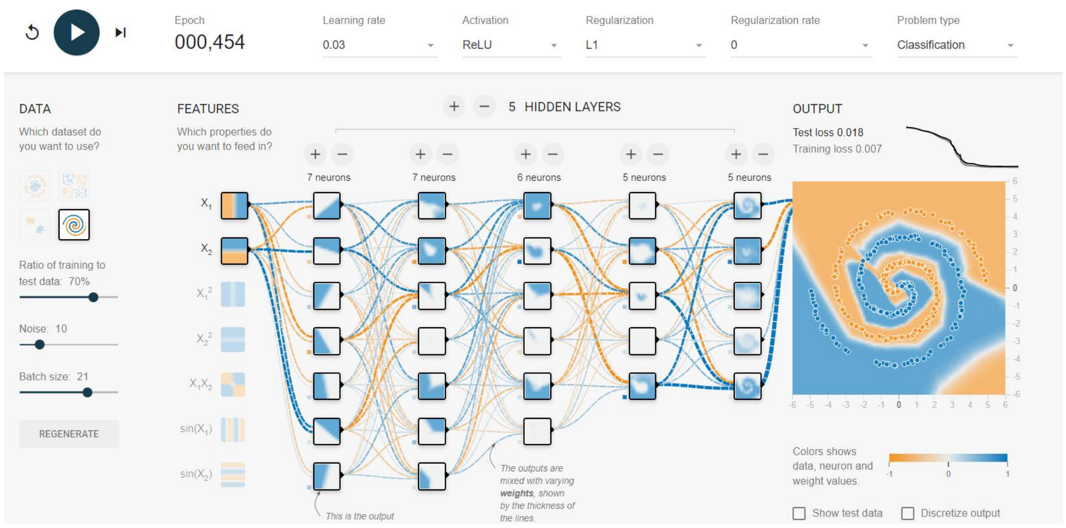
Test loss 0.061
Training loss 0.041

DATA
Which dataset do you want to use?
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT

For this Circle data set , all that is required is 3 Neurons
The hyper parameters are

Learning rate = 0.3	test Loss = 0.061
Activation = tanh	Training loss = 0.041

Regularization = None /L1
Epoch =000,275
Problem type = Classification
It has 2 features , 3 neurons
The ration of training to test data = 70%
Noise = 10
Batch Size = 21


Epoch 000,454
Learning rate 0.03
Activation ReLU
Regularization L1
Regularization rate 0
Problem type Classification

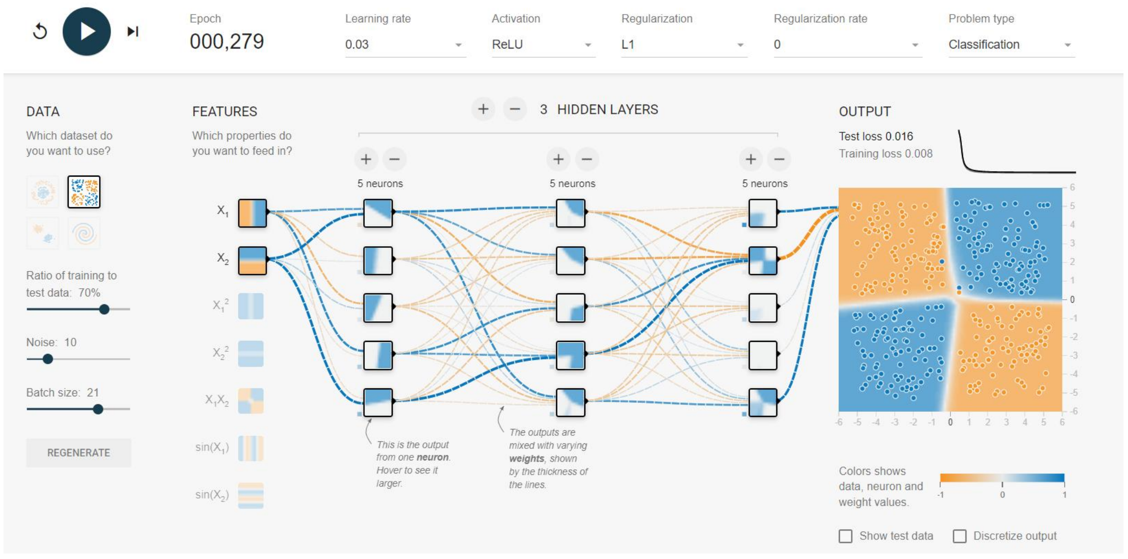
Ratio of training to test data: 70%
Noise: 10
Batch size: 21

Test loss 0.018
Training loss 0.007

DATA
Which dataset do you want to use?
REGENERATE
FEATURES
Which properties do you want to feed in?
5 HIDDEN LAYERS
7 neurons
7 neurons
6 neurons
5 neurons
5 neurons
This is the output
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Colors shows data, neuron and weight values.
Show test data
Discretize output

SPIRAL data set:
This requires more layers an neurons

Learning rate = 0.03	Layers = 5
Epoch = 454	Neuron = 30
Activation = Relu	Batch size = 21
Regularization = L1	Training loss = 0.007
Features = 2	Test loss 0.018



Epoch 000,279

Learning rate 0.03

Activation ReLU

Regularization L1

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 70%

Noise: 10

Batch size: 21

REGENERATE

FEATURES

Which properties do you want to feed in?

3 HIDDEN LAYERS

5 neurons

5 neurons

5 neurons

This is the output from one *neuron*. Hover to see it larger.

The outputs are mixed with varying *weights*, shown by the thickness of the lines.

OUTPUT

Test loss 0.016

Training loss 0.008

Colors shows data, neuron and weight values.

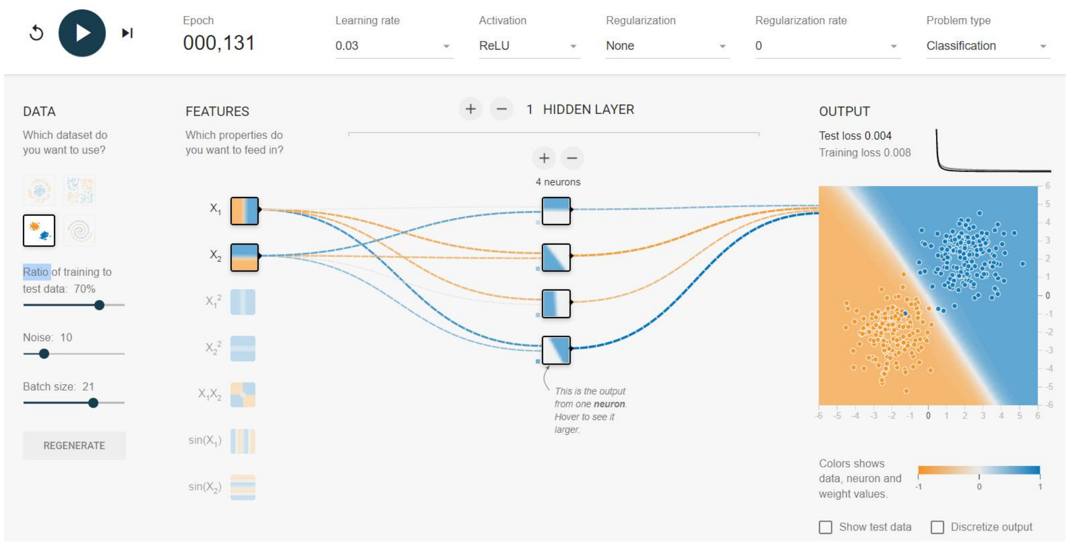
Show test data

Discretize output

Exclusive or

Epoch = 279
Activation = ReLU
Regularization = L1
Batch size = 21
Number of features = 2
Layers = 3
Neurons = 15

Test Loss 0.016
training Loss = 0.008



Epoch 000,131

Learning rate 0.03

Activation ReLU

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 70%

Noise: 10

Batch size: 21

REGENERATE

FEATURES

Which properties do you want to feed in?

1 HIDDEN LAYER

4 neurons

This is the output from one *neuron*. Hover to see it larger.

OUTPUT

Test loss 0.004

Training loss 0.008

Colors shows data, neuron and weight values.

Show test data

Discretize output

Gaussian
Epoch = 279
Activation = ReLU
Regularization = none
Batch size = 21
features = 2
Neurons = 4

Programming Task 2 .

Instructions :
Load the CIFAR10 dataset and select a subset of three classes.
▪ Split the training set into training and validation subset.
▪ Vectorize the images and encode the know class labels using categorical encoding.
▪ Design a fully connected neural network to perform multi-class classification of this data
▪ Justify your network design decisions (number of hidden layers, number of units per layer, loss
function, and evaluation metric)
▪ Build and compile the network designed.
▪ Plot training and validation loss as a function of epochs, plot training and validation accuracy.
▪ Tune model hyper – parameters to improve performance.
▪ Retrain the final mode and test performance on the test collection.
▪ Report the performance obtained.
▪ Save and load the weights of the trained network

Implementation Process :

- import packages

Keras
Matplotlib
Numpy

```
from tensorflow import keras
from tensorflow.keras import layers
from keras.datasets import cifar10
from keras import models
from keras.layers import Dense
import keras.utils as ku
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from keras.layers.core import Activation
import numpy as np
import matplotlib.pyplot as plt
```

-Load the CIFAR10 data and split the data into training and testing subset.
-Select only three classes of the training data. This is achieved by using following numpy function.
train_images[np.where(train_labels[:,0] < 3)[0],:]

We create the training data ,train labels and the test data , test labels from the data.

-Verify the size of new training and testing dataset that have only 3 class out of 10, by using
following method:

train_images_subset.shape, train_labels_subset.shape, test_images_subset.shape, train_labels_subset.shape

convert the training_labels_subset and testing_labels_subset to canonical encoding by using
to_categorical method
to_categorical(train_labels_subset,3)
to_categorical(test_labels_subset,3)

Using the method to_categorical(), a numpy array (or) a vector which has integers that represent different categories, can be converted into a numpy array (or) a matrix which has binary values and has columns equal to the number of categories in the data.

-In order to vectorize we convert the training_images_subset and testing_images_subset to float and divide each pixel by 255

-the training data is split into partial training and validation data.

-then Build a sequential model with input shape (32,32,3) i.e. the dimension of images in the CIFAR10 dataset.

-thereafter add Conv2D layer with (3,3) padding across the edges along with activation function relu.

-add a Flatten() layer which will reduce the size of the image by half.

-add a Dense layer with output size 3 (3 classes out of 10) and an activation function softmax.

-compile the model using the optimizer SGD() (Stochastic Gradient Descend)and Loss Function as categorical_crossentropy and metrics as accuracy.

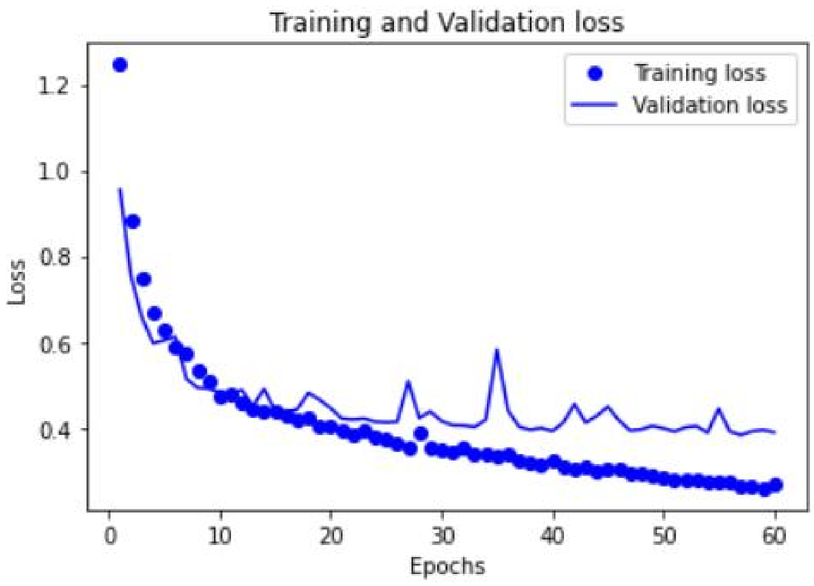
-We now train the model using fit() function for partial training data with a batch size of 512 and 60 epochs and store the training details in history variable.
-The model is validated using the validation data.

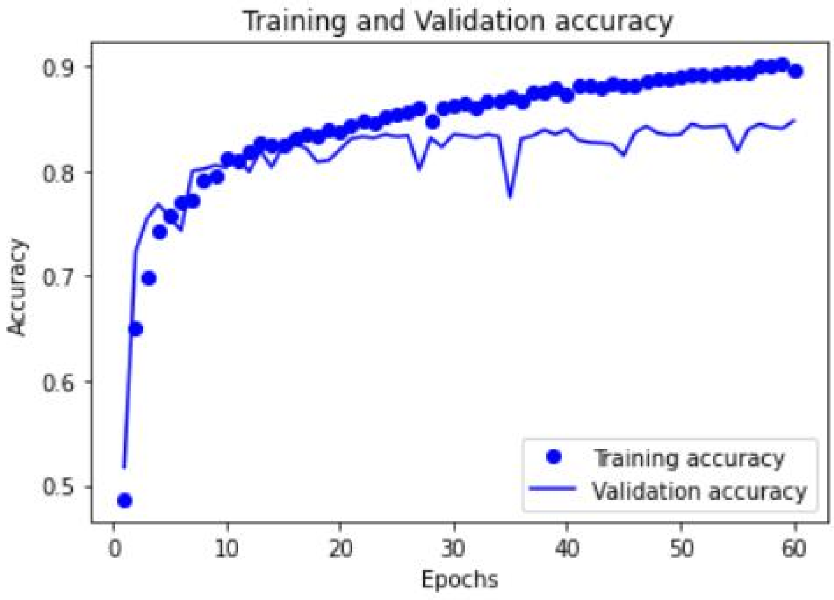
-Then we plot training and validation loss with respect to epochs by using matplotlib.

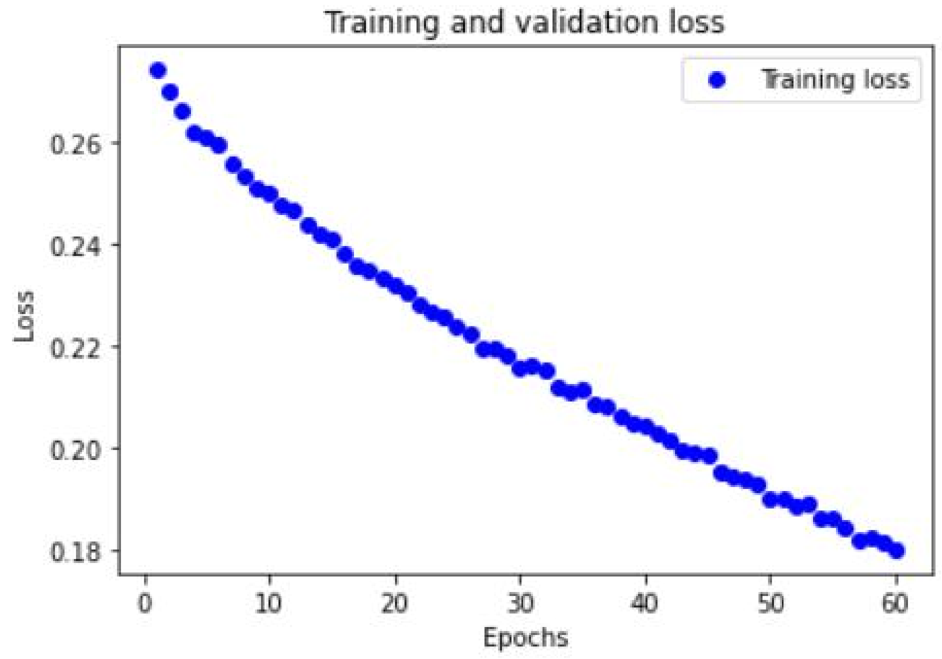
-Similarly we plot the training and validation accuracy by using matplotlib.

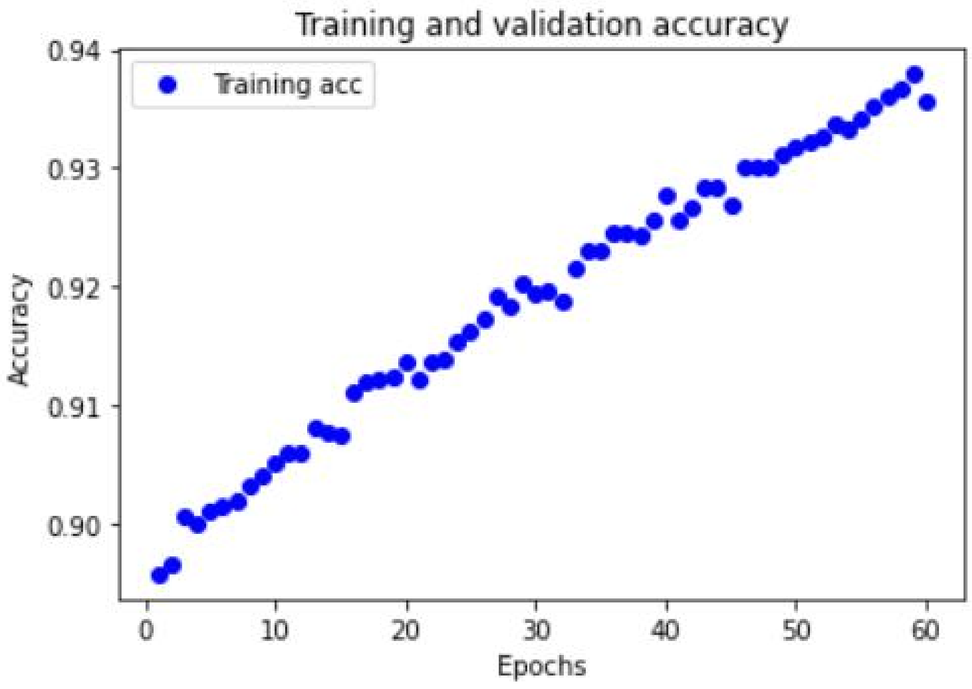
-Retrain the model using learning rate of 0.001 as the model accuracy was dropping after every 8 epochs. Also I have trained the final model for 60 epochs.

-Reload the model and test the model accuracy on test data. I obtain 85.03 % accuracy on the testing data.






Training and validation loss
Training loss
Loss
0.26
0.24
0.22
0.20
0.18
0
10
20
30
40
50
60
Epochs


Training and validation accuracy
Training acc
Accuracy
0.94
0.93
0.92
0.91
0.90
0
10
20
30
40
50
60
Epochs

Programming Task 3 :
Problem Statement
▪ Load the spam email data from UCI repository "spambase" (https://archive.ics.uci.edu/ml/datasets/spambase)
▪ Prepare the data you load as tensor suitable for neural network.
▪ Normalize the features as needed.
Explain the steps in preparing the data and justify them
Write a function "load_spam_data" to load the data and split it into training and testing dataset

Implementation Process :

-Import packages ,
tensorflow,
Keras,
matplotlib and Numpy

-Create a function load_spam_data()

-Extract the values from the dataframe using dataframe.values (this is a process for data preparation)

-Split the data into spam and non_spam datasets, such that the training and testing subset contains the same amount of spam and non_spam data.
-splitting the data:

```
not_spam, spam = np.split(dataframe, np.where(np.diff(dataframe[:,57]))[0]+1)
```

-After splitting the data into equal amounts of spam and non_spam emails
-we again stack the data using the following function:

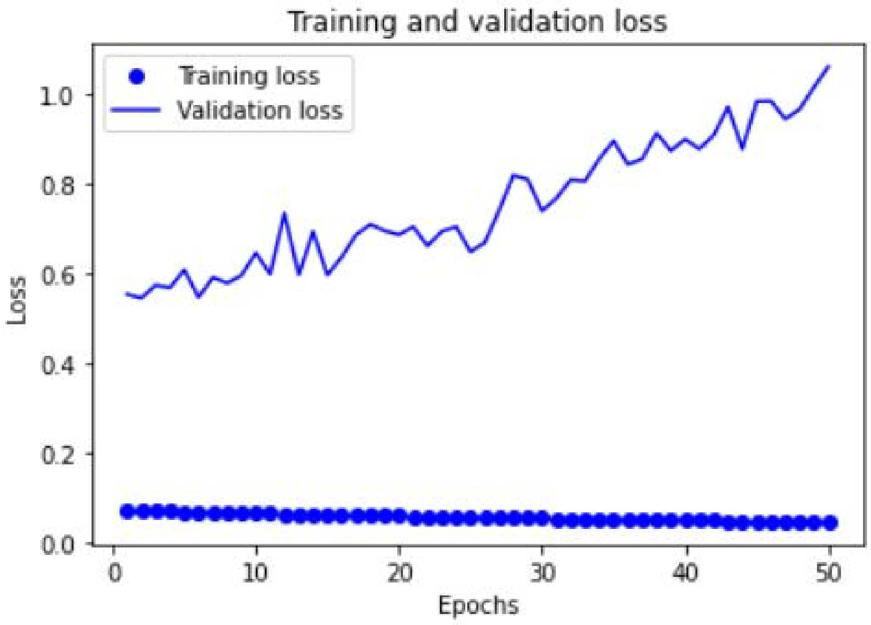
```
train_data = np.vstack([train_ns_emails,train_s_emails])[:,:57]
```

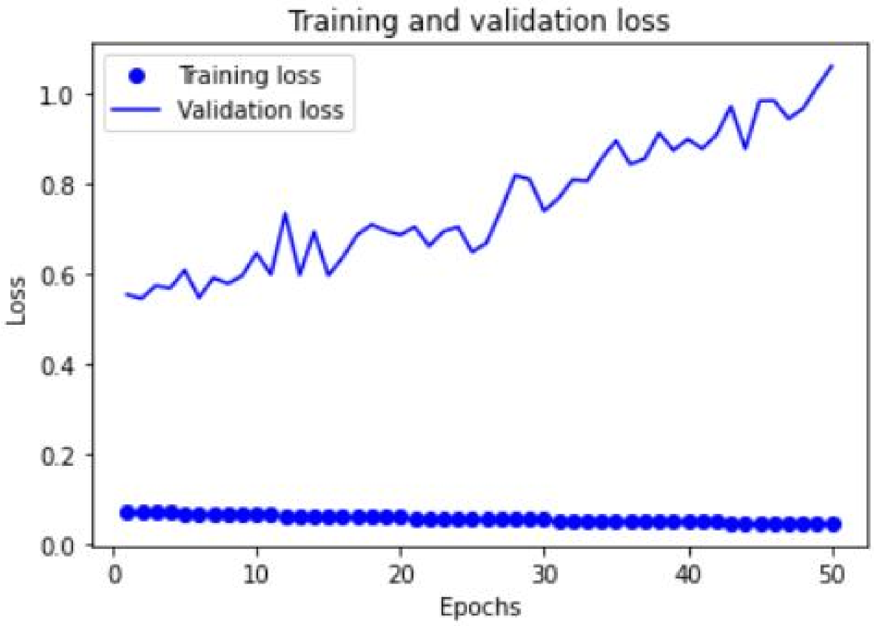
-Load the spambase data and split the data into training and testing subset.

-Normalize the features (*normalization is done by using the mean and standard deviation of the train data,The data can be normalized by subtracting the mean (μ) of each feature and a division by the standard deviation (σ).*
Subtracting the mean centers the input to 0, and dividing by the standard deviation makes any scaled feature value the number of standard deviations away from the mean.)

-Find the mean and Standard deviation of training emails
-Then we subtract all the elements in the dataset with the mean and then we divide it with standard deviation.
-split the training data into partial training and validation data
-Build a sequential model with input size (57,) i.e. the number of columns we have in the dataset.

-Then we add 1 dense layer of size 16 with an activation of relu and a hidden dense layer with activation of relu.
-Then at the end we add the output layer of size 1 (because it is a binary classification) and activation as sigmoid.
-Then we compile the model using the optimizer RMSprop() (Root Mean Square Propogation) and Loss Function as binary_crossentropy and metrics as binary_accuracy.
-We train the model using fit() function for partial training data with a batch size of 32 and 50 epochs and store the training details in history variable..
-The training details are stored in the history variable.
-Then we plot training and validation loss with respect to epochs by using matplotlib.
-Similarly we plot the training and validation accuracy by using matplotlib
-Retrain the model. In this case I didn't change any hyper parameters,but I train the model on the entire training data
-Reload the model and test the model accuracy .


Training and validation loss
Training loss
Validation loss
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50


Training and validation loss
Training loss
Validation loss
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50

Programming Task 4:
Problem Statement
▪ Load the crime data from UCI repository "Communities and crime" (https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime).
▪ Prepare the data you load as a tensor suitable for a neural network.
▪ Normalize features as needed.
▪ Explain the steps you perform in preparing the data and justify them.
. Write the function to "load_crime_data" to load the data and split into training and testing Data sets
▪ Repeat the steps performed for CIRAF 10 expect that in this case perform the k-fold cross validation.
. to report the cross validation results , average the validation error of all the folds and plot it as a function of epochs
. Retrain the final model on all training data(I.e all folds) and test the final performance on the test set.

Implementation Process:

▪ Import all packages that are necessary such as
keras,
matplotlib
and numpy.

The set up looks like this

```
import numpy as np
import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers
from keras.datasets import cifar10
from keras import models
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from keras.layers.core import Activation
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
```

▪ Create a function load_crime_data()

▪ Read the csv file using "pandas.read_csv".and then replace "?" with "np.NAN"

▪ extract the values from the dataframe using dataframe.values

▪ Then split the data into training and testing subsets.

- We take columns from 6 to 127 into consideration . Because columns from 1 – 5 are non predictive and also column 128 is a label. we take column 128 as the class label.

- Since there are many np.nan values in the dataset so we take mean of the column where missing values are present and place it with the mean of that column.

- Build a sequential model with input size (57,) i.e. we have 122 trainable attributes in the training data.

- Then we add 1 dense layer of size 64 with an activation of relu and a hidden dense layer of size 64 with activation of relu.

- Then at the end we add the output layer of size 1 (because it is a binary classification) and activation as sigmoid.

- Then we compile the model using the optimizer RMSprop() (Root Mean Square Propogation) and Loss Function as mse(mean squared error) and metrics as MAE(mean absolute error).

- Then we train the model for using k-fold cross validation. This is done typically because we have less data to work with.

- split the training data into a subset of training and validation data. and take a a different set of training data for each fold as the training and validation data.

- We take scores of each of the fold and append it into an array.

- The MAE(mean absolute error) is mean of all the mae of the K-fold.

- Plot a the graph for the result

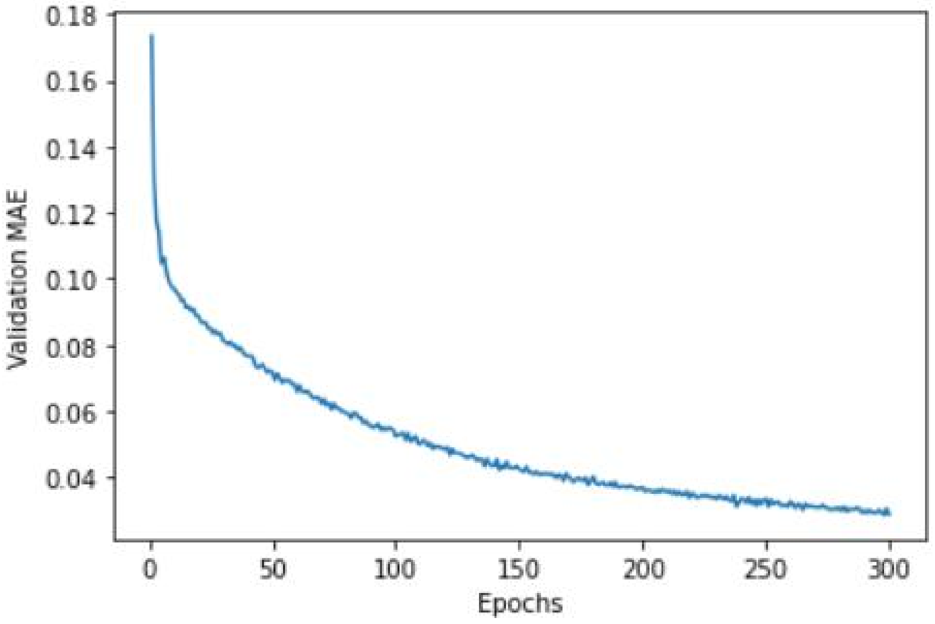
- history variable stores training

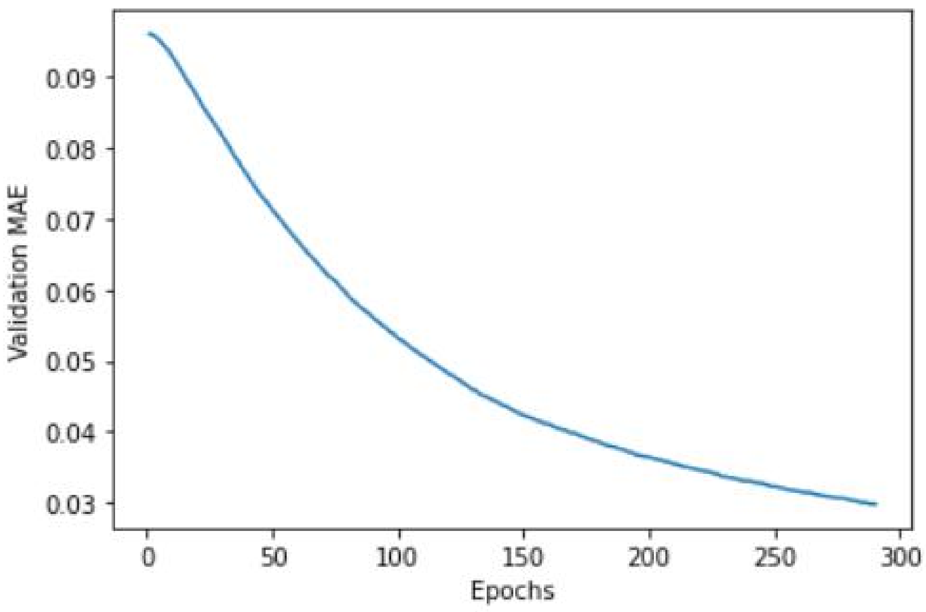
- Retrain the model using all training data for 10 epochs only.

- Save the model as hdf5 file.

- Reload the model and test the model accuracy on test data.

Below is the graph representation of the cross validation results


0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
Validation MAE
0
50
100
150
200
250
300
Epochs


0.09
0.08
0.07
0.06
0.05
0.04
0.03
Validation MAE
0
50
100
150
200
250
300
Epochs