Project Report

CISC 867 Project 1:

the Leaf Classification dataset using a neural network architecture

Monday, 7 march 2022

By:

Aliaa Faisel Kashwa

Afaf Saied Abd elrahman

Amira Abdelghany Nassar

Sara Alaa Mohamed

Dr: Hazem Abbas

Table of Contents

1. Introduction

1.1 Objective of the Report

There are estimated to be nearly half a million species of plant in the world. Classification of species has been historically problematic and often results in duplicate identifications.

1.2 The goal of project

Create Classification in our data especially with **species** feature.

1.3 Data Used

The data used in this project will help to predict the **species** feature.

The original 100 species, we have eliminated one on account of incomplete associated data in the original dataset.

2. The libraries

We used some important libraries in python to help us for building the classifiers :

import libraries

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remour

[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import LabelEncoder # for categorical data
    from sklearn.preprocessing import StandardScaler #to scale the data
    import os
    import random
    import cv2 as cv
    from keras.preprocessing.image import load_img
    from sklearn.utils import shuffle
    import seaborn as sns
    sns.set(
        font_scale=1.5,
        style="whitegrid",
        rc={'figure.figsize':(20,7)}
    )
```

Figure 1: libraries

3. Data processing

```
[ ] #reading the train data from path
    trainData = pd.read_csv('/content/drive/MyDrive/leaf-classification/train.csv')

    #reading the train data as all columns except the first one(id column)
    trainData = trainData.iloc[:,1:]

    #displaying the first 5 rows from train data
    trainData.head()
```

	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	...	texture55	texture56	texture57	texture58	texture59	texture60	textur
0	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	0.001953	...	0.007812	0.000000	0.002930	0.002930	0.035156	0.0	
1	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953	0.019531	0.0	0.000000	...	0.000977	0.000000	0.000000	0.000977	0.023438	0.0	
2	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859	0.068359	0.0	0.000000	...	0.154300	0.000000	0.005859	0.000977	0.007812	0.0	
3	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531	0.023438	0.0	0.013672	...	0.000000	0.000977	0.000000	0.000000	0.020508	0.0	
4	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625	0.005859	0.0	0.000000	...	0.096680	0.000000	0.021484	0.000000	0.000000	0.0	

5 rows × 193 columns

Figure 2: Running train data

As we saw after running the data it consists of **990rows × 193 columns**…

After reading train file we used python iloc() function that enables us to select a particular cell of the dataset, that is, it helps us select a value that belongs to a particular row or

column from a set of values of a data frame or dataset.

Then we use **head()** to display the first 5 rows from test data…

```python
#reading the test data from path
testData = pd.read_csv('/content/drive/MyDrive/leaf-classification/test.csv')

#reading the test data as all columns except the first one(id column)
testData = testData.iloc[:,1:]

#displaying the first 5 rows from test data
testData.head()
```

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	...	texture55	texture56	texture57	texture58	texture59	texture60	texture61	textu
0	0.019531	0.009766	0.078125	0.011719	0.003906	0.015625	0.005859	0.0	0.005859	0.023438	...	0.006836	0.000000	0.015625	0.000977	0.015625	0.0	0.0	0.00
1	0.007812	0.005859	0.064453	0.009766	0.003906	0.013672	0.007812	0.0	0.033203	0.023438	...	0.000000	0.000000	0.006836	0.001953	0.013672	0.0	0.0	0.00
2	0.000000	0.000000	0.001953	0.021484	0.041016	0.000000	0.023438	0.0	0.011719	0.005859	...	0.128910	0.000000	0.000977	0.000000	0.000000	0.0	0.0	0.01
3	0.000000	0.000000	0.009766	0.011719	0.017578	0.000000	0.003906	0.0	0.003906	0.001953	...	0.012695	0.015625	0.002930	0.036133	0.013672	0.0	0.0	0.08
4	0.001953	0.000000	0.015625	0.009766	0.039062	0.000000	0.009766	0.0	0.005859	0.000000	...	0.000000	0.042969	0.016602	0.010742	0.041016	0.0	0.0	0.00

5 rows × 192 columns

Figure 3: Running test data

```python
# to describe train data to know some information about it
trainData.describe()
```

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	...	texture55	texture56	texture57	texture58	texture59
count	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	...	990.000000	990.000000	990.000000	990.000000	990.000000
mean	0.017412	0.028539	0.031988	0.023280	0.014264	0.038579	0.019202	0.001083	0.007167	0.018639	...	0.036501	0.005024	0.015944	0.011586	0.016108
std	0.019739	0.038855	0.025847	0.028411	0.018390	0.052030	0.017511	0.002743	0.008933	0.016071	...	0.063403	0.019321	0.023214	0.025040	0.015335
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.000000	0.001953	0.005859	...	0.000000	0.000000	0.000977	0.000000	0.004883
50%	0.009766	0.011719	0.025391	0.013672	0.007812	0.015625	0.015625	0.000000	0.005859	0.015625	...	0.004883	0.000000	0.005859	0.000977	0.012695
75%	0.025391	0.041016	0.044922	0.029297	0.017578	0.056153	0.029297	0.000000	0.007812	0.027344	...	0.043701	0.000000	0.022217	0.009766	0.021484
max	0.087891	0.205080	0.156250	0.169920	0.111330	0.310550	0.091797	0.031250	0.076172	0.097656	...	0.429690	0.202150	0.172850	0.200200	0.106450

8 rows × 192 columns

Figure 4: describe train data

```python
# to know some information about train data
trainData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 990 entries, 0 to 989
Columns: 193 entries, species to texture64
dtypes: float64(192), object(1)
memory usage: 1.5+ MB
```

Figure 6: info for train data

```
[ ]  #check null values in train data
     trainData.isnull().values.any()

     False

[ ]  #check null values in test data
     testData.isnull().values.any()

     False
```

Figure 9: check null values in train and test data

Observation: There is nomissing values in train & test data

```
# to check if our train data is duplicated or not
trainData.duplicated().sum()

0

[ ] # to check if our test data is duplicated or not
testData.duplicated().sum()

0
```

Observation: There is no duplicated data

Figure 10: observation to check if the data duplicated or not

```
[ ]  # to know the value_counts in species feature
     trainData["species"].value_counts()

     Acer_Opalus                    10
     Crataegus_Monogyna             10
     Acer_Mono                      10
     Magnolia_Heptapeta             10
     Acer_Capillipes                10
                                    ..
     Alnus_Rubra                    10
     Rhododendron_x_Russellianum    10
     Cytisus_Battandieri            10
     Liriodendron_Tulipifera        10
     Sorbus_Aria                    10
     Name: species, Length: 99, dtype: int64
```

Figure 11: value_counts in species feature

4. Visualizing dataset…

One of the most important skills in data science is data visualisation. We need to understand the underlying dataset before we can start creating viable models. You'll never be an expert on the data you're dealing with, and you'll always need to go deep into the variables before moving on to developing a model or doing something else with it. The most crucial tool in your arsenal for accomplishing this is effective data visualisation.

```
[ ] #read some of images in size (20,15)
    os.chdir("/content/drive/MyDrive/leaf-classification/images")
    plt.figure(figsize=(20,15))
    for i in range(5):
        w=np.random.choice((os.listdir()))
        plt.subplot(5,5,i+1)
        img=load_img(os.path.join('/content/drive/MyDrive/leaf-classification/images',w))
        plt.imshow(img)
        plt.imshow(img)
```

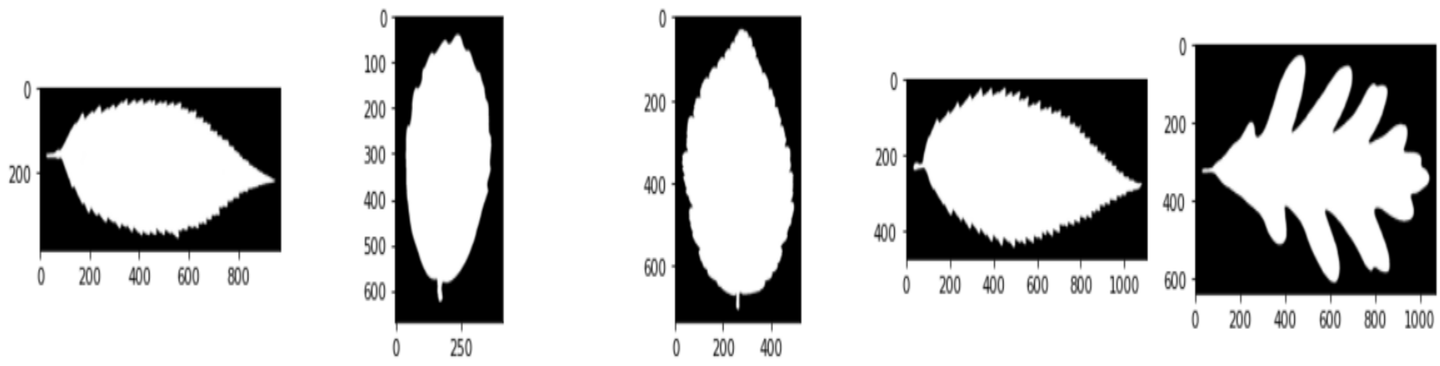

Figure 12: showing some of images from the image file

```
corr=trainData.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Figure 7: visualize the data that show the correlation between features

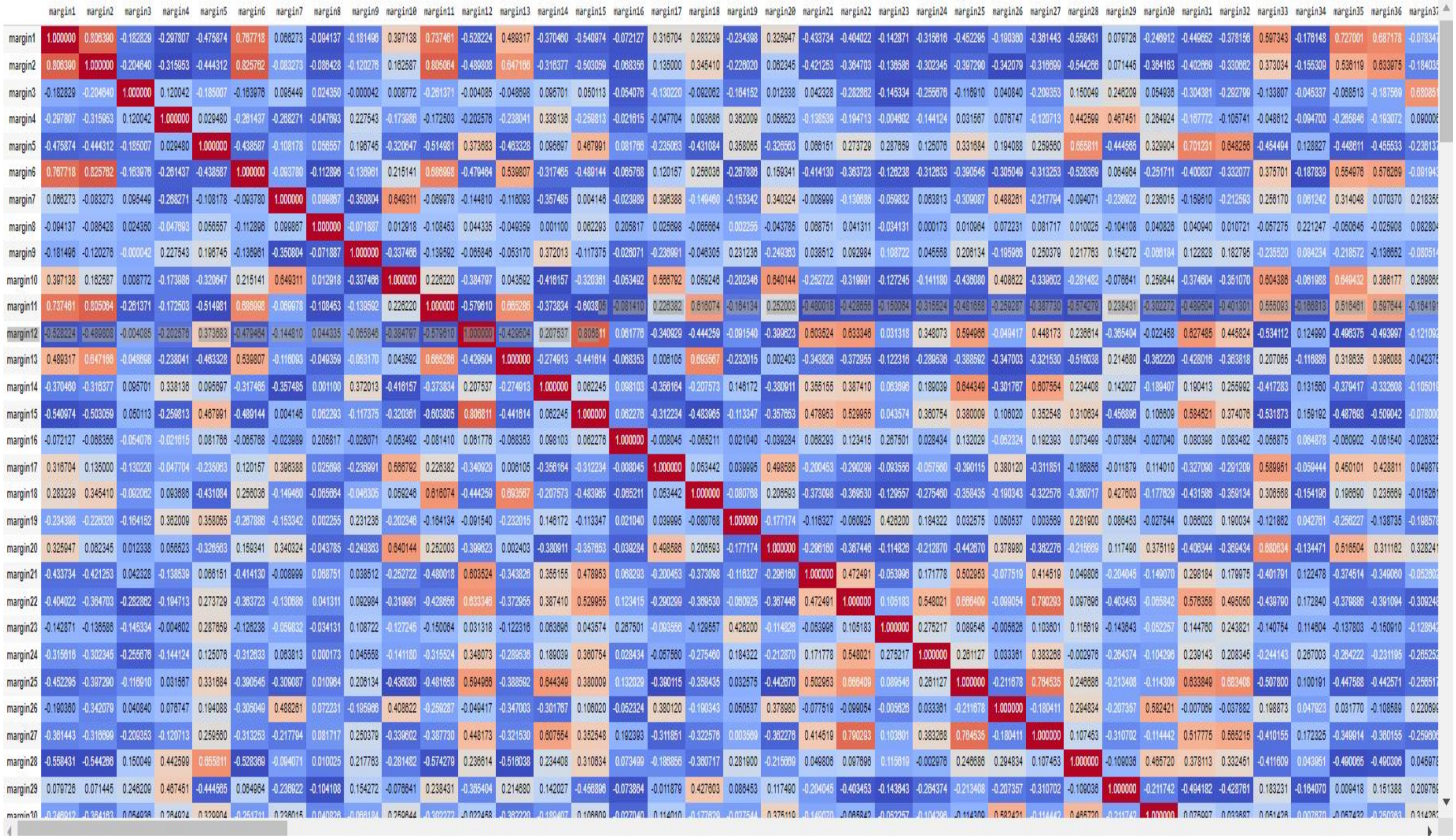

Figure 8: correlation matrix for the train data

```
[ ] #plot correlation between columns
    plt.figure(figsize=(15,7))
    sns.heatmap(data=trainData.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f431f378850>
```

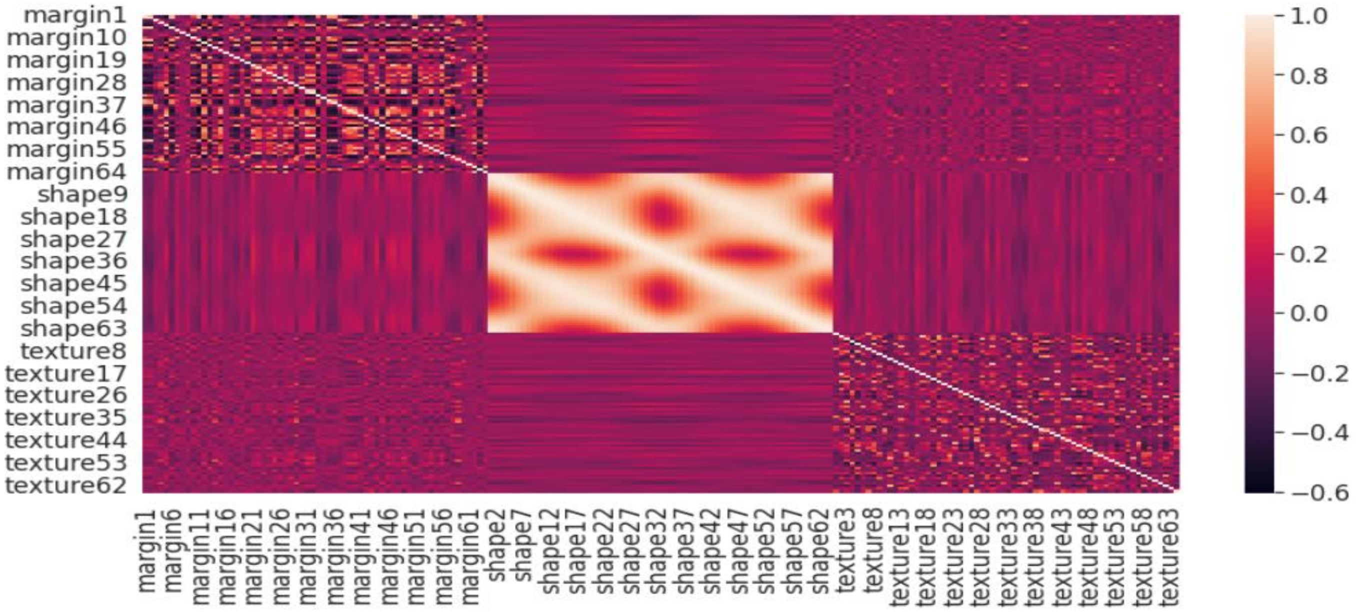

Figure 13: plot correlation between columns

5. Splitting data to x , y…

split the data to input and output as we see x is all the data without spaces feature and use shuffle for data to prevent overfitting bu change values with each other…

and y is considered as output…

```
# change the data with each other to prevent the overfitting
trainData = shuffle(trainData)
```

```
# to assign y and x to values
trainData = trainData.values
X= trainData[:,1:]
y = trainData[:,0:1]
```

```
[ ] #shape of training data
    trainData.shape

(990, 193)
```

```
[ ] #shape of training data features
    X.shape

(990, 192)
```

```
[ ] ##shape of training data label
    y.shape

(990, 1)
```

```
[ ] #number of unique values in label column
    trainData['species'].nunique()

99
```

```
encoder = LabelEncoder()
y_fit = encoder.fit(trainData['species'])
y_label = y_fit.transform(trainData['species'])
classes = list(y_fit.classes_)
classes
```

```
 'Magnolia_Salicifolia',
 'Morus_Nigra',
 'Olea_Europaea',
 'Phildelphus',
 'Populus_Adenopoda',
 'Populus_Grandidentata',
 'Populus_Nigra',
 'Prunus_Avium',
 'Prunus_X_Shmittii',
 'Pterocarya_Stenoptera',
 'Quercus_Afares',
 'Quercus_Agrifolia',
 'Quercus_Alnifolia',
 'Quercus_Brantii',
 'Quercus_Canariensis',
 'Quercus_Castaneifolia',
 'Quercus_Cerris',
 'Quercus_Chrysolepis',
 'Quercus_Coccifera',
 'Quercus_Coccinea',
 'Quercus_Crassifolia',
 'Quercus_Crassipes',
 'Quercus_Dolicholepis',
```

Figure 14: Label Encoder

6. Split to train & validation data…

Split to train & validation data

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

Figure 15: Split to train & validation data

By scikit-learn library we used the train-test split evaluation procedure via the train_test_split() function, that takes a loaded dataset as input and returns the dataset split into two subsets(train and test subsets).

We split the data into 20% for the testing and 80% for the training.

7. Scaling...

```
[ ] scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    test_Data=scaler.transform(testData)

    /usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: UserWarning: X has feature names, but StandardScaler was fitted without feature names
      f"X has feature names, but {self.__class__.__name__} was fitted without"
```

Figure 16: scaling the data

8. Model...

```
[ ] # Part 2 - Now let's make the ANN!
    import tensorflow
    # Importing the Keras libraries and packages
    from keras.models import Sequential  #to initialize the neural network
    from keras.layers import Dense  # to build the layers of ANN
    from keras.layers import Dropout
    import keras
```

Figure 17: useful import for models

Training function of choose optimizer:

```
from keras import regularizers
from keras.callbacks import EarlyStopping

#Trining function
def training(optemizer):

  # structure model
  features= X_train.shape[1]
  model = Sequential()

  model.add(Dense(units = 512, activation = 'tanh', input_shape=(features,)))


  model.add(Dense(units=99, activation = 'softmax'))

  # Compiling the ANN
  early_stop = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.001)
  model.compile(optemizer, loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

  # Fitting the ANN to the Training set
  history= model.fit(X_train, y_train,validation_data=(X_val, y_val), batch_size = 32, epochs = 100)

  return model,history
```

Figure 18: choose optimizer

Evaluation function

```
#evaluation function
def evaluate(model,x,y):
  loss,acc=model.evaluate(x,y)
  print("\n loss= ",loss)
  print("\n Accuracy= ",acc)
```

Figure 19: evaluation functions

8.1 Some trials using different hyperparameters

8.1.1 Optimizer traial_1

Trial_1 (Adam)

```
[7] model,history= training("adam")
    model
```

```
Epoch 1/100
25/25 [==============================] - 1s 17ms/step - loss: 3.1638 - accuracy: 0.3687 - val_loss: 1.8078 - val_accuracy: 0.7071
Epoch 2/100
25/25 [==============================] - 0s 5ms/step - loss: 0.8578 - accuracy: 0.9331 - val_loss: 0.7559 - val_accuracy: 0.9343
Epoch 3/100
25/25 [==============================] - 0s 6ms/step - loss: 0.3209 - accuracy: 0.9861 - val_loss: 0.4215 - val_accuracy: 0.9545
Epoch 4/100
25/25 [==============================] - 0s 5ms/step - loss: 0.1717 - accuracy: 0.9937 - val_loss: 0.3409 - val_accuracy: 0.9444
Epoch 5/100
25/25 [==============================] - 0s 6ms/step - loss: 0.1094 - accuracy: 0.9975 - val_loss: 0.2765 - val_accuracy: 0.9545
Epoch 6/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0779 - accuracy: 0.9975 - val_loss: 0.2522 - val_accuracy: 0.9545
Epoch 7/100
25/25 [==============================] - 0s 5ms/step - loss: 0.0594 - accuracy: 1.0000 - val_loss: 0.2190 - val_accuracy: 0.9646
Epoch 8/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0470 - accuracy: 1.0000 - val_loss: 0.2048 - val_accuracy: 0.9646
Epoch 9/100
25/25 [==============================] - 0s 7ms/step - loss: 0.0393 - accuracy: 1.0000 - val_loss: 0.1921 - val_accuracy: 0.9646
Epoch 10/100
25/25 [==============================] - 0s 7ms/step - loss: 0.0319 - accuracy: 1.0000 - val_loss: 0.1781 - val_accuracy: 0.9646
Epoch 11/100
25/25 [==============================] - 0s 5ms/step - loss: 0.0272 - accuracy: 1.0000 - val_loss: 0.1730 - val_accuracy: 0.9646
Epoch 12/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0234 - accuracy: 1.0000 - val_loss: 0.1637 - val_accuracy: 0.9646
Epoch 13/100
25/25 [==============================] - 0s 5ms/step - loss: 0.0206 - accuracy: 1.0000 - val_loss: 0.1582 - val_accuracy: 0.9646
Epoch 14/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0182 - accuracy: 1.0000 - val_loss: 0.1522 - val_accuracy: 0.9646
Epoch 15/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0162 - accuracy: 1.0000 - val_loss: 0.1500 - val_accuracy: 0.9646
Epoch 16/100
25/25 [==============================] - 0s 5ms/step - loss: 0.0146 - accuracy: 1.0000 - val_loss: 0.1438 - val_accuracy: 0.9646
```

Figure 20: Optimizer traial_1 (Adam)

```
[ ] evaluate(model,X_train,y_train)

    25/25 [==============================] - 0s 3ms/step - loss: 5.3341e-04 - accuracy: 1.0000

     loss=  0.0005334122106432915

     Accuracy=  1.0

[ ] evaluate(model,X_val,y_val)

    7/7 [==============================] - 0s 3ms/step - loss: 0.0980 - accuracy: 0.9747

     loss=  0.09799925237894058

     Accuracy=  0.9747474789619446

[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

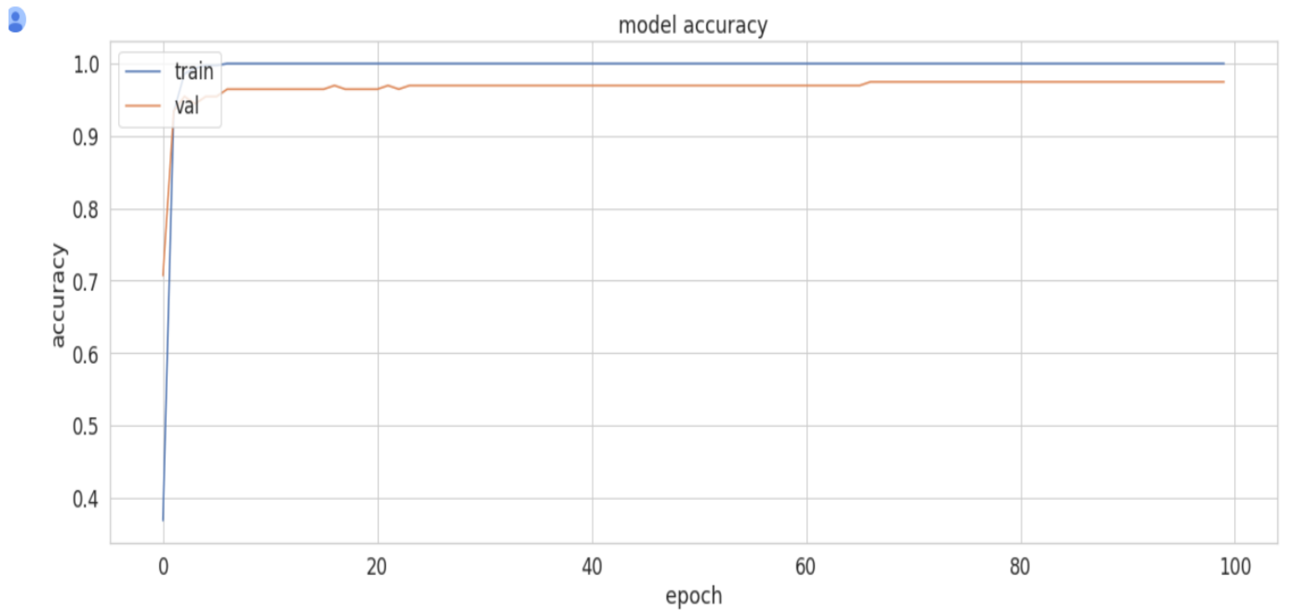

Figure 21: accuracy plot for Adam Optimizer traial_1

```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

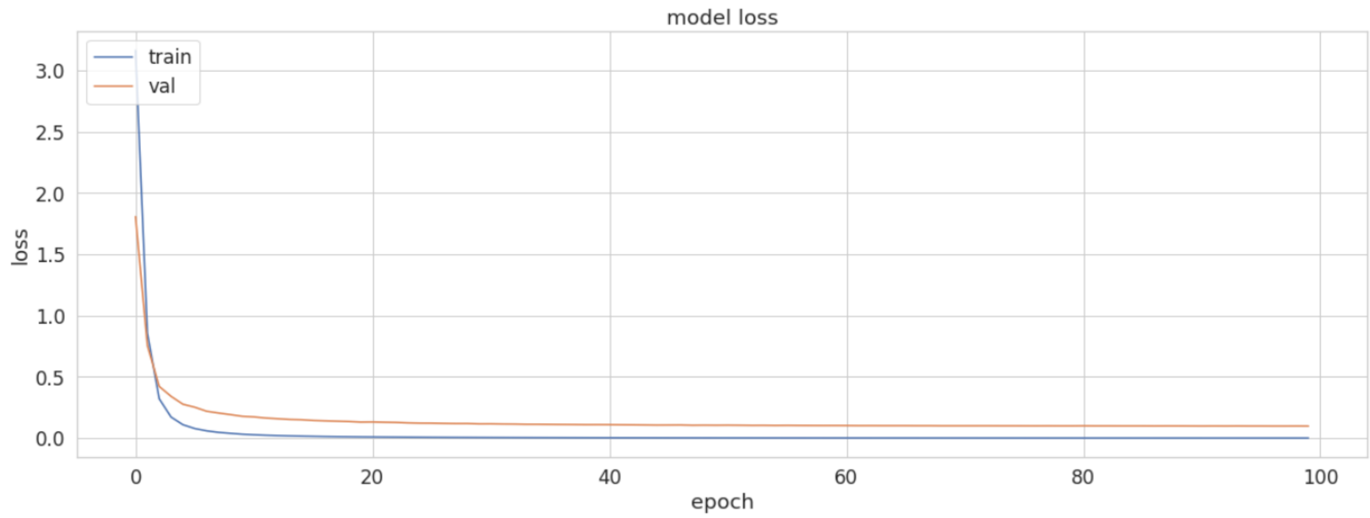

Figure 22: loss plot for Adam Optimizer traial_1

8.1.2 Optimizer traial_2

Trial_2 (SGD)

```
[ ] model,history= training("SGD")
```

```
Epoch 1/100
25/25 [==============================] - 1s 12ms/step - loss: 4.5821 - accuracy: 0.0114 - val_loss: 4.2909 - val_accuracy: 0.040
Epoch 2/100
25/25 [==============================] - 0s 5ms/step - loss: 4.0129 - accuracy: 0.0833 - val_loss: 3.8507 - val_accuracy: 0.1465
Epoch 3/100
25/25 [==============================] - 0s 6ms/step - loss: 3.5196 - accuracy: 0.2525 - val_loss: 3.4697 - val_accuracy: 0.2879
Epoch 4/100
25/25 [==============================] - 0s 6ms/step - loss: 3.0914 - accuracy: 0.4419 - val_loss: 3.1369 - val_accuracy: 0.4242
Epoch 5/100
25/25 [==============================] - 0s 5ms/step - loss: 2.7177 - accuracy: 0.6174 - val_loss: 2.8450 - val_accuracy: 0.5101
Epoch 6/100
25/25 [==============================] - 0s 5ms/step - loss: 2.3961 - accuracy: 0.7096 - val_loss: 2.5892 - val_accuracy: 0.5960
Epoch 7/100
25/25 [==============================] - 0s 5ms/step - loss: 2.1188 - accuracy: 0.7992 - val_loss: 2.3638 - val_accuracy: 0.6667
Epoch 8/100
25/25 [==============================] - 0s 5ms/step - loss: 1.8815 - accuracy: 0.8624 - val_loss: 2.1660 - val_accuracy: 0.7374
Epoch 9/100
25/25 [==============================] - 0s 5ms/step - loss: 1.6801 - accuracy: 0.8927 - val_loss: 1.9872 - val_accuracy: 0.7626
Epoch 10/100
25/25 [==============================] - 0s 6ms/step - loss: 1.5029 - accuracy: 0.9129 - val_loss: 1.8311 - val_accuracy: 0.7677
Epoch 11/100
25/25 [==============================] - 0s 6ms/step - loss: 1.3535 - accuracy: 0.9369 - val_loss: 1.6922 - val_accuracy: 0.7828
Epoch 12/100
25/25 [==============================] - 0s 5ms/step - loss: 1.2249 - accuracy: 0.9482 - val_loss: 1.5691 - val_accuracy: 0.8131
Epoch 13/100
25/25 [==============================] - 0s 5ms/step - loss: 1.1125 - accuracy: 0.9571 - val_loss: 1.4603 - val_accuracy: 0.8232
Epoch 14/100
25/25 [==============================] - 0s 5ms/step - loss: 1.0152 - accuracy: 0.9646 - val_loss: 1.3616 - val_accuracy: 0.8434
Epoch 15/100
25/25 [==============================] - 0s 5ms/step - loss: 0.9301 - accuracy: 0.9710 - val_loss: 1.2743 - val_accuracy: 0.8687
Epoch 16/100
25/25 [==============================] - 0s 5ms/step - loss: 0.8560 - accuracy: 0.9773 - val_loss: 1.1960 - val_accuracy: 0.8838
```

Figure 23: Optimizer traial_2 (SGD)

```
evaluate(model,X_train,y_train)

25/25 [==============================] - 0s 3ms/step - loss: 0.0795 - accuracy: 1.0000
 loss=  0.07948926836252213
 Accuracy=  1.0
```

```
evaluate(model,X_val,y_val)

7/7 [==============================] - 0s 3ms/step - loss: 0.2578 - accuracy: 0.9697
 loss=  0.2577986717224121
 Accuracy=  0.9696969985961914
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

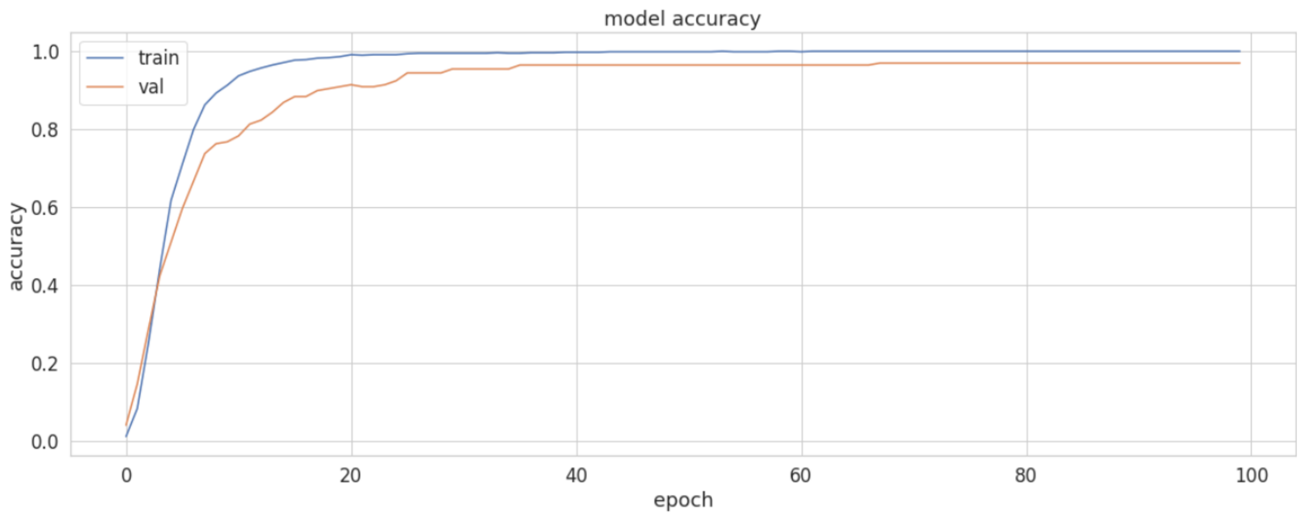

Figure 24: accuracy plot for SGD Optimizer traial_2

```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

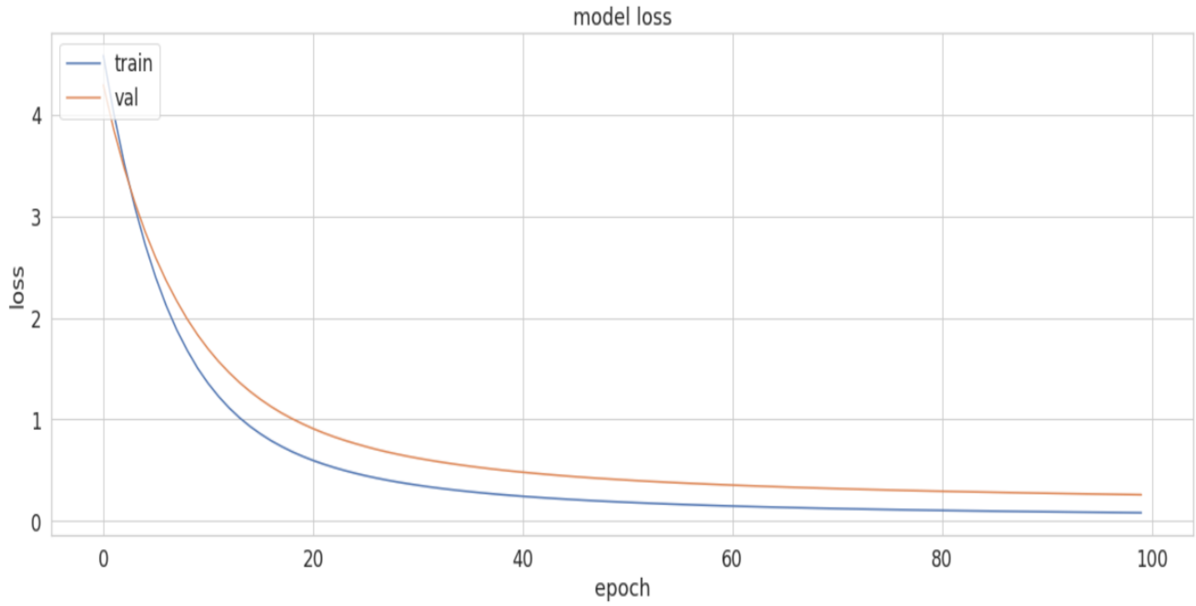

Figure 25: loss plot for SGD Optimizer traial_2

8.1.3 Optimizer traial_3

Trial_3 (RMSProp)

```
model,history= training("RMSProp")
```

```
Epoch 1/100
25/25 [==============================] - 1s 14ms/step - loss: 2.6067 - accuracy: 0.5455 - val_loss: 1.4478 - val_accuracy: 0.8081
Epoch 2/100
25/25 [==============================] - 0s 6ms/step - loss: 0.7290 - accuracy: 0.9470 - val_loss: 0.6551 - val_accuracy: 0.9242
Epoch 3/100
25/25 [==============================] - 0s 6ms/step - loss: 0.2936 - accuracy: 0.9811 - val_loss: 0.3510 - val_accuracy: 0.9495
Epoch 4/100
25/25 [==============================] - 0s 6ms/step - loss: 0.1304 - accuracy: 0.9937 - val_loss: 0.2449 - val_accuracy: 0.9697
Epoch 5/100
25/25 [==============================] - 0s 7ms/step - loss: 0.0682 - accuracy: 0.9962 - val_loss: 0.2139 - val_accuracy: 0.9495
Epoch 6/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0348 - accuracy: 0.9975 - val_loss: 0.1588 - val_accuracy: 0.9646
Epoch 7/100
25/25 [==============================] - 0s 7ms/step - loss: 0.0194 - accuracy: 0.9987 - val_loss: 0.1403 - val_accuracy: 0.9596
Epoch 8/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0114 - accuracy: 0.9987 - val_loss: 0.1306 - val_accuracy: 0.9646
Epoch 9/100
25/25 [==============================] - 0s 7ms/step - loss: 0.0070 - accuracy: 0.9975 - val_loss: 0.1081 - val_accuracy: 0.9747
Epoch 10/100
25/25 [==============================] - 0s 7ms/step - loss: 0.0032 - accuracy: 1.0000 - val_loss: 0.1078 - val_accuracy: 0.9646
Epoch 11/100
25/25 [==============================] - 0s 7ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 0.1495 - val_accuracy: 0.9596
Epoch 12/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.1098 - val_accuracy: 0.9697
Epoch 13/100
25/25 [==============================] - 0s 6ms/step - loss: 6.2847e-04 - accuracy: 1.0000 - val_loss: 0.1248 - val_accuracy: 0.9646
Epoch 14/100
25/25 [==============================] - 0s 7ms/step - loss: 3.2498e-04 - accuracy: 1.0000 - val_loss: 0.0936 - val_accuracy: 0.9747
Epoch 15/100
```

Figure 26: Optimizer traial_3 (RMSProp)

```
evaluate(model,X_train,y_train)
```

```
25/25 [==============================] - 0s 3ms/step - loss: 3.7027e-08 - accuracy: 1.0000
 loss=  3.702712447761769e-08
 Accuracy=  1.0
```

```
[ ] evaluate(model,X_val,y_val)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.1434 - accuracy: 0.9798
 loss=  0.14340300858020782
 Accuracy=  0.9797979593276978
```

```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

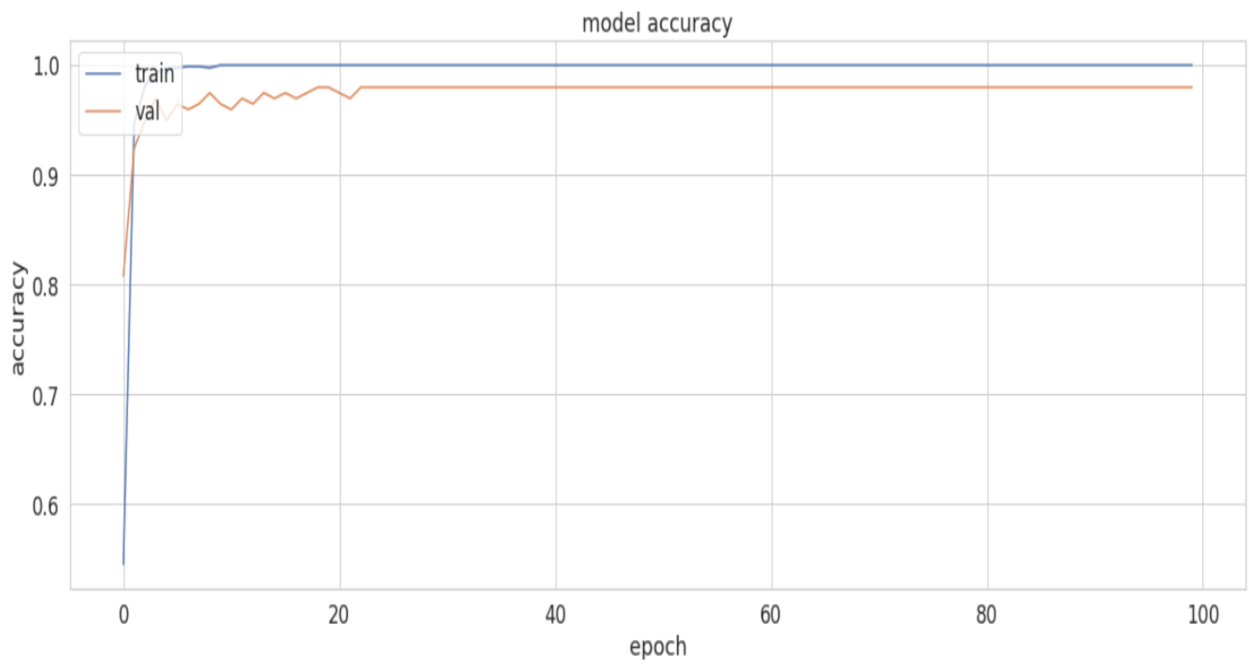

Figure 27: accuracy plot for RMSProp Optimizer traial_3

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

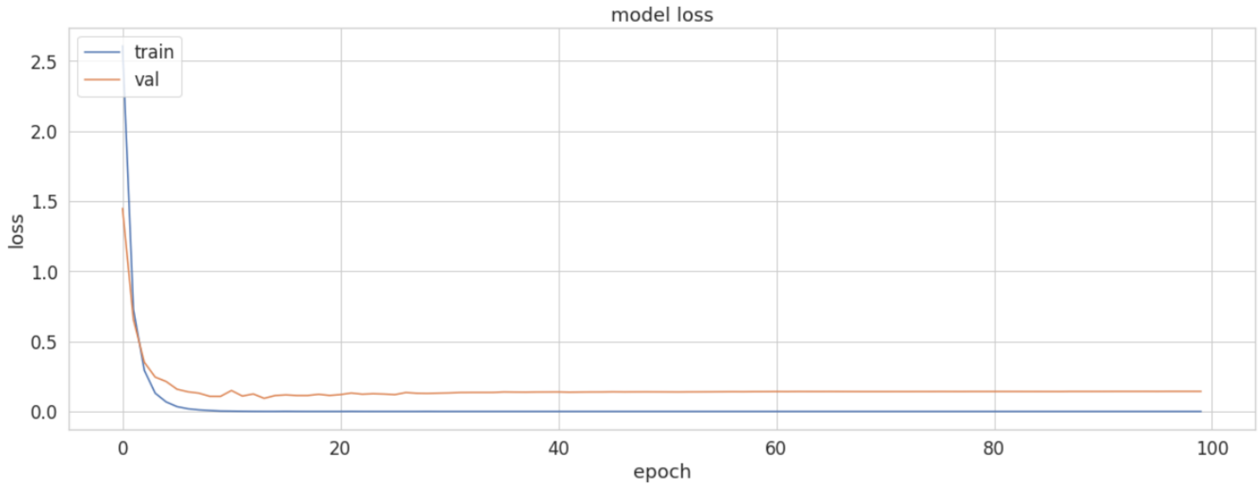

Figure 28: loss plot for RMSProp Optimizer trail_3

Observation: From the previous trials, we discovered that **Adam** optimizer is the best optimizer

8.2 Model learning rate

Learning rate

```
[ ] from keras import regularizers
    from keras.callbacks import EarlyStopping
    def training(lr):

      # structure model
      features= X_train.shape[1]
      model = Sequential()

      model.add(Dense(units = 512, activation = 'tanh', input_shape=(features,)))
      # model.add(Dropout(0.1))


      model.add(Dense(units=99, activation = 'softmax'))

      # Compiling the ANN
      early_stop = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.001)
      opt = tensorflow.keras.optimizers.Adam(lr)
      model.compile(opt, loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

      # Fitting the ANN to the Training set
      history= model.fit(X_train, y_train,validation_data=(X_val, y_val), batch_size = 32, epochs = 100,verbose=0)

      return model,history
```

Figure 29: Model learning rate

8.2.1 learning rate traial_4

```
model,history= training(.1)
```

```
[ ] evaluate(model,X_train,y_train)

25/25 [==============================] - 0s 3ms/step - loss: 2.4922 - accuracy: 0.9495
 loss=  2.4921934604644775
 Accuracy=  0.9494949579238892
```

```
[ ] evaluate(model,X_val,y_val)

7/7 [==============================] - 0s 3ms/step - loss: 17.2739 - accuracy: 0.8283
 loss=  17.273889541625977
 Accuracy=  0.8282828330993652
```

```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

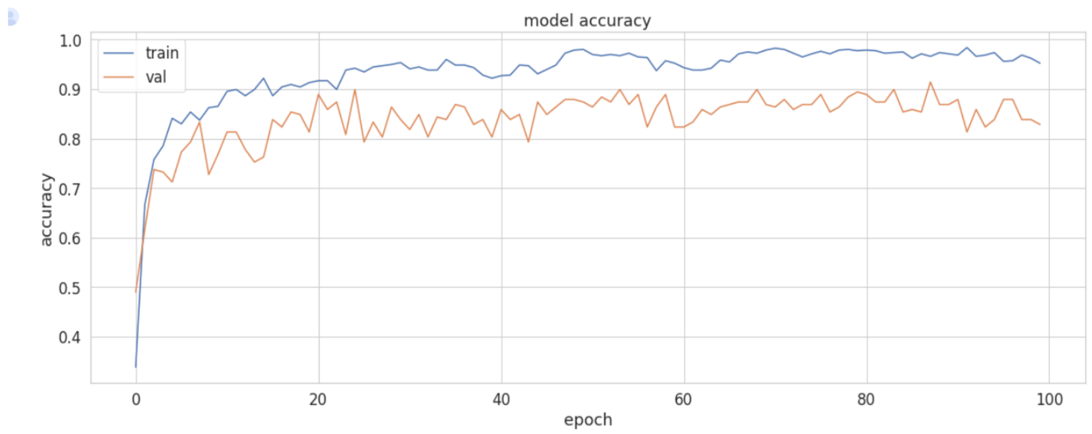

Figure 30: accuracy plot for Learning rate Optimizer traial_4

```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

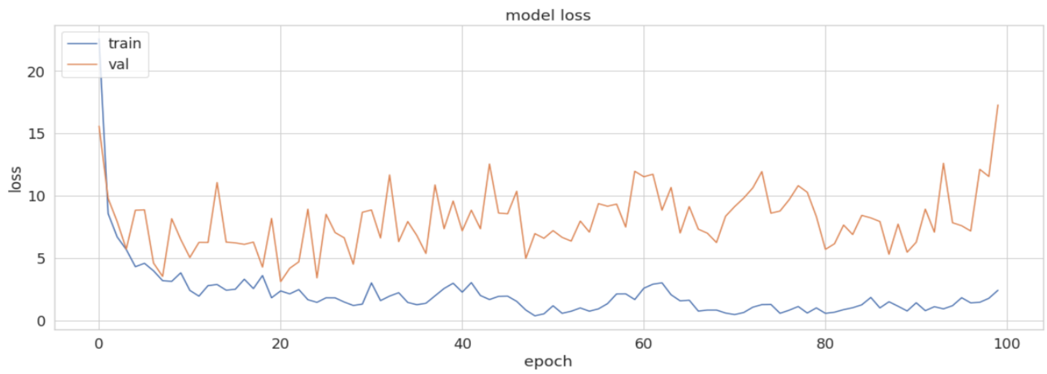

Figure 31: loss plot for Learning rate Optimizer traial_4

8.2.2 learning rate traial_5

```
[ ]  model,history= training(0.01)

[ ]  evaluate(model,X_train,y_train)

     25/25 [==============================] - 0s 3ms/step - loss: 1.6623e-05 - accuracy: 1.0000

      loss=  1.6623000192339532e-05

      Accuracy=  1.0

[ ]  evaluate(model,X_val,y_val)

     7/7 [==============================] - 0s 6ms/step - loss: 0.1565 - accuracy: 0.9747

      loss=  0.1565122753381729

      Accuracy=  0.9747474789619446

[ ]  plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'val'], loc='upper left')
     plt.show()
```

Figure 32: Learning rate Optimizer traial_5

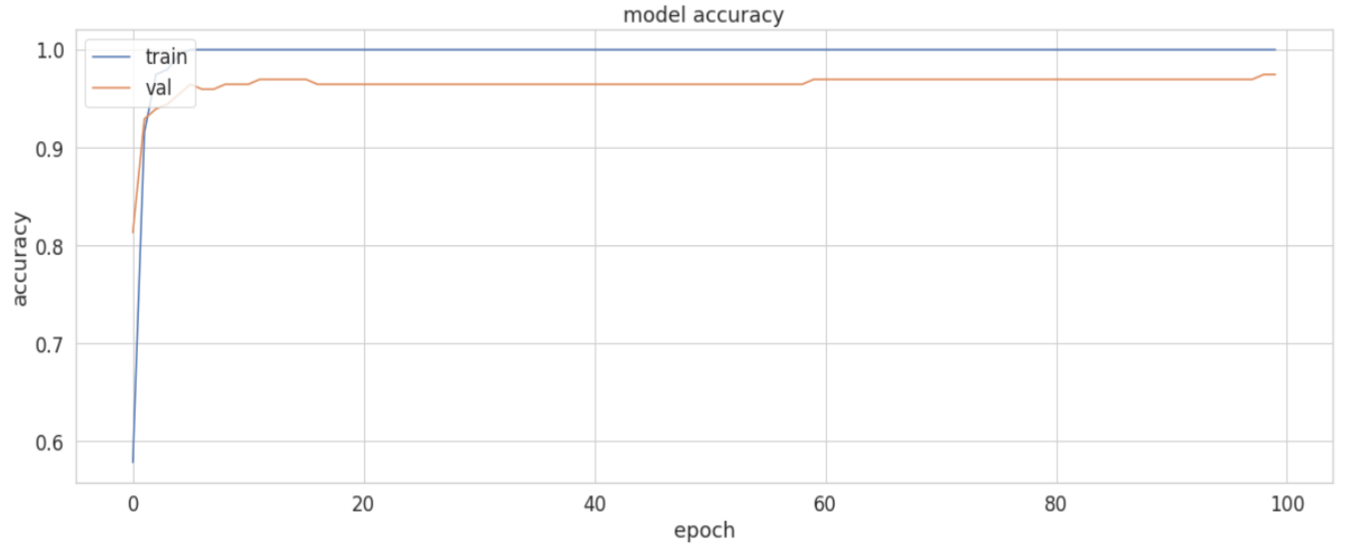

Figure 33: accuracy plot for Learning rate Optimizer traial_5

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

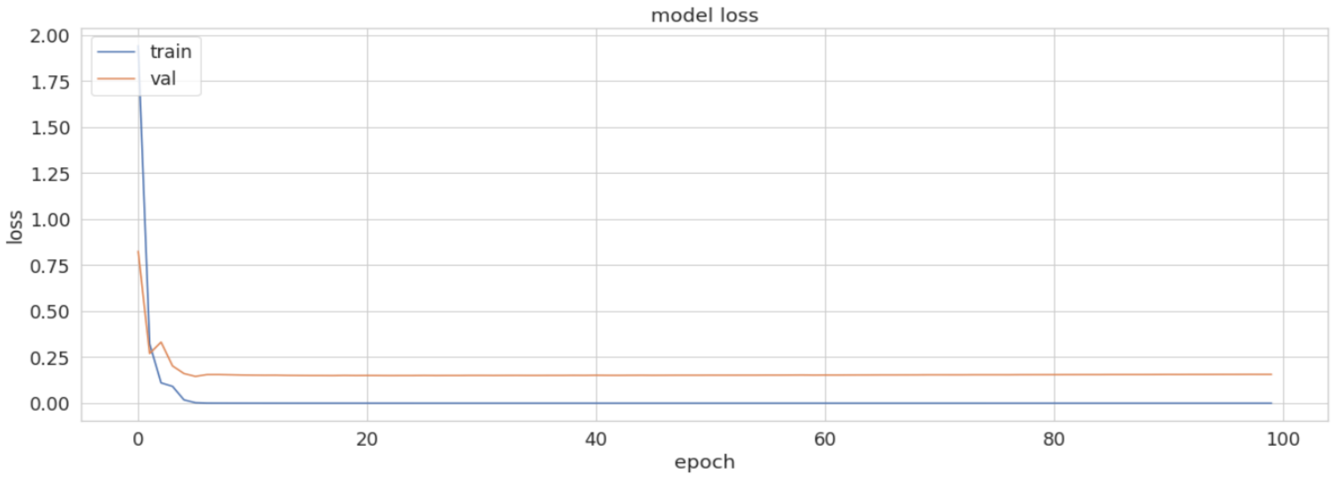

Figure 34: loss plot for Learning rate Optimizer traial_5

8.2.3 learning rate traial_6

Trial_6(0.001)

```
[ ] model,history= training(0.001)
```

```
[ ] evaluate(model,X_train,y_train)
```

```
25/25 [==============================] - 0s 3ms/step - loss: 5.3141e-04 - accuracy: 1.0000

 loss=  0.0005314061418175697

 Accuracy=  1.0
```

```
[ ] evaluate(model,X_val,y_val)
```

```
7/7 [==============================] - 0s 4ms/step - loss: 0.0949 - accuracy: 0.9798

 loss=  0.09492850303649902

 Accuracy=  0.9797979593276978
```

```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

Figure 34: Learning rate Optimizer traial_6

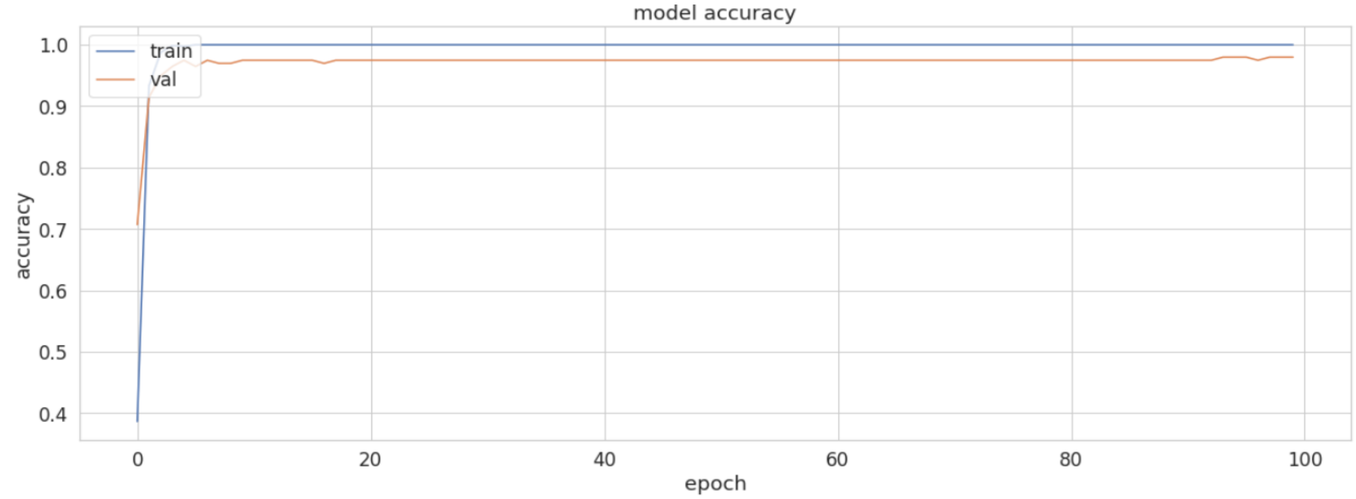

Figure 35: accuracy plot for Learning rate Optimizer traial_6

```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

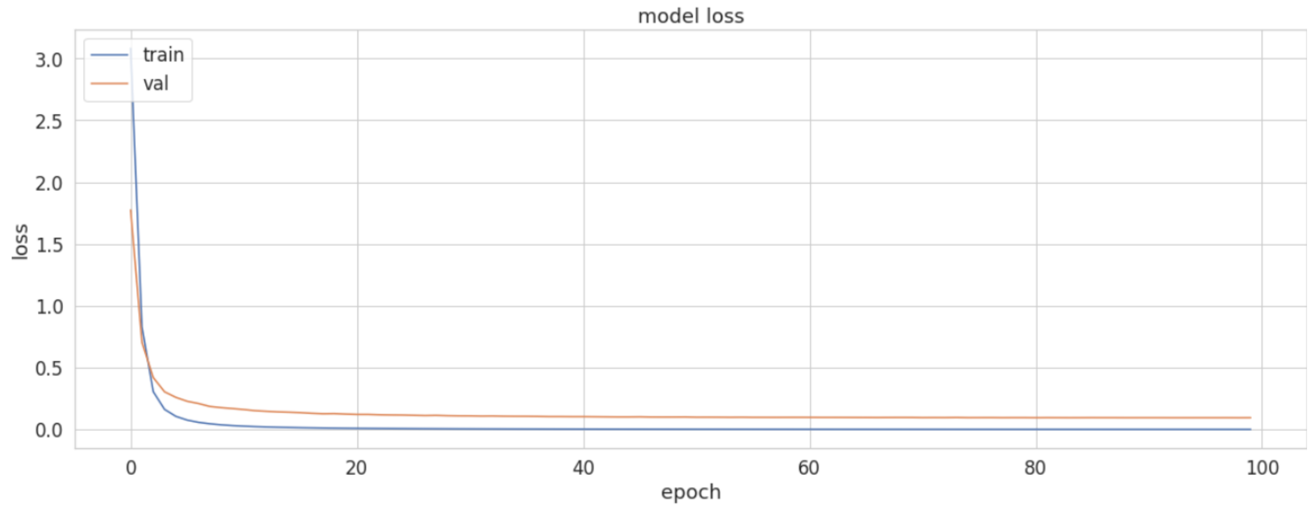

Figure 36: loss plot for Learning rate Optimizer traial_6

8.3 Model Batch_size

Batch_size

```
[ ] from keras import regularizers
    from keras.callbacks import EarlyStopping
    def training(batch):

      # structure model
      features= X_train.shape[1]
      model = Sequential()

      model.add(Dense(units = 512, activation = 'tanh', input_shape=(features,)))
      # model.add(Dropout(0.1))


      model.add(Dense(units=99, activation = 'softmax'))

      # Compiling the ANN
      early_stop = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.001)
      opt = tensorflow.keras.optimizers.Adam(0.001)
      model.compile(opt, loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

      # Fitting the ANN to the Training set
      history= model.fit(X_train, y_train,validation_data=(X_val, y_val), batch_size = batch, epochs = 100,verbose=0)

      return model,history
```

Figure 37: Model Batch_size

8.3.1 Batch_size traial_7

Trial_7 (16)

```
[ ] model,history= training(16)
```

```
[ ] evaluate(model,X_train,y_train)
```

```
25/25 [==============================] - 0s 3ms/step - loss: 1.6695e-04 - accuracy: 1.0000

 loss=  0.00016694940859451 89

 Accuracy=  1.0
```

```
[ ] evaluate(model,X_val,y_val)
```

```
7/7 [==============================] - 0s 5ms/step - loss: 0.0914 - accuracy: 0.9747

 loss=  0.09139124304056168

 Accuracy=  0.9747474789619446
```

```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

Figure 38: Batch_size Optimizer learning rate traial_7

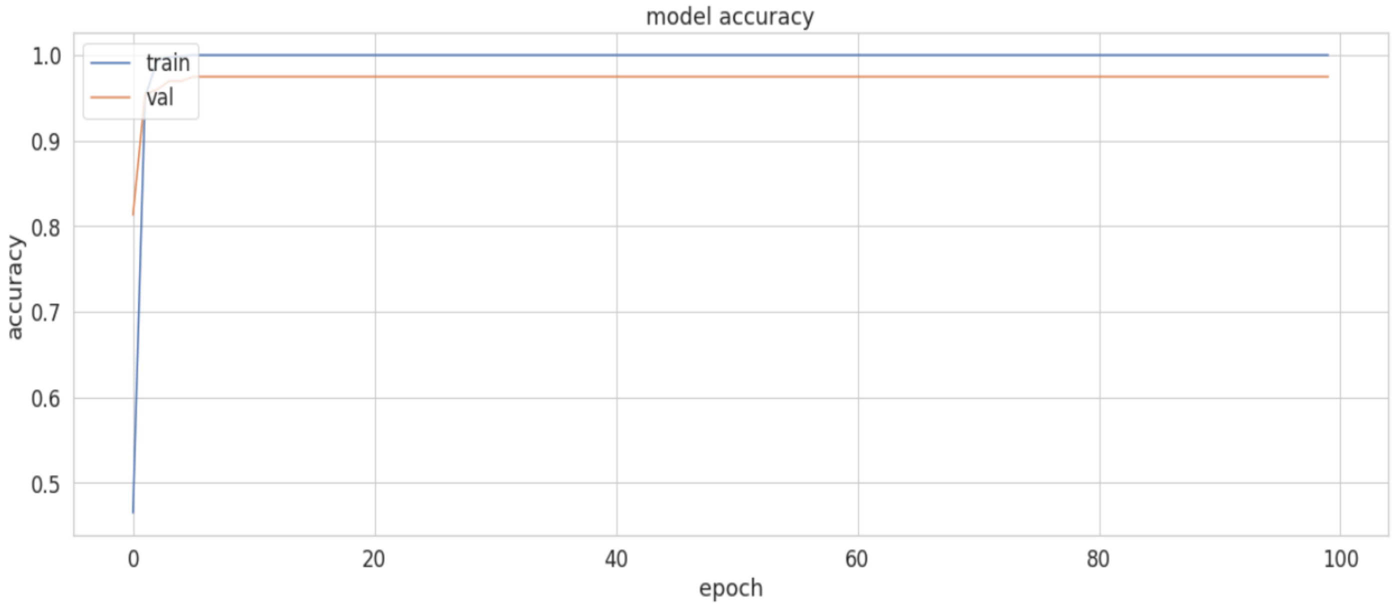

Figure 39: accuracy plot for Batch_size Optimizer learning rate traial_7

```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

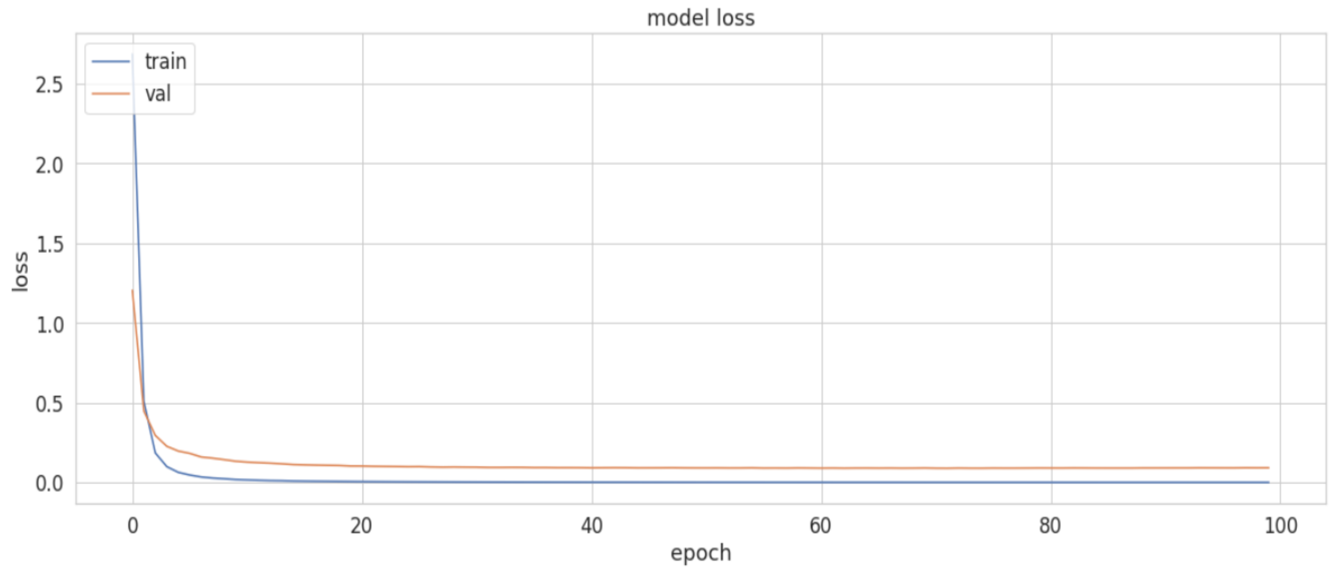

Figure 40: loss plot for Batch_size Optimizer learning rate traial_7

8.3.Batch_size traial_8

Trial_8(32)

```
[ ] model,history= training(32)
```

```
[ ] evaluate(model,X_train,y_train)
```

```
25/25 [==============================] - 0s 3ms/step - loss: 5.3600e-04 - accuracy: 1.0000
 loss=  0.0005360028007999063
 Accuracy=  1.0
```

```
[ ] evaluate(model,X_val,y_val)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.0891 - accuracy: 0.9747
 loss=  0.08907036483287811
 Accuracy=  0.9747474789619446
```

```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

Figure 41: Batch_size Optimizer learning rate traial_8

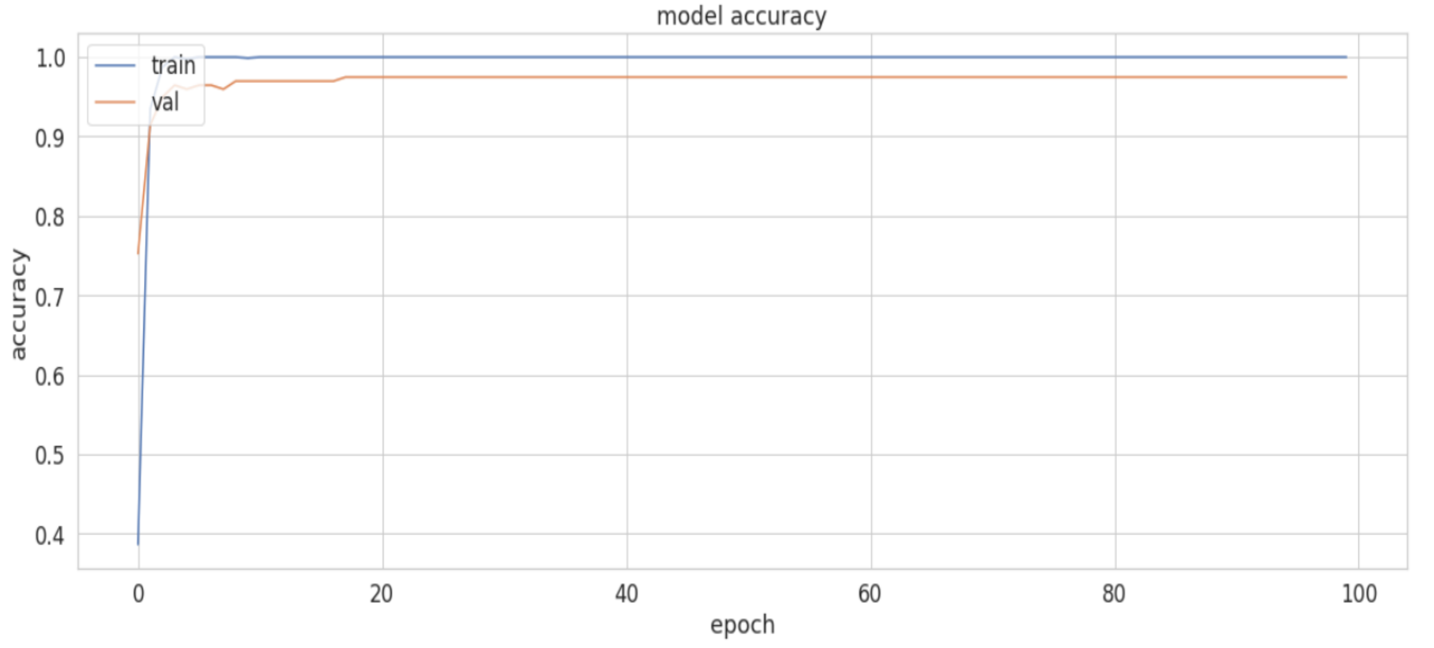

Figure 42: accuracy plot for Batch_size Optimizer learning rate traial_8

```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

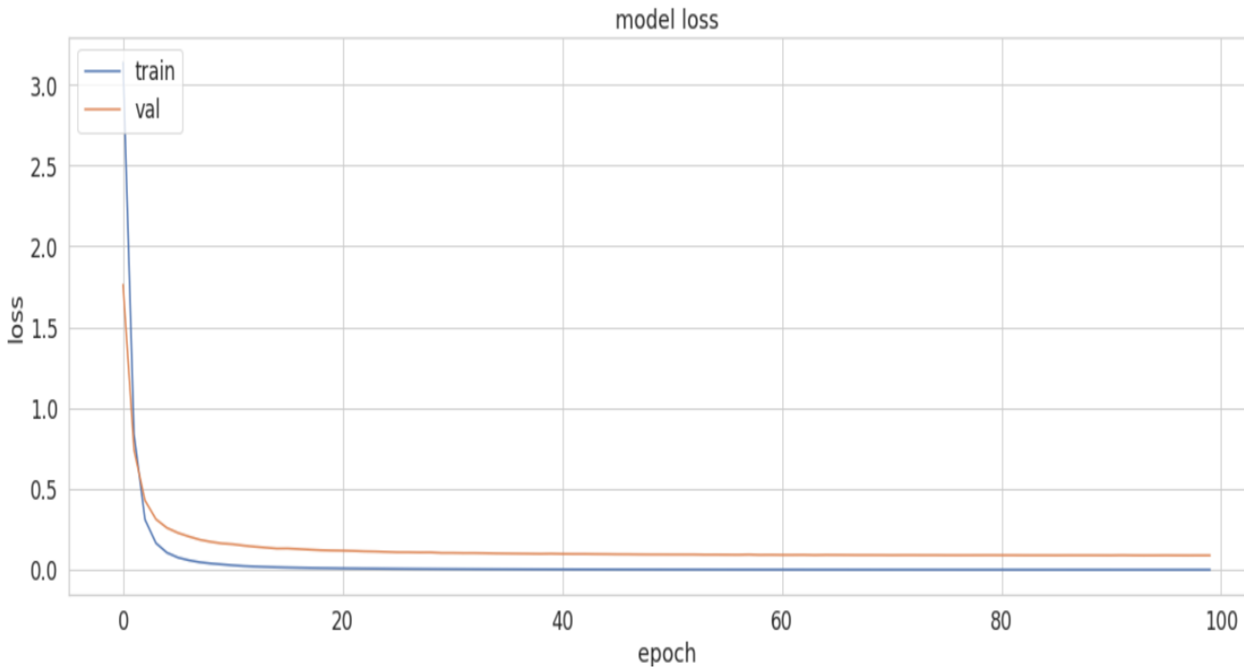

Figure 43: loss plot for Batch_size Optimizer learning rate traial_8

8.2.3 Batch_size traial_9

Trial_9(64)

```
[ ] model,history= training(64)
```

```
[ ] evaluate(model,X_train,y_train)

25/25 [==============================] - 0s 2ms/step - loss: 0.0012 - accuracy: 1.0000
 loss=  0.0012065600603818893
 Accuracy=  1.0
```

```
[ ] evaluate(model,X_val,y_val)

7/7 [==============================] - 0s 3ms/step - loss: 0.0897 - accuracy: 0.9798
 loss=  0.08972612023353577
 Accuracy=  0.9797979593276978
```

```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

Figure 44: Batch_size Optimizer learning rate traial_9

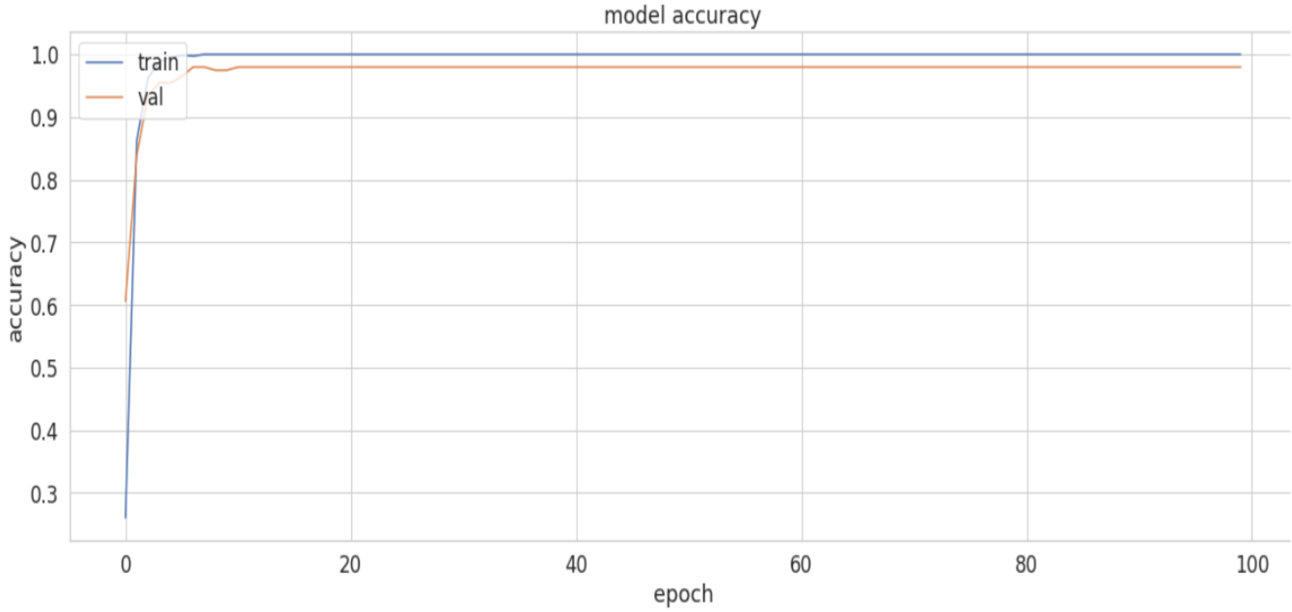

Figure 45: accuracy plot for Batch_size Optimizer learning rate traial_9

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

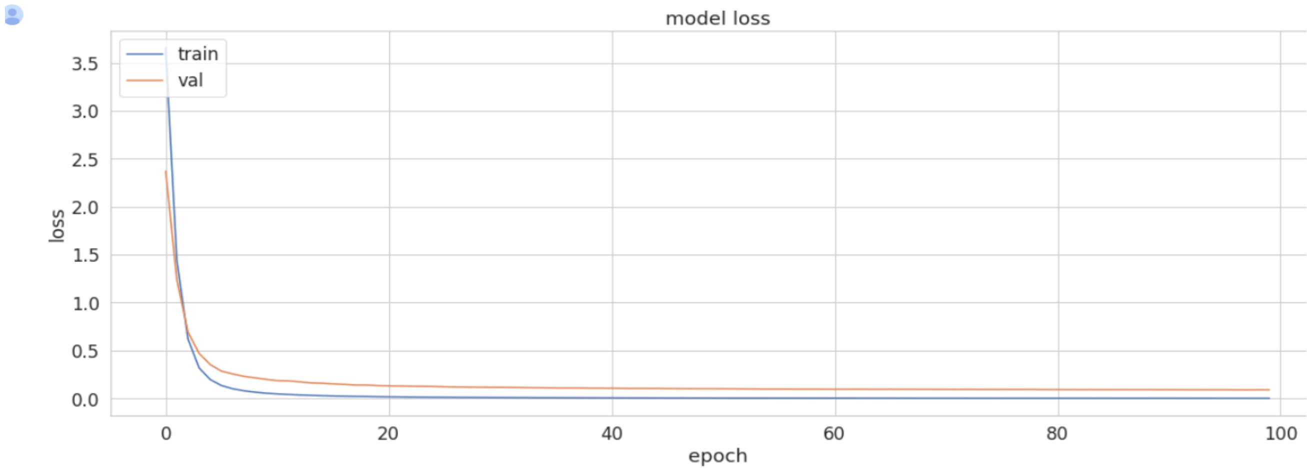

Figure 46: loss plot for Batch_size Optimizer learning rate traial_9

Observation: From the previous trials, we discovered that the best **batch_size=32**

8.4 Model hidden units

```
[ ] from keras import regularizers
    from keras.callbacks import EarlyStopping
    def training(unit):

      # structure model
      features= X_train.shape[1]
      model = Sequential()

      model.add(Dense(units = unit, activation = 'tanh', input_shape=(features,)))
      # model.add(Dropout(0.1))


      model.add(Dense(units=99, activation = 'softmax'))

      # Compiling the ANN
      early_stop = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.001)
      opt = tensorflow.keras.optimizers.Adam(0.001)
      model.compile(opt, loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

      # Fitting the ANN to the Training set
      history= model.fit(X_train, y_train,validation_data=(X_val, y_val), batch_size = 32, epochs = 100,verbose=0)
```

Figure 47: Model hidden units

8.4.1 hidden unit traial_10

Trial_10 (256)

```
[ ] model,history= training(256)
```

```
[ ] evaluate(model,X_train,y_train)

    25/25 [==============================] - 0s 2ms/step - loss: 0.0012 - accuracy: 1.0000

     loss=  0.0011642652098089457

     Accuracy=  1.0
```

```
[ ] evaluate(model,X_val,y_val)

    7/7 [==============================] - 0s 3ms/step - loss: 0.0667 - accuracy: 0.9798

     loss=  0.0667441263794899

     Accuracy=  0.9797979593276978
```

```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

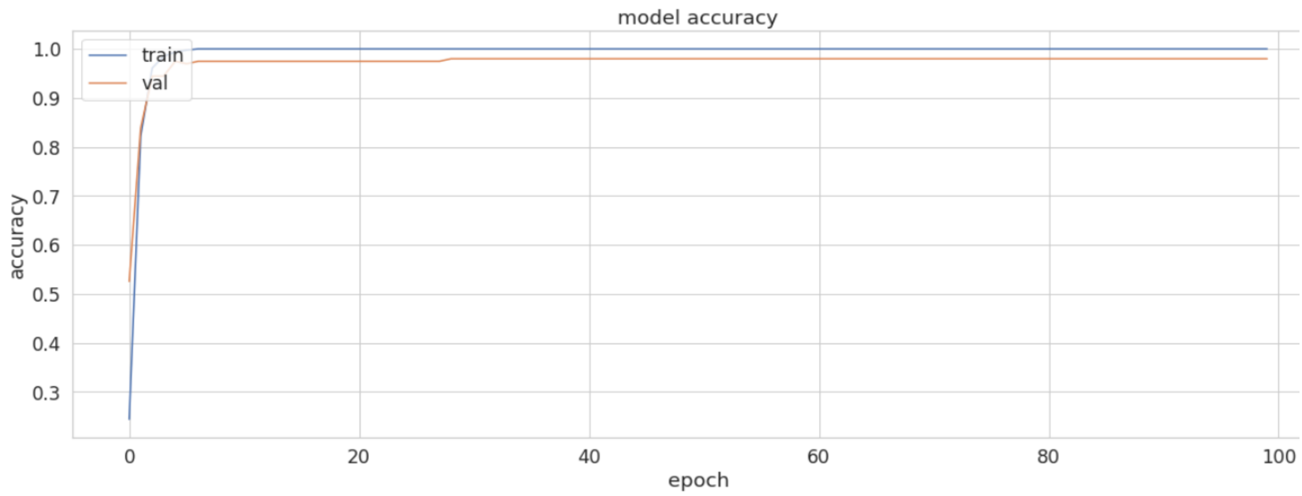

Figure 48: accuracy plot for hidden unit Batch_size Optimizer learning rate traial_10

```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

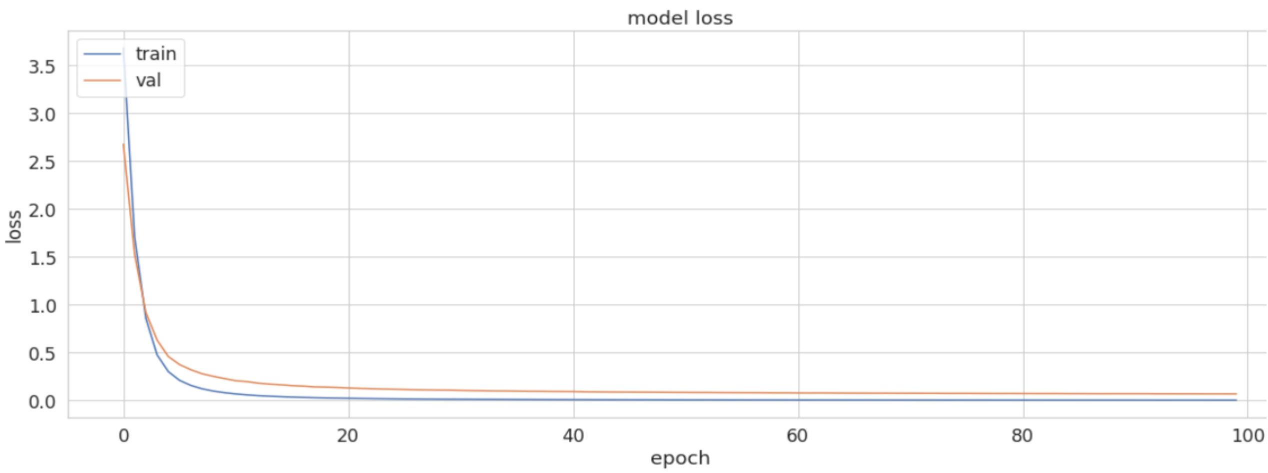

Figure 49: loss plot for hidden unit Batch_size Optimizer learning rate traial_10

8.4.2 hidden unit traial_11

Trial_11(512)

```
[ ] model,history= training(512)
```

```
[ ] evaluate(model,X_train,y_train)
```

```
25/25 [==============================] - 0s 4ms/step - loss: 5.2321e-04 - accuracy: 1.0000

 loss=  0.0005232118419371545

 Accuracy=  1.0
```

```
[ ] evaluate(model,X_val,y_val)
```

```
7/7 [==============================] - 0s 4ms/step - loss: 0.0906 - accuracy: 0.9798

 loss=  0.09059968590736389

 Accuracy=  0.9797979593276978
```

```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

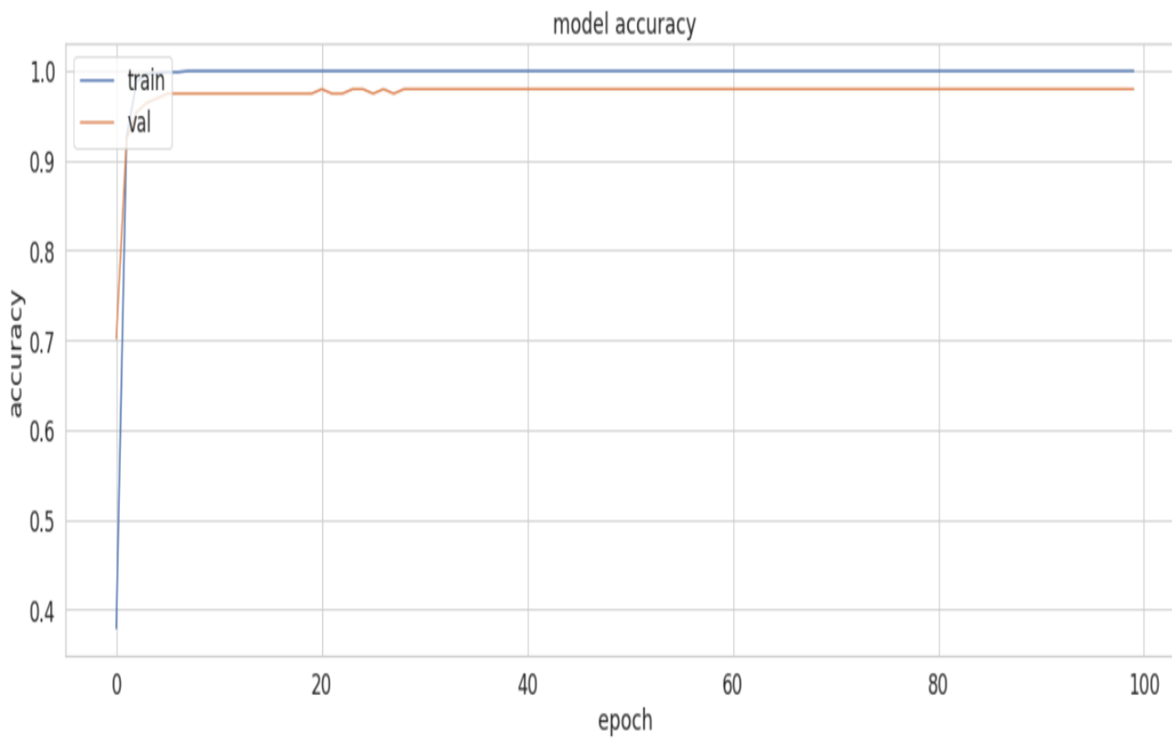

Figure 50: accuracy plot for hidden unit Batch_size Optimizer learning rate traial_11

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

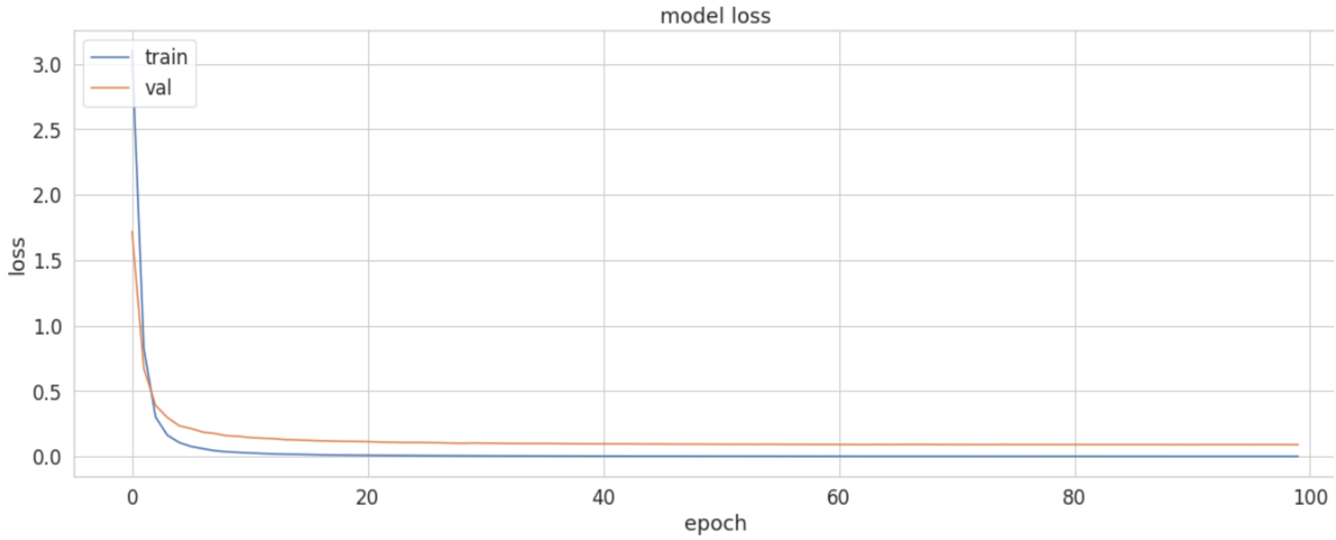

Figure 51: accuracy plot for hidden unit Batch_size Optimizer learning rate traial_11

8.4.3 hidden unit traial_12

Trial_12(128)

```
[ ] model,history= training(128)
```

```
[ ] evaluate(model,X_train,y_train)
```

```
25/25 [==============================] - 0s 2ms/step - loss: 0.0029 - accuracy: 1.0000

 loss=  0.002917163074016571

 Accuracy=  1.0
```

```
[ ] evaluate(model,X_val,y_val)
```

```
7/7 [==============================] - 0s 2ms/step - loss: 0.0859 - accuracy: 0.9747

 loss=  0.08594694733361969

 Accuracy=  0.9747474789619446
```

```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

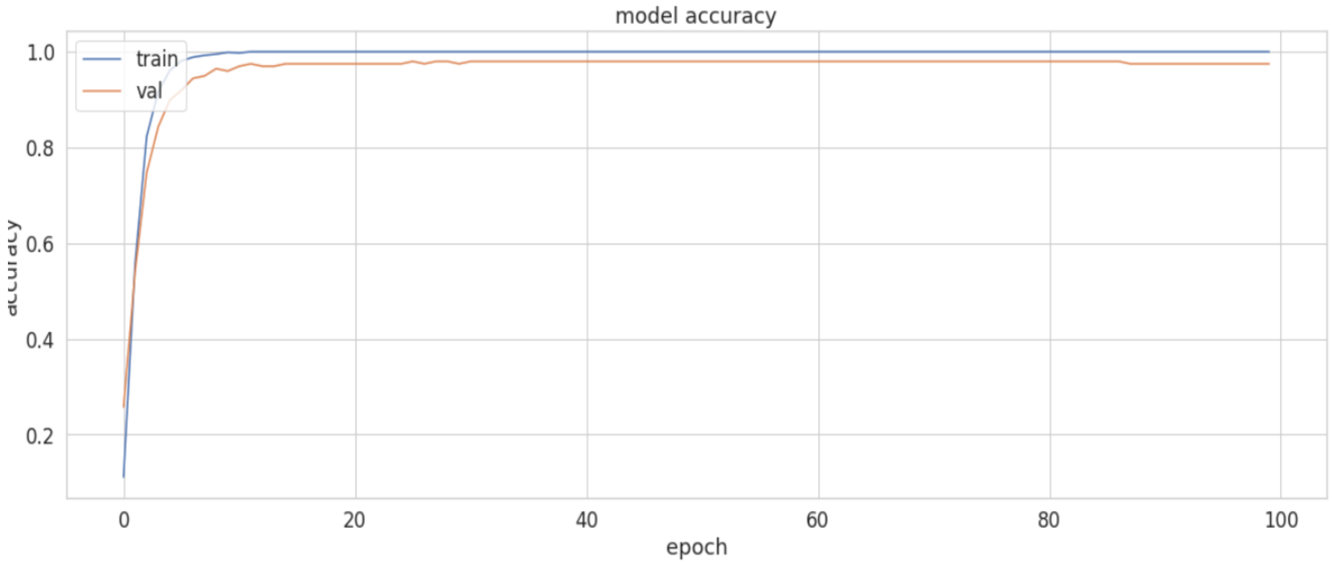

Figure 51: accuracy plot for hidden unit Batch_size Optimizer learning rate traial_12

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

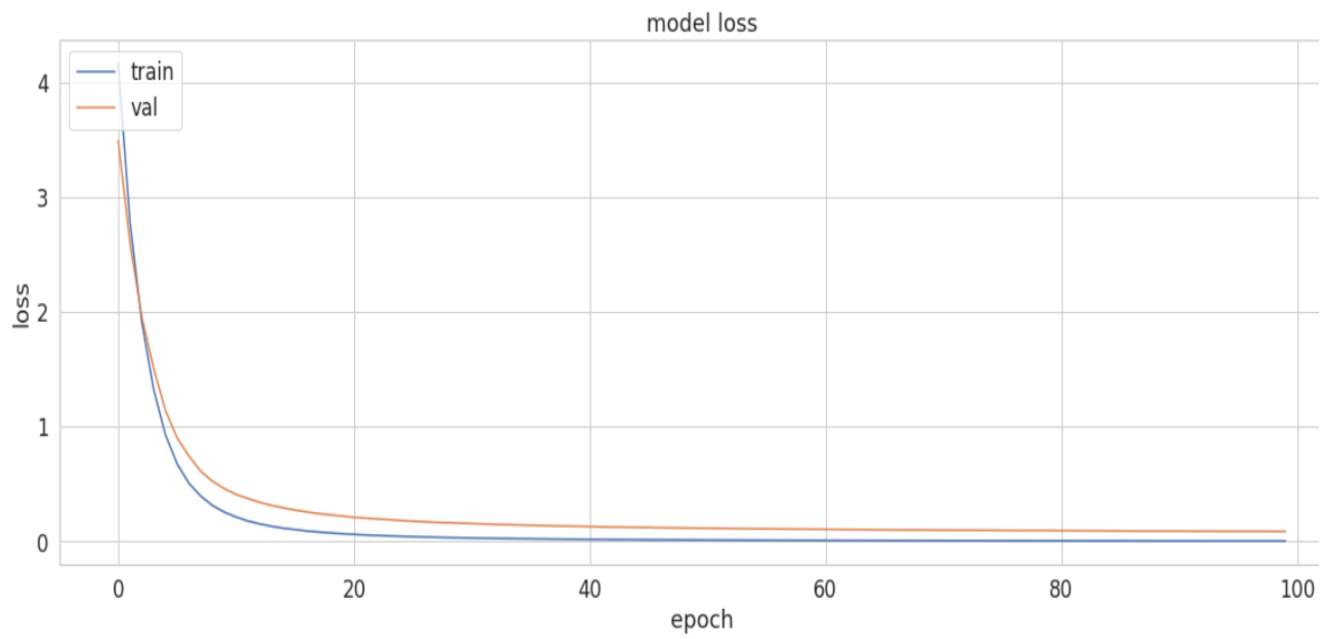

Figure 52: loss plot for hidden unit Batch_size Optimizer learning rate traial_12

Observation: From the previous trials, we discovered that best hidden **units=256**

9. Conclusion

We tried different trials with different hyperparameters and chosed the best trial as the end.

The trials were from these hyperparameters values:

1) optimizer (Adam, SGD, RMSprop)

2) learning rate (.1, .01, .001)

3) Batch size (16, 32, 64)

4) Hidden units (128, 256, 512)

Optimizer	Learning rate	Batch size	Hidden units
Adam	0.1	16	128
SGB	0.01	32	256
RMSprop	0.001	64	512

From these trials we discovered that the best trial of them is (Optimizer: Adam) , (Learning rate : 0.001) , (batch size : 32) and (Hidden unit : 256)

With validation loss = `0.0667441263794899`

and validation accuracy = `0.9797979593276978`

10. Reference

https://towardsdatascience.com/how-to-explore-and-visualize-a-dataset-with-python-7da5024900ef

https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/

https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/