NASA Asteroid Classification Using Deep Neural Network

May 31, 2021

1 NASA Asteroid Classification Using Deep Neural Network

1.1 Group Members:

- 2018BTEEN00065 Pradyumna Santosh Akolkar
- 2019BTEEN00206 Aishwarya Jagannath Kumbhar

1.2 Objectives:

- Understanding variables included and preprocessing the data
- Graphically representing the data for insights
- Using Deep Learning to classify the asteroids as Hazardous or Non-Hazardous

1.3 Introduction:

Machine Learning can be used to classify the data which depends on many complex variables. Various classification methods can used to do so. Here, we use Deep Neural Network for analyzing a dataset containing information about different asteroids taken from kaggle.

Link for the Dataset - https://www.kaggle.com/shrutimehta/nasa-asteroids-classification

This dataset contains a csv file which has 4687 rows and 40 columns having various information such as Neo Reference ID, Absolute Magnitude, Orbit ID, Estimated Diameter, Eccentricity etc.

1.4 Importing Necessary Libraries:

[1]:
```
import os
import numpy as np
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
from seaborn import heatmap
```

1.5 Preprocessing the Data:

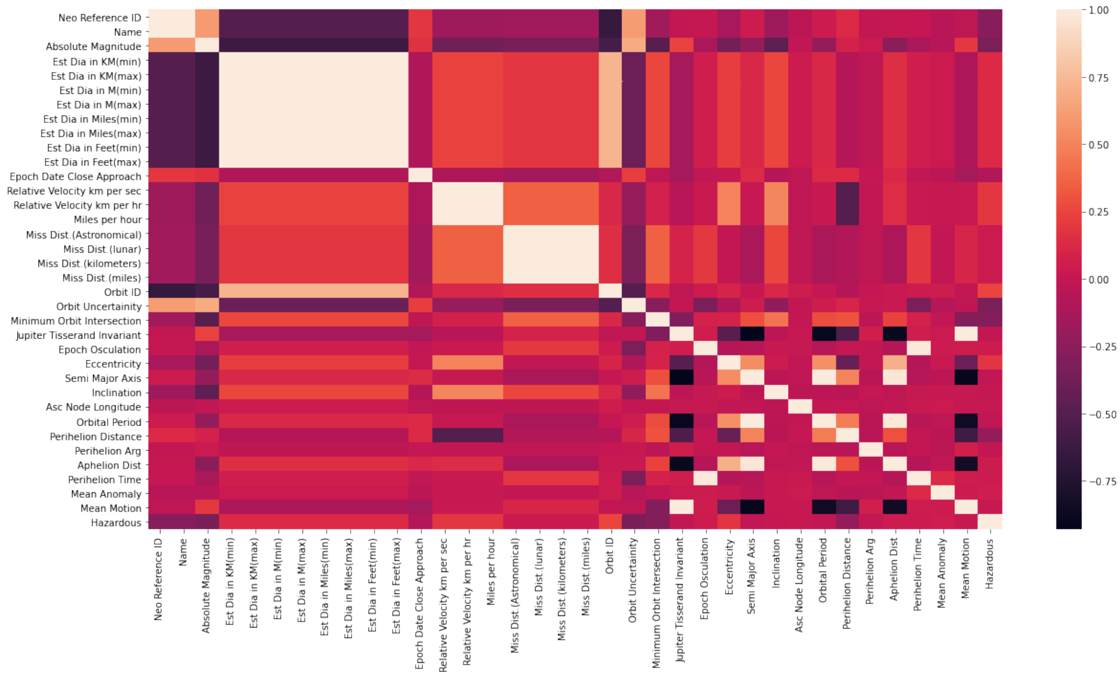
[2]:
```
# Reading csv file
df = pd.read_csv('nasa.csv')
```

The dataset contains 40 columns, which are as follows-

- Neo Reference ID
- Name
- Absolute Magnitude
- Est Dia in KM (min)
- Est Dia in KM (max)
- Est Dia in M (min)
- Est Dia in M (max)
- Est Dia in Miles (min)
- Est Dia in Miles (max)
- Est Dia in Feet (min)
- Est Dia in Feet (max)
- Close Approach Date
- Epoch Date Close Approach
- Relative Velocity KM per Sec
- Relative Velocity M per Hour
- Miles per Hour
- Miss Dist. (Astronomical)
- Miss Dist. (Lunar)
- Miss Dist. (Kilometer)
- Miss Dist. (Miles)
- Orbiting Body
- Orbit ID
- Orbit Determination Date
- Orbit Uncertainty
- Minimum Orbit Intersection
- Jupiter Tisserand Invariant
- Epoch Osculation
- Eccentricity
- Semi Major Axis
- Inclination
- Asc Node Longitude
- Orbital Period
- Perihelion Distance
- Perihelion Arg
- Apehelion Dist.
- Perihelion Time
- Mean Anomaly
- Mean Motion
- Equinox
- Hazardous

Let us take Correlation Matrix as Heatmap to understand dependency in between columns.

[3]:
```
# Plotting Correlation Matrix as Heatmap
plt.figure(figsize=(20,10))
heatmap(df.corr())
plt.show()
```



From above Heatmap we decided to take following columns as features to Machine Learning Model-

1. Absolute Magnitude
2. Est Dia in M (max)
3. Relative Velocity KM per Sec
4. Miss Dist. (Lunar)
5. Minimum Orbit Intersection
6. Jupiter Tisserand Invariant
7. Eccentricity
8. Inclination
9. Asc Node Longitude
10. Perihelion Distance
11. Mean Anomaly

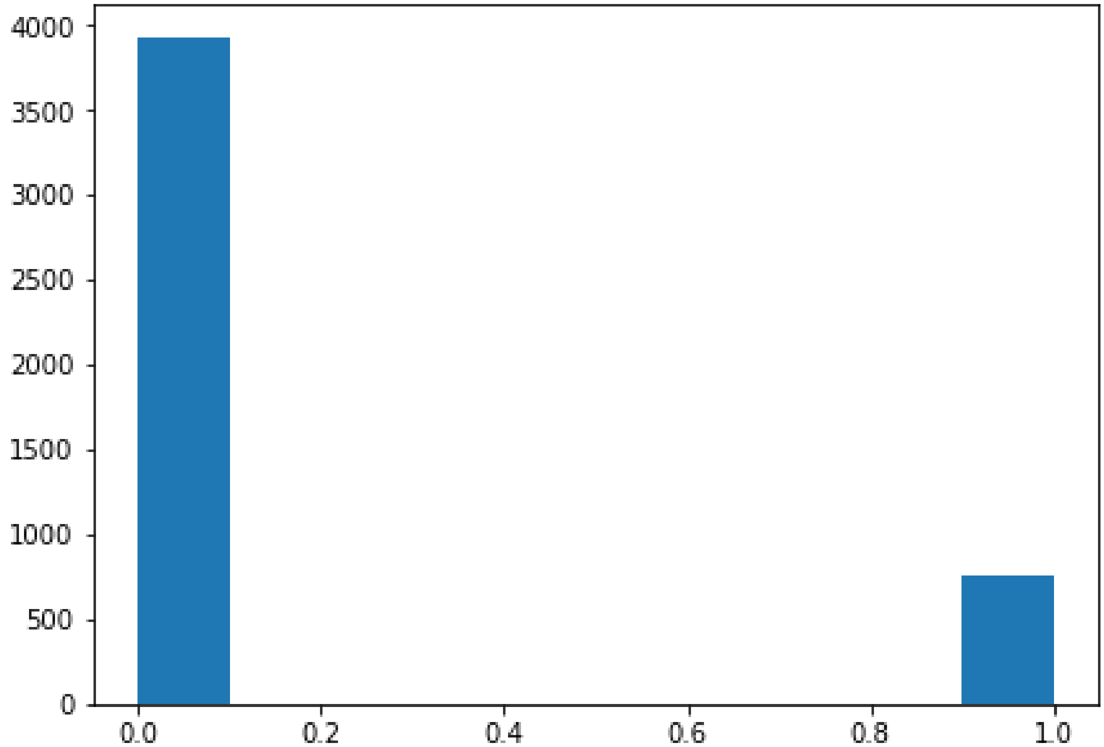
And here, the column 'Hazardous' serves as label to data. We use these 11 features along with labels to train our Deep Neural Network.

1.6 Dataset Insights:

[4]:
```
labels = df['Hazardous']
labels = labels.astype('float')
print(labels.value_counts())
labels.hist(figsize=(7,5),grid=False)
plt.show()
```

```
0.0    3932
1.0     755
```

```
Name: Hazardous, dtype: int64
```



Here, we see that there is large difference between number of True (1.0) and False (0.0) values for 'Hazardous' column. Thus, we can conclude that our dataset is skewed.

1.7 Model Insights:

The important part of Machine Learning Model are the different hyperparameters such as choice of activation functions, number of layers, type of layers etc. In our model, we choose Dense layer with activation function as 'ReLU' as it gives better performance and due to its properties related to non-linearity, such as reduced likelihood of gradient vanishing. For the output layer, we choose 'Sigmoid' as activation function because we need to perform binary classification (Hazardous or Non-Hazardous). We use Dropout layers in between, to reduce overfitting of the data. We use 'adam' as optimizer it is one of the more robust optimizers. As this is binary classification, we use 'binary crossentropy' as loss function. We have a skewed dataset, thus, along with accuracy, we use precision and recall to measure the performance of our neural network.

1.8 Defining Helper Functions:

[5]:
```
# Function to perform min-max normalization on the dataframe
def normalize(df):
    normalized_df = df.copy()
```

```
    normalized_df = (normalized_df - normalized_df.min()) / (normalized_df.
↪max() - normalized_df.min())

    return normalized_df
```

[6]:
```
# Function to create normalized dataset from a given dataframe and␣
↪train_percentage
def create_dataset(df,train_percentage):
    #Selecting necessary columns from df
    selected_df = df[['Name','Absolute Magnitude','Est Dia in M(max)',
                      'Relative Velocity km per sec','Miss Dist.(lunar)',
                      'Minimum Orbit Intersection','Jupiter Tisserand␣
↪Invariant',
                      'Eccentricity','Inclination','Asc Node Longitude',
                      'Perihelion Distance','Mean Anomaly','Hazardous']]

    #Shuffling the selected dataframe to improve ML Classifier Performance
    selected_df.sample(frac=1).reset_index(drop=True,inplace=True)

    #Separating labels and names from selected_df
    labels = selected_df['Hazardous']
    names = selected_df['Name']
    selected_df = selected_df.drop(labels=['Hazardous','Name'],axis=1)

    #Normalizing selected_df
    selected_df = normalize(selected_df)

    #Allocating train-test percentage
    num_examples = selected_df.shape[0]
    train_index = int(num_examples*train_percentage/100)

    #Dividing datasets in train dataset and test dataset
    train_names = names[:train_index]
    train_x = selected_df[:train_index]
    train_y = labels[:train_index]
    test_names = names[train_index:]
    test_x = selected_df[train_index:]
    test_y = labels[train_index:]

    return train_x, train_y, test_x, test_y, train_names, test_names
```

[7]:
```
# Function to plot performance of model using predicted and actual values
def plot_performance(x,pred_y,actual_y,title,fontsize=22):
    plt.rcParams['font.family'] = 'Times New Roman'
    plt.rcParams['font.size'] = 18
    fig, (ax1,ax2) = plt.subplots(1,2,figsize=(20,9))
    fig.suptitle(title,fontsize=fontsize)
```

```
    #Plotting scatter plot of predicted values
    ax1.scatter(x,pred_y,color='red',label="Predicted Values")
    ax1.set_xlabel("Name (ID) of Asteroid")
    ax1.set_ylabel("Hazardous / Non-Hazardous")
    ax1.legend(['Predicted Values'],loc='upper right',bbox_to_anchor = (1.0,1.
↪1))

    #Plotting scatter plot of actual values
    ax2.scatter(x,actual_y,color='blue',label="Actual Values")
    ax2.set_xlabel("Name (ID) of Asteroid")
    ax2.set_ylabel("Hazardous / Non-Hazardous")
    ax2.legend(['Actual Values'],loc='upper right', bbox_to_anchor = (1.0,1.1))
    plt.subplots_adjust(wspace=0.125)
    plt.show()
```

[8]:
```
# Function to calculate F1 Score
def calculate_f1_score(precision,recall):
    f1_score = 2*precision*recall/(precision + recall)

    return f1_score
```

1.9 Building Deep Neural Network:

In our model we implement following architecture- InputLayer ⟶ Dense ⟶ Dense ⟶ Dropout ⟶ Dense ⟶ Dropout ⟶ Dense ⟶ Dropout ⟶ Dense ⟶ Output

[9]:
```
# Function to build model
def build_model(seed):
    np.random.seed(seed)
    model=tf.keras.Sequential([
                        tf.keras.layers.InputLayer(input_shape=(11,)),
                        tf.keras.layers.Dense(units=10,activation='relu'),
                        tf.keras.layers.Dense(units=50,activation='relu'),
                        tf.keras.layers.Dropout(0.2),
                        tf.keras.layers.Dense(units=100,activation='relu'),
                        tf.keras.layers.Dropout(0.4),
                        tf.keras.layers.Dense(units=75,activation='relu'),
                        tf.keras.layers.Dropout(0.2),
                        tf.keras.layers.Dense(units=50,activation='relu'),
                        tf.keras.layers.Dense(units=1,activation='sigmoid')
    ])
    model.
↪compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy',tf.
↪keras.metrics.Recall(name='recall'),tf.keras.metrics.
↪Precision(name='precision')])
```

```
    return model
```

[10]:
```
# Function to run model
def run_model(model,train_x,train_y,epochs,batch_size=64,verbose=0):
    history=model.
↪fit(x=train_x,y=train_y,batch_size=batch_size,epochs=epochs,verbose=verbose,validation_spli
↪1)

    return history
```

1.10 Implementation of Machine Learning Model:

[11]:
```
# Creating dataset using 'nasa.csv' as dataframe df
train_set_percentage = 90
train_x, train_y, test_x, test_y, train_names, test_names =␣
↪create_dataset(df,train_set_percentage)
```

[12]:
```
# Summary of created dataset
print("Number of Features: ",train_x.shape[1])
print("Number of Train Set Examples: ",train_x.shape[0])
print("Number of Test Set Examples: ",test_x.shape[0])
```

```
Number of Features:  11
Number of Train Set Examples:  4218
Number of Test Set Examples:  469
```

[13]:
```
# Building Model
model = build_model(1)
```

[14]:
```
# Running Model
history = run_model(model,train_x,train_y,epochs=100,verbose=1)
```

```
Epoch 1/100
60/60 [==============================] - 4s 19ms/step - loss: 0.4966 - accuracy:
0.8269 - recall: 0.0094 - precision: 0.1818 - val_loss: 0.3971 - val_accuracy:
0.8578 - val_recall: 0.0000e+00 - val_precision: 0.0000e+00
Epoch 2/100
60/60 [==============================] - 0s 5ms/step - loss: 0.4270 - accuracy:
0.8325 - recall: 0.0000e+00 - precision: 0.0000e+00 - val_loss: 0.3572 -
val_accuracy: 0.8578 - val_recall: 0.0000e+00 - val_precision: 0.0000e+00
Epoch 3/100
60/60 [==============================] - 0s 5ms/step - loss: 0.3904 - accuracy:
0.8325 - recall: 0.0000e+00 - precision: 0.0000e+00 - val_loss: 0.3389 -
val_accuracy: 0.8578 - val_recall: 0.0000e+00 - val_precision: 0.0000e+00
Epoch 4/100
60/60 [==============================] - 0s 5ms/step - loss: 0.3561 - accuracy:
0.8325 - recall: 0.0000e+00 - precision: 0.0000e+00 - val_loss: 0.2900 -
```

```
val_accuracy: 0.8578 - val_recall: 0.0000e+00 - val_precision: 0.0000e+00
Epoch 5/100
60/60 [==============================] - 0s 5ms/step - loss: 0.3094 - accuracy:
0.8417 - recall: 0.0739 - precision: 0.7966 - val_loss: 0.2673 - val_accuracy:
0.8910 - val_recall: 0.5667 - val_precision: 0.6296
Epoch 6/100
60/60 [==============================] - 0s 5ms/step - loss: 0.2641 - accuracy:
0.8870 - recall: 0.4874 - precision: 0.7506 - val_loss: 0.2145 - val_accuracy:
0.9100 - val_recall: 0.6000 - val_precision: 0.7200
Epoch 7/100
60/60 [==============================] - 0s 5ms/step - loss: 0.2248 - accuracy:
0.9031 - recall: 0.6415 - precision: 0.7445 - val_loss: 0.1996 - val_accuracy:
0.9171 - val_recall: 0.6000 - val_precision: 0.7660
Epoch 8/100
60/60 [==============================] - 0s 5ms/step - loss: 0.2064 - accuracy:
0.9139 - recall: 0.7107 - precision: 0.7597 - val_loss: 0.1914 - val_accuracy:
0.9265 - val_recall: 0.6167 - val_precision: 0.8222
Epoch 9/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1855 - accuracy:
0.9228 - recall: 0.7280 - precision: 0.7942 - val_loss: 0.1736 - val_accuracy:
0.9242 - val_recall: 0.6333 - val_precision: 0.7917
Epoch 10/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1717 - accuracy:
0.9318 - recall: 0.7531 - precision: 0.8244 - val_loss: 0.1576 - val_accuracy:
0.9265 - val_recall: 0.6500 - val_precision: 0.7959
Epoch 11/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1589 - accuracy:
0.9399 - recall: 0.7877 - precision: 0.8434 - val_loss: 0.1601 - val_accuracy:
0.9360 - val_recall: 0.6333 - val_precision: 0.8837
Epoch 12/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1439 - accuracy:
0.9397 - recall: 0.7893 - precision: 0.8409 - val_loss: 0.1386 - val_accuracy:
0.9384 - val_recall: 0.7333 - val_precision: 0.8148
Epoch 13/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1426 - accuracy:
0.9391 - recall: 0.7893 - precision: 0.8381 - val_loss: 0.1341 - val_accuracy:
0.9336 - val_recall: 0.6333 - val_precision: 0.8636
Epoch 14/100
60/60 [==============================] - 0s 6ms/step - loss: 0.1247 - accuracy:
0.9489 - recall: 0.8302 - precision: 0.8599 - val_loss: 0.1439 - val_accuracy:
0.9336 - val_recall: 0.6167 - val_precision: 0.8810
Epoch 15/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1242 - accuracy:
0.9478 - recall: 0.8192 - precision: 0.8626 - val_loss: 0.1212 - val_accuracy:
0.9431 - val_recall: 0.6833 - val_precision: 0.8913
Epoch 16/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1128 - accuracy:
0.9555 - recall: 0.8459 - precision: 0.8834 - val_loss: 0.1132 - val_accuracy:
```

```
0.9550 - val_recall: 0.8000 - val_precision: 0.8727
Epoch 17/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1117 - accuracy:
0.9560 - recall: 0.8569 - precision: 0.8776 - val_loss: 0.1125 - val_accuracy:
0.9431 - val_recall: 0.6833 - val_precision: 0.8913
Epoch 18/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1015 - accuracy:
0.9534 - recall: 0.8412 - precision: 0.8756 - val_loss: 0.0853 - val_accuracy:
0.9716 - val_recall: 0.9000 - val_precision: 0.9000
Epoch 19/100
60/60 [==============================] - 0s 5ms/step - loss: 0.1004 - accuracy:
0.9568 - recall: 0.8601 - precision: 0.8794 - val_loss: 0.0773 - val_accuracy:
0.9716 - val_recall: 0.9000 - val_precision: 0.9000
Epoch 20/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0855 - accuracy:
0.9655 - recall: 0.8884 - precision: 0.9040 - val_loss: 0.0668 - val_accuracy:
0.9787 - val_recall: 0.9167 - val_precision: 0.9322
Epoch 21/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0788 - accuracy:
0.9697 - recall: 0.9041 - precision: 0.9141 - val_loss: 0.0699 - val_accuracy:
0.9763 - val_recall: 0.8667 - val_precision: 0.9630
Epoch 22/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0710 - accuracy:
0.9715 - recall: 0.9104 - precision: 0.9190 - val_loss: 0.0573 - val_accuracy:
0.9763 - val_recall: 0.8667 - val_precision: 0.9630
Epoch 23/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0737 - accuracy:
0.9655 - recall: 0.8868 - precision: 0.9053 - val_loss: 0.0538 - val_accuracy:
0.9834 - val_recall: 0.9167 - val_precision: 0.9649
Epoch 24/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0647 - accuracy:
0.9747 - recall: 0.9182 - precision: 0.9299 - val_loss: 0.0487 - val_accuracy:
0.9834 - val_recall: 0.9167 - val_precision: 0.9649
Epoch 25/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0619 - accuracy:
0.9766 - recall: 0.9245 - precision: 0.9348 - val_loss: 0.0473 - val_accuracy:
0.9810 - val_recall: 0.9167 - val_precision: 0.9483
Epoch 26/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0634 - accuracy:
0.9731 - recall: 0.9151 - precision: 0.9238 - val_loss: 0.0458 - val_accuracy:
0.9858 - val_recall: 0.9167 - val_precision: 0.9821
Epoch 27/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0687 - accuracy:
0.9710 - recall: 0.9167 - precision: 0.9109 - val_loss: 0.0676 - val_accuracy:
0.9716 - val_recall: 0.8167 - val_precision: 0.9800
Epoch 28/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0565 - accuracy:
0.9787 - recall: 0.9214 - precision: 0.9498 - val_loss: 0.0621 - val_accuracy:
```

```
0.9739 - val_recall: 0.8333 - val_precision: 0.9804
Epoch 29/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0587 - accuracy:
0.9760 - recall: 0.9214 - precision: 0.9346 - val_loss: 0.0527 - val_accuracy:
0.9763 - val_recall: 0.8500 - val_precision: 0.9808
Epoch 30/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0486 - accuracy:
0.9818 - recall: 0.9371 - precision: 0.9536 - val_loss: 0.0462 - val_accuracy:
0.9810 - val_recall: 0.8833 - val_precision: 0.9815
Epoch 31/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0493 - accuracy:
0.9808 - recall: 0.9340 - precision: 0.9504 - val_loss: 0.0269 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 32/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0492 - accuracy:
0.9784 - recall: 0.9340 - precision: 0.9369 - val_loss: 0.0467 - val_accuracy:
0.9810 - val_recall: 0.8667 - val_precision: 1.0000
Epoch 33/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0478 - accuracy:
0.9808 - recall: 0.9277 - precision: 0.9562 - val_loss: 0.0259 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 34/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0387 - accuracy:
0.9855 - recall: 0.9528 - precision: 0.9604 - val_loss: 0.0315 - val_accuracy:
0.9882 - val_recall: 0.9333 - val_precision: 0.9825
Epoch 35/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0484 - accuracy:
0.9776 - recall: 0.9245 - precision: 0.9408 - val_loss: 0.0311 - val_accuracy:
0.9858 - val_recall: 0.9167 - val_precision: 0.9821
Epoch 36/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0410 - accuracy:
0.9847 - recall: 0.9418 - precision: 0.9661 - val_loss: 0.0279 - val_accuracy:
0.9882 - val_recall: 0.9333 - val_precision: 0.9825
Epoch 37/100
60/60 [==============================] - 0s 7ms/step - loss: 0.0436 - accuracy:
0.9829 - recall: 0.9465 - precision: 0.9510 - val_loss: 0.0273 - val_accuracy:
0.9882 - val_recall: 0.9167 - val_precision: 1.0000
Epoch 38/100
60/60 [==============================] - 0s 7ms/step - loss: 0.0468 - accuracy:
0.9808 - recall: 0.9371 - precision: 0.9475 - val_loss: 0.0372 - val_accuracy:
0.9882 - val_recall: 0.9167 - val_precision: 1.0000
Epoch 39/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0396 - accuracy:
0.9842 - recall: 0.9481 - precision: 0.9571 - val_loss: 0.0338 - val_accuracy:
0.9882 - val_recall: 0.9500 - val_precision: 0.9661
Epoch 40/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0456 - accuracy:
0.9797 - recall: 0.9340 - precision: 0.9444 - val_loss: 0.0237 - val_accuracy:
```

```
0.9905 - val_recall: 0.9500 - val_precision: 0.9828
Epoch 41/100
60/60 [==============================] - 0s 6ms/step - loss: 0.0399 - accuracy:
0.9850 - recall: 0.9528 - precision: 0.9573 - val_loss: 0.0384 - val_accuracy:
0.9810 - val_recall: 0.8833 - val_precision: 0.9815
Epoch 42/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0436 - accuracy:
0.9834 - recall: 0.9403 - precision: 0.9599 - val_loss: 0.0394 - val_accuracy:
0.9834 - val_recall: 0.9000 - val_precision: 0.9818
Epoch 43/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0373 - accuracy:
0.9860 - recall: 0.9528 - precision: 0.9634 - val_loss: 0.0252 - val_accuracy:
0.9882 - val_recall: 0.9333 - val_precision: 0.9825
Epoch 44/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0364 - accuracy:
0.9850 - recall: 0.9513 - precision: 0.9588 - val_loss: 0.0264 - val_accuracy:
0.9882 - val_recall: 0.9333 - val_precision: 0.9825
Epoch 45/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0386 - accuracy:
0.9850 - recall: 0.9497 - precision: 0.9603 - val_loss: 0.0186 - val_accuracy:
0.9905 - val_recall: 0.9500 - val_precision: 0.9828
Epoch 46/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0352 - accuracy:
0.9839 - recall: 0.9465 - precision: 0.9571 - val_loss: 0.0305 - val_accuracy:
0.9882 - val_recall: 0.9167 - val_precision: 1.0000
Epoch 47/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0391 - accuracy:
0.9829 - recall: 0.9465 - precision: 0.9510 - val_loss: 0.0277 - val_accuracy:
0.9882 - val_recall: 0.9333 - val_precision: 0.9825
Epoch 48/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0379 - accuracy:
0.9852 - recall: 0.9465 - precision: 0.9647 - val_loss: 0.0214 - val_accuracy:
0.9905 - val_recall: 0.9500 - val_precision: 0.9828
Epoch 49/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0364 - accuracy:
0.9837 - recall: 0.9560 - precision: 0.9470 - val_loss: 0.0216 - val_accuracy:
0.9905 - val_recall: 0.9500 - val_precision: 0.9828
Epoch 50/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0309 - accuracy:
0.9887 - recall: 0.9575 - precision: 0.9744 - val_loss: 0.0208 - val_accuracy:
0.9882 - val_recall: 0.9167 - val_precision: 1.0000
Epoch 51/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0363 - accuracy:
0.9850 - recall: 0.9497 - precision: 0.9603 - val_loss: 0.0217 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 52/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0391 - accuracy:
0.9839 - recall: 0.9513 - precision: 0.9528 - val_loss: 0.0178 - val_accuracy:
```

```
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 53/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0304 - accuracy:
0.9871 - recall: 0.9607 - precision: 0.9622 - val_loss: 0.0241 - val_accuracy:
0.9882 - val_recall: 0.9333 - val_precision: 0.9825
Epoch 54/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0341 - accuracy:
0.9855 - recall: 0.9528 - precision: 0.9604 - val_loss: 0.0179 - val_accuracy:
0.9882 - val_recall: 0.9667 - val_precision: 0.9508
Epoch 55/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0318 - accuracy:
0.9868 - recall: 0.9607 - precision: 0.9607 - val_loss: 0.0158 - val_accuracy:
0.9976 - val_recall: 0.9833 - val_precision: 1.0000
Epoch 56/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0302 - accuracy:
0.9876 - recall: 0.9607 - precision: 0.9652 - val_loss: 0.0179 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 57/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0367 - accuracy:
0.9829 - recall: 0.9465 - precision: 0.9510 - val_loss: 0.0265 - val_accuracy:
0.9882 - val_recall: 0.9167 - val_precision: 1.0000
Epoch 58/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0334 - accuracy:
0.9863 - recall: 0.9481 - precision: 0.9695 - val_loss: 0.0178 - val_accuracy:
0.9953 - val_recall: 0.9667 - val_precision: 1.0000
Epoch 59/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0381 - accuracy:
0.9837 - recall: 0.9575 - precision: 0.9457 - val_loss: 0.0487 - val_accuracy:
0.9787 - val_recall: 0.8500 - val_precision: 1.0000
Epoch 60/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0371 - accuracy:
0.9837 - recall: 0.9481 - precision: 0.9541 - val_loss: 0.0356 - val_accuracy:
0.9810 - val_recall: 0.8667 - val_precision: 1.0000
Epoch 61/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0286 - accuracy:
0.9871 - recall: 0.9497 - precision: 0.9726 - val_loss: 0.0242 - val_accuracy:
0.9905 - val_recall: 0.9333 - val_precision: 1.0000
Epoch 62/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0326 - accuracy:
0.9866 - recall: 0.9575 - precision: 0.9621 - val_loss: 0.0209 - val_accuracy:
0.9882 - val_recall: 0.9833 - val_precision: 0.9365
Epoch 63/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0366 - accuracy:
0.9834 - recall: 0.9434 - precision: 0.9569 - val_loss: 0.0249 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 64/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0314 - accuracy:
0.9874 - recall: 0.9654 - precision: 0.9594 - val_loss: 0.0285 - val_accuracy:
```

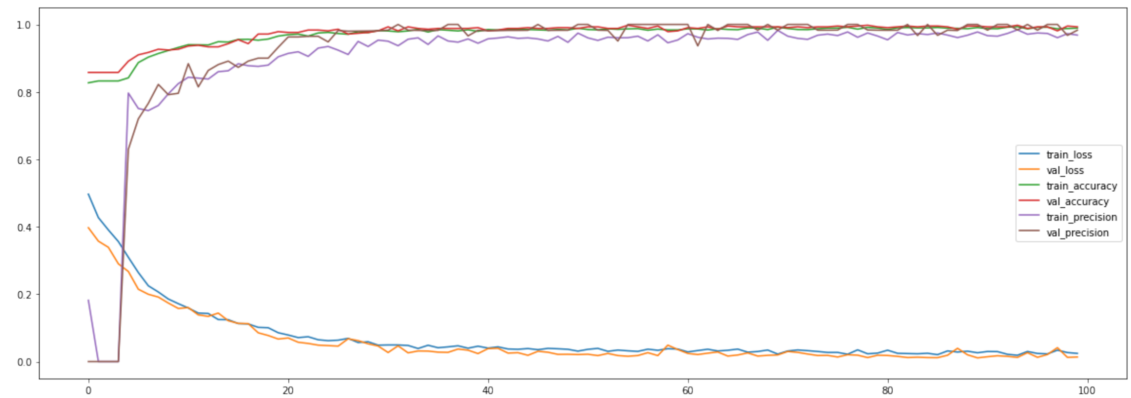
```
0.9858 - val_recall: 0.9167 - val_precision: 0.9821
Epoch 65/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0335 - accuracy:
0.9852 - recall: 0.9528 - precision: 0.9589 - val_loss: 0.0164 - val_accuracy:
0.9953 - val_recall: 0.9667 - val_precision: 1.0000
Epoch 66/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0370 - accuracy:
0.9845 - recall: 0.9513 - precision: 0.9558 - val_loss: 0.0195 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 67/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0278 - accuracy:
0.9897 - recall: 0.9686 - precision: 0.9701 - val_loss: 0.0257 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 68/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0298 - accuracy:
0.9892 - recall: 0.9575 - precision: 0.9775 - val_loss: 0.0164 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 69/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0342 - accuracy:
0.9847 - recall: 0.9560 - precision: 0.9530 - val_loss: 0.0185 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 70/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0223 - accuracy:
0.9921 - recall: 0.9701 - precision: 0.9825 - val_loss: 0.0197 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 71/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0314 - accuracy:
0.9879 - recall: 0.9623 - precision: 0.9653 - val_loss: 0.0299 - val_accuracy:
0.9905 - val_recall: 0.9333 - val_precision: 1.0000
Epoch 72/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0344 - accuracy:
0.9850 - recall: 0.9513 - precision: 0.9588 - val_loss: 0.0273 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 73/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0323 - accuracy:
0.9847 - recall: 0.9528 - precision: 0.9558 - val_loss: 0.0224 - val_accuracy:
0.9905 - val_recall: 0.9333 - val_precision: 1.0000
Epoch 74/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0299 - accuracy:
0.9871 - recall: 0.9544 - precision: 0.9681 - val_loss: 0.0180 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 75/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0271 - accuracy:
0.9892 - recall: 0.9638 - precision: 0.9715 - val_loss: 0.0184 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 76/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0272 - accuracy:
0.9892 - recall: 0.9686 - precision: 0.9670 - val_loss: 0.0136 - val_accuracy:
```

```
0.9953 - val_recall: 0.9833 - val_precision: 0.9833
Epoch 77/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0217 - accuracy:
0.9916 - recall: 0.9717 - precision: 0.9778 - val_loss: 0.0204 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 78/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0348 - accuracy:
0.9860 - recall: 0.9544 - precision: 0.9620 - val_loss: 0.0190 - val_accuracy:
0.9953 - val_recall: 0.9667 - val_precision: 1.0000
Epoch 79/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0231 - accuracy:
0.9913 - recall: 0.9733 - precision: 0.9748 - val_loss: 0.0119 - val_accuracy:
0.9976 - val_recall: 1.0000 - val_precision: 0.9836
Epoch 80/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0251 - accuracy:
0.9895 - recall: 0.9717 - precision: 0.9656 - val_loss: 0.0188 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 81/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0337 - accuracy:
0.9850 - recall: 0.9560 - precision: 0.9545 - val_loss: 0.0181 - val_accuracy:
0.9905 - val_recall: 0.9500 - val_precision: 0.9828
Epoch 82/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0245 - accuracy:
0.9900 - recall: 0.9638 - precision: 0.9761 - val_loss: 0.0155 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 83/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0238 - accuracy:
0.9908 - recall: 0.9764 - precision: 0.9688 - val_loss: 0.0120 - val_accuracy:
0.9953 - val_recall: 0.9667 - val_precision: 1.0000
Epoch 84/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0231 - accuracy:
0.9897 - recall: 0.9654 - precision: 0.9731 - val_loss: 0.0129 - val_accuracy:
0.9929 - val_recall: 0.9833 - val_precision: 0.9672
Epoch 85/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0246 - accuracy:
0.9897 - recall: 0.9686 - precision: 0.9701 - val_loss: 0.0119 - val_accuracy:
0.9953 - val_recall: 0.9667 - val_precision: 1.0000
Epoch 86/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0205 - accuracy:
0.9910 - recall: 0.9717 - precision: 0.9748 - val_loss: 0.0116 - val_accuracy:
0.9953 - val_recall: 1.0000 - val_precision: 0.9677
Epoch 87/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0316 - accuracy:
0.9889 - recall: 0.9654 - precision: 0.9685 - val_loss: 0.0188 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 88/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0283 - accuracy:
0.9884 - recall: 0.9701 - precision: 0.9611 - val_loss: 0.0393 - val_accuracy:
```

```
0.9858 - val_recall: 0.9167 - val_precision: 0.9821
Epoch 89/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0308 - accuracy:
0.9874 - recall: 0.9560 - precision: 0.9682 - val_loss: 0.0201 - val_accuracy:
0.9953 - val_recall: 0.9667 - val_precision: 1.0000
Epoch 90/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0261 - accuracy:
0.9908 - recall: 0.9670 - precision: 0.9777 - val_loss: 0.0110 - val_accuracy:
0.9953 - val_recall: 0.9667 - val_precision: 1.0000
Epoch 91/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0299 - accuracy:
0.9876 - recall: 0.9591 - precision: 0.9667 - val_loss: 0.0144 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 92/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0296 - accuracy:
0.9874 - recall: 0.9591 - precision: 0.9652 - val_loss: 0.0173 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 93/100
60/60 [==============================] - 0s 6ms/step - loss: 0.0215 - accuracy:
0.9918 - recall: 0.9780 - precision: 0.9734 - val_loss: 0.0155 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 94/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0187 - accuracy:
0.9934 - recall: 0.9764 - precision: 0.9842 - val_loss: 0.0124 - val_accuracy:
0.9976 - val_recall: 1.0000 - val_precision: 0.9836
Epoch 95/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0299 - accuracy:
0.9868 - recall: 0.9497 - precision: 0.9711 - val_loss: 0.0264 - val_accuracy:
0.9882 - val_recall: 0.9167 - val_precision: 1.0000
Epoch 96/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0241 - accuracy:
0.9913 - recall: 0.9733 - precision: 0.9748 - val_loss: 0.0127 - val_accuracy:
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
Epoch 97/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0228 - accuracy:
0.9918 - recall: 0.9780 - precision: 0.9734 - val_loss: 0.0209 - val_accuracy:
0.9929 - val_recall: 0.9500 - val_precision: 1.0000
Epoch 98/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0338 - accuracy:
0.9871 - recall: 0.9623 - precision: 0.9608 - val_loss: 0.0412 - val_accuracy:
0.9810 - val_recall: 0.8667 - val_precision: 1.0000
Epoch 99/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0267 - accuracy:
0.9879 - recall: 0.9544 - precision: 0.9728 - val_loss: 0.0125 - val_accuracy:
0.9953 - val_recall: 1.0000 - val_precision: 0.9677
Epoch 100/100
60/60 [==============================] - 0s 5ms/step - loss: 0.0243 - accuracy:
0.9892 - recall: 0.9670 - precision: 0.9685 - val_loss: 0.0136 - val_accuracy:
```

```
0.9929 - val_recall: 0.9667 - val_precision: 0.9831
```

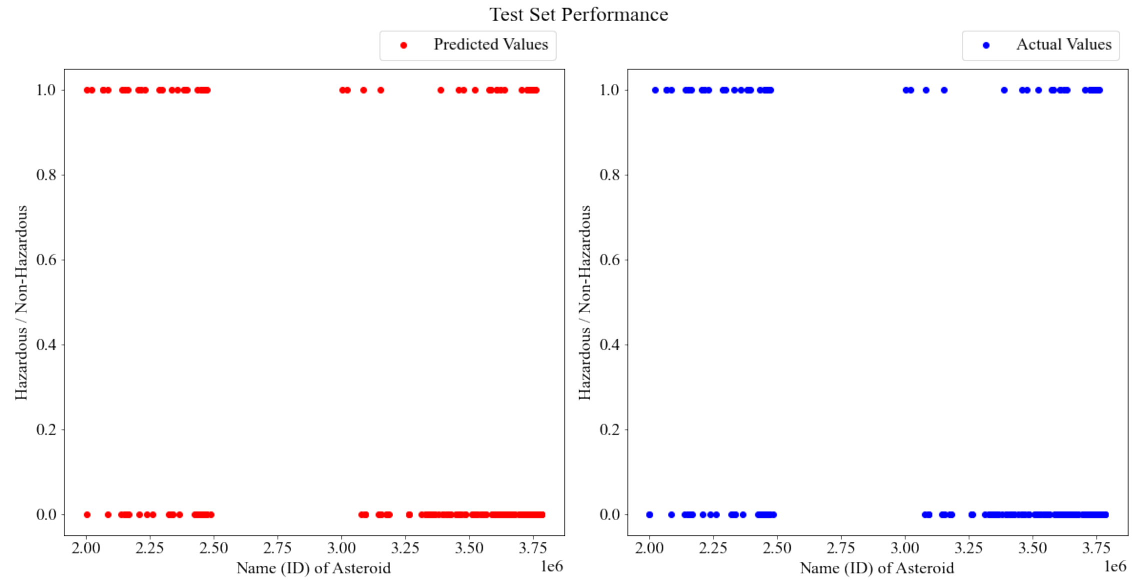
[15]:
```
# Ploting Metrics
plt.figure(figsize=(20,7))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['precision'])
plt.plot(history.history['val_precision'])
plt.legend(['train_loss',␣
↪'val_loss','train_accuracy','val_accuracy','train_precision','val_precision'],␣
↪loc='center right')
plt.show()
```



[16]:
```
# Gathering predictions for Test Set
pred_y = model.predict(test_x,batch_size=64,verbose=1)
pred_y = np.where(pred_y>=0.5,1,0)
```

```
8/8 [==============================] - 0s 2ms/step
```

[17]:
```
# Ploting performance on Test Set
plot_performance(test_names,pred_y,test_y,"Test Set Performance")
```



[18]:
```python
# Evaluating Test Set performance
test_metrics = model.evaluate(test_x,test_y,verbose=0)
test_metrics = np.round(np.multiply(test_metrics,100), 4)
f1_score = round(calculate_f1_score(test_metrics[2],test_metrics[3])/100,4)
print("% Loss: ",test_metrics[0])
print("% Accuracy: ", test_metrics[1])
print("% Recall: ", test_metrics[2])
print("% Precision: ", test_metrics[3])
print("F1 Score: ",f1_score)
```

```
% Loss:  1.9859
% Accuracy:  99.7868
% Recall:  100.0
% Precision:  98.3333
F1 Score:  0.9916
```

1.11 Conclusion:

Various insights such as maximum, minimum and average values are gathered through preprocessing of the data. Built Deep Neural Network using number of Dense layers having activation functions such as 'ReLU' and 'Sigmoid' along with few Dropout layers works with 99.79 % accuracy on Test Set. As the dataset is Skewed, a better measure of performance is F1 Score which comes out to be 0.99 on Test Set. From the graph, we can conclude that predicted and actual values have similar nature suggesting that model is extracting features correctly. Hyperparameter tuning can be further done to improve performance of the Deep Neural Network.

1.12 References:

1. https://www.tensorflow.org/

2. https://stats.stackexchange.com/questions/126238/what-are-the-advantages-of-relu-over-sigmoid-function-in-deep-neural-networks
3. https://stats.stackexchange.com/questions/232719/what-is-the-reason-that-the-adam-optimizer-is-considered-robust-to-the-value-of