Configuration Manual

MSc Research Project
Data Analytics

Nikhil Kadam
Student ID: 22110712

School of Computing
National College of Ireland

Supervisor: Prof. Aaloka Anant

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name	Nikhil Kadam
Student ID	22110712
Programme	Data Analytics
Year:	2023
Module:	Msc Research Project
Supervisor:	Prof. Aaloka Anant
Submission Due Date:	14-12-2023
Project Title:	Jax base machine learning models for early breast cancer detection
Word Count:	1357
Page Count:	38

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature	Nikhil Kadam

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Nikhil Kadam
Student ID:22110712

1 Introduction

This Configuration Manual lists together all prerequisites needed to duplicate the studies and their effects on a specific setting. A glimpse of the source for Exploratory Data analysis for both tabular and image data and images augmentation is done followed by label encoding, class balancing using Smote, and feature engineering and after that all the algorithms are created, and Evaluations is also supplied, together with the necessary hardware components as well as Software applications. The report is organized as follows, with details relating environment configuration provided in Section 2.

Information about data collection is detailed in Section 3. Exploratory Data Analysis is done in Section 4. Label Encoding and Class Balancing is included in Section 5. In section 6, the Feature Engineering is described. Section 7 provides details of data preprocessing and image augmentation. Details about models that were created and evaluated are provided in Section 8. How the results are calculated and shown is described in Section 9.

2 System Requirements

The specific needs for hardware as well as software to put the research into use are detailed in this section.

2.1 Hardware Requirements

The necessary hardware specs are shown in Figure 1 below. MacOs M1 Chip, macOS 10.15.x (Catalilna) operating system, 8GB RAM, 256GB Storage, 24'' Display.

Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 (Version 4.8.0)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

2.3 Code Execution

The code can be run in jupyter notebook. The jupyter notebook comes with Anaconda 3, run the jupyter notebook from startup. This will open jupyter notebook in web browser. The web browser will show the folder structure of the system, move to the folder where the code file is located. Open the code file from the folder and to run the code, go to Kernel menu and run all cells.

3 Data Gathering

The dataset is collected from https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic for tabular data. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. The data is also taken from https://www.kaggle.com/datasets/aryashah2k/breast-ultrasound-images-dataset for images. The data collected at baseline include breast ultrasound images among women in ages between 25 and 75 years old. This data was collected in 2018. The number of patients is 600 female patients. The dataset consists of 780 images with an average image size of 500*500 pixels.

4 Exploratory Data Analysis

Figure 2 includes a list of every Python library necessary to complete the project.

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from collections import Counter
from imblearn.over_sampling import SMOTE
import xgboost as xgb
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
import jax
import jax.numpy as jnp
from jax import grad, jit, vmap
from jax import random
from jax.scipy.special import logsumexp
import glob
import shutil
import os
import cv2
from sklearn import preprocessing
import random
from sklearn.utils import class_weight
import tensorflow
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv1D, Layer,Attention, GlobalAveragePooling2D
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
from keras.applications.inception_v3 import InceptionV3
from keras.applications.densenet import DenseNet121
```

Figure 2: Necessary Python libraries

Figure 3 represents the block of code to import data as pandas' data frame and print top 10 rows of the data.

```python
df = pd.read_csv("data.csv")
```

```python
df.head(10)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	te
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	
5	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	...	
6	844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	...	
7	84458202	M	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	...	
8	844981	M	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	...	
9	84501001	M	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	...	

10 rows × 33 columns

Figure 3: Data import

As seen in Figure 4, the column names and information of the data.

```
df.columns
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Figure 4: Data information

In figure 5, the code to generate data statistics.

```
df.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800

Figure 5: Data Statistics

The Figure 6, illustrate the plot for the value counts in target column for each class.

```
df['diagnosis'].value_counts().plot(kind='barh')
df['diagnosis'].value_counts() #357 Benign and 212 Malignant
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

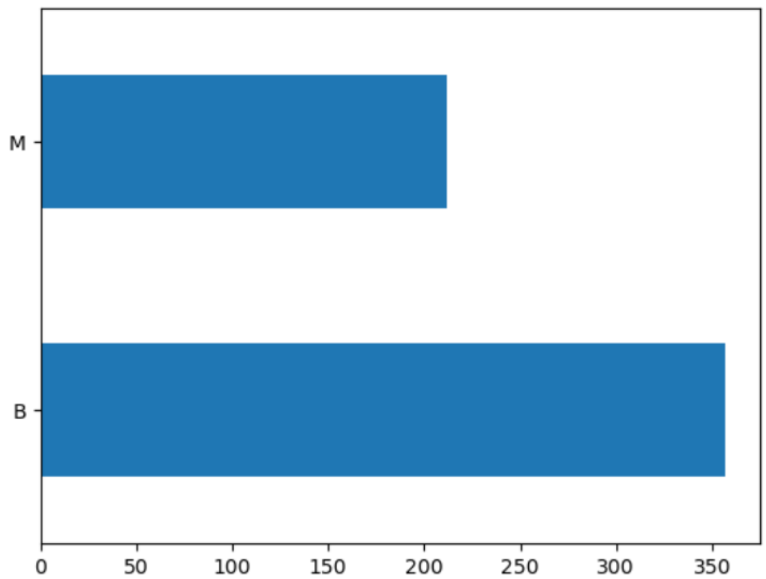


Figure 6: Class Count

Figure 7 includes a code segment to check for missing values in each column and treatment.

```
df.isnull().sum()
```

```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

```
df.dropna(axis=1, inplace=True)
df.drop('id', axis=1, inplace=True)
df
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	ar
0	M	17.99	10.38	122.80	
1	M	20.57	17.77	132.90	
2	M	19.69	21.25	130.00	
3	M	11.42	20.38	77.58	
4	M	20.29	14.34	135.10	
...	...	...	...	...	
564	M	21.56	22.39	142.00	
565	M	20.13	28.25	131.20	
566	M	16.60	28.08	108.30	
567	M	20.60	29.33	140.10	
568	B	7.76	24.54	47.92	

569 rows × 31 columns

Figure 7: Missing Value Treatment

The Figure 8 represents the block of code to generate the list of features.

```
features = df.columns
print(features)
features = features.drop('diagnosis')
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

```
len(features)
```

```
30
```

Figure 8: Features

As seen in Figure 9, the box plot for all column to check for outliers.

```
fig, axs = plt.subplots(len(features)//10,10,figsize=(18, 6),sharey='row')
for id, feature in enumerate(features):
    ax = sns.boxplot(data=df,x=feature,y='diagnosis',showfliers = False, orient='h', ax=axs[id//10,id%10])
    ax.set(ylabel=None)
    ax.xaxis.set_label_position('top')
    axs[id//10,0].set_yticklabels(['Benign','Malignant'], rotation=90, va='center')
fig.tight_layout()
plt.show()
```

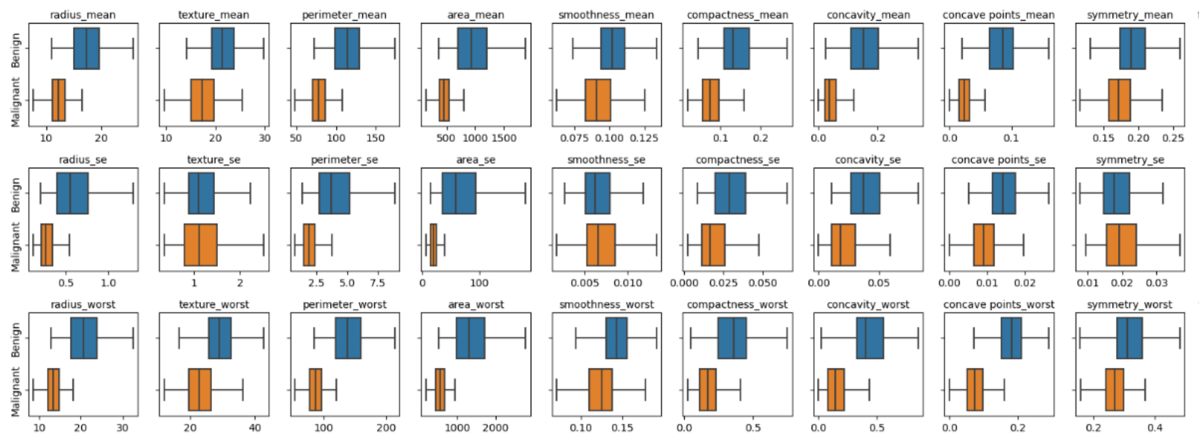


Figure 9: Box Plot

In figure 10, the code to generate heatmap for correlations.

```
plt.figure(figsize=(20, 10))
df2 = df.copy()
df2['diagnosis'] = df2['diagnosis'].map({'B':0,'M':1}).astype('category')
heatmap = sns.heatmap(df2.corr(numeric_only=False), annot=True)
```

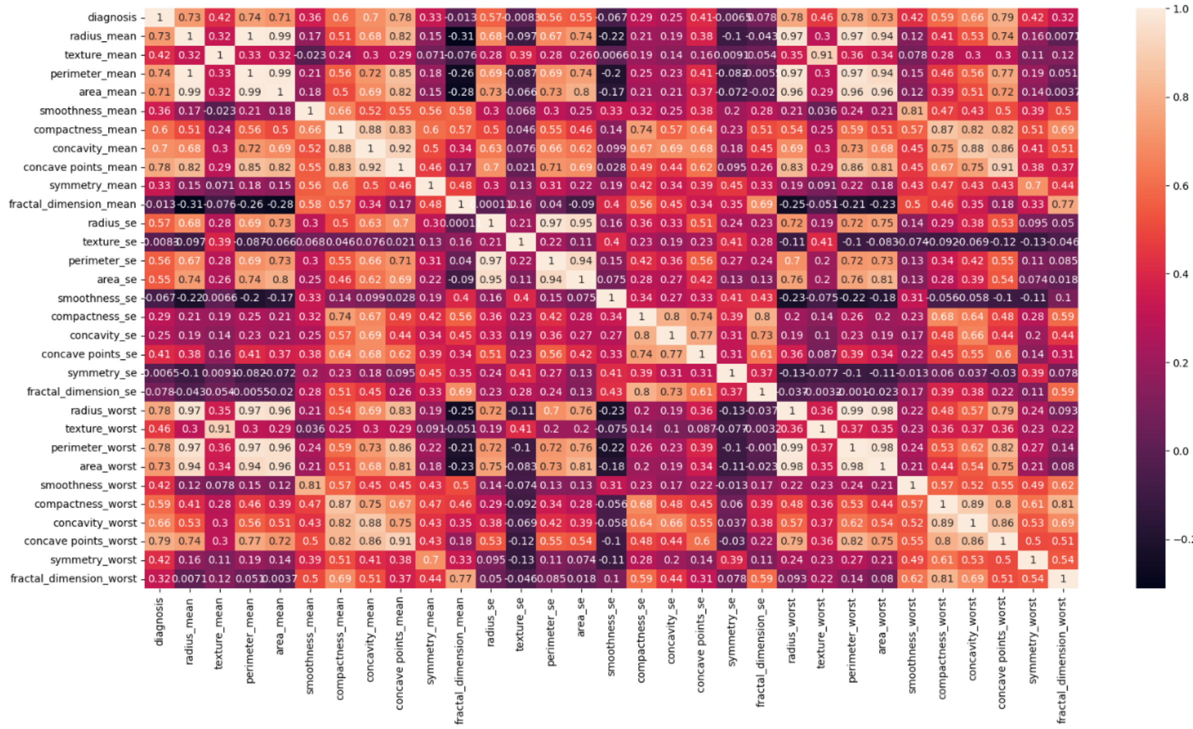


Figure 10: Correlation heatmap

5 Label Encoding and Class Balancing

The Figure 11, illustrate the section to display data information before label encoding.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   diagnosis                569 non-null    object 
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

Figure 11: Data Information

The Figure 12, illustrate the label encoder and data information after encoding.

```
le = LabelEncoder()
```

```
df['diagnosis'] = le.fit_transform(df['diagnosis'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   diagnosis                569 non-null    int32  
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

Figure 12: Label Encoder

The Figure 13, illustrate the class value count pie chart.

```
target = df['diagnosis'].value_counts()
plt.pie(target, labels=target.index, autopct='%0.2f%%')
```

```
([<matplotlib.patches.Wedge at 0x16d24f03e90>,
  <matplotlib.patches.Wedge at 0x16d24e787d0>],
 [Text(-0.4286546999573329, 1.0130425204326268, '0'),
  Text(0.428654605109445, -1.013042560566172, '1')],
 [Text(-0.23381165452218156, 0.5525686475087055, '62.74%'),
  Text(0.23381160278696997, -0.55256866939973, '37.26%')])
```

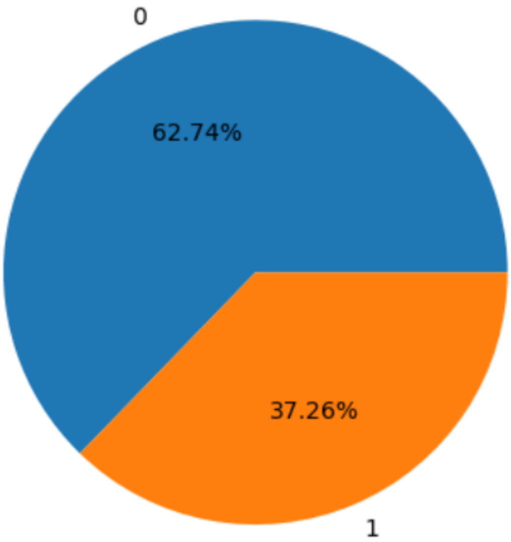


Figure 13: Class Value count

The Figure 14, illustrate the separation of features and target columns int o x and y and checking their shape.

```
x = df[features]
y = df['diagnosis']
```

```
x.shape
```

```
(569, 30)
```

```
y.shape
```

```
(569,)
```

Figure 14: Features and Target

Figure 15 illustrates the use of SMOTE for class balancing.

```
sm = SMOTE()
print("Before ", Counter(y))
```

```
Before  Counter({0: 357, 1: 212})
```

```
x, y = sm.fit_resample(x, y)
print("After ", Counter(y))
```

```
After  Counter({1: 357, 0: 357})
```

Figure 15: SMOTE

6 Feature Engineering

Figure 16 illustrates the numeric pipeline generation and creation of column transformer.

```
numeric_pipeline = Pipeline(
    steps=[("scale", StandardScaler())]
)
```

```
full_processor = ColumnTransformer(
    transformers=[
        ("numeric", numeric_pipeline, x.columns),
    ]
)
```

Figure 16: Generating Pipeline and Transformer

Figure 17 illustrates the code to process features and divide them into training and test set of data.

```
X_processed = full_processor.fit_transform(x)
X_train, X_test, y_train, y_test = train_test_split(X_processed, y,test_size=0.3, stratify=y)
```

```
X_processed.shape
```

```
(714, 30)
```

Figure 17: Processing features

Figure 18 illustrates creating hyper parameters list for XGBoost classifier for feature selection and checking for best score and parameter set.

```
param_grid = {
    "max_depth": [1, 2, 3, 4, 5, 7],
    "learning_rate": [0.5, 0.1, 0.01, 0.05],
    "gamma": [0, 0.25, 1, 3, 5, 7],
    "reg_lambda": [0, 1, 10],
    "scale_pos_weight": [1, 3, 5],
    "subsample": [0.8],
    "colsample_bytree": [0.5],
}
```

```
xgb_cl = xgb.XGBClassifier(objective="binary:logistic")
grid_cv = GridSearchCV(xgb_cl, param_grid, n_jobs=-1, cv=3, scoring="roc_auc")

_ = grid_cv.fit(X_processed, y)
```

```
grid_cv.best_score_
```

```
0.9985405927076713
```

```
grid_cv.best_params_
```

```
{'colsample_bytree': 0.5,
 'gamma': 0,
 'learning_rate': 0.5,
 'max_depth': 3,
 'reg_lambda': 1,
 'scale_pos_weight': 1,
 'subsample': 0.8}
```

Figure 18: XGBoost

Figure 19 illustrates the best features and checking for their probability scores.

```
final_cl = xgb.XGBClassifier(
    **grid_cv.best_params_,
    objective="binary:logistic",
)

_ = final_cl.fit(X_train, y_train)
```

```
feature_important = final_cl.feature_importances_
```

```
feature_important
```

```
array([0.        , 0.01664119, 0.00072673, 0.01058594, 0.0044788 ,
       0.00849016, 0.05354065, 0.04669156, 0.00058317, 0.00208098,
       0.00340098, 0.00082663, 0.00308148, 0.01196237, 0.00391021,
       0.0019248 , 0.00460635, 0.00069372, 0.00065432, 0.00929699,
       0.21085519, 0.01033519, 0.17262805, 0.05468515, 0.01082895,
       0.        , 0.03338491, 0.31545332, 0.00348293, 0.00416933],
      dtype=float32)
```

```
keys = x.columns.values
values = feature_important

data = pd.DataFrame(data=values, index=keys, columns=["score"]).sort_values(by="score", ascending=False)
data.nlargest(10, columns="score").sort_values('score', ascending=True).plot(kind='barh', figsize = (20,10))
```

```
<Axes: >
```

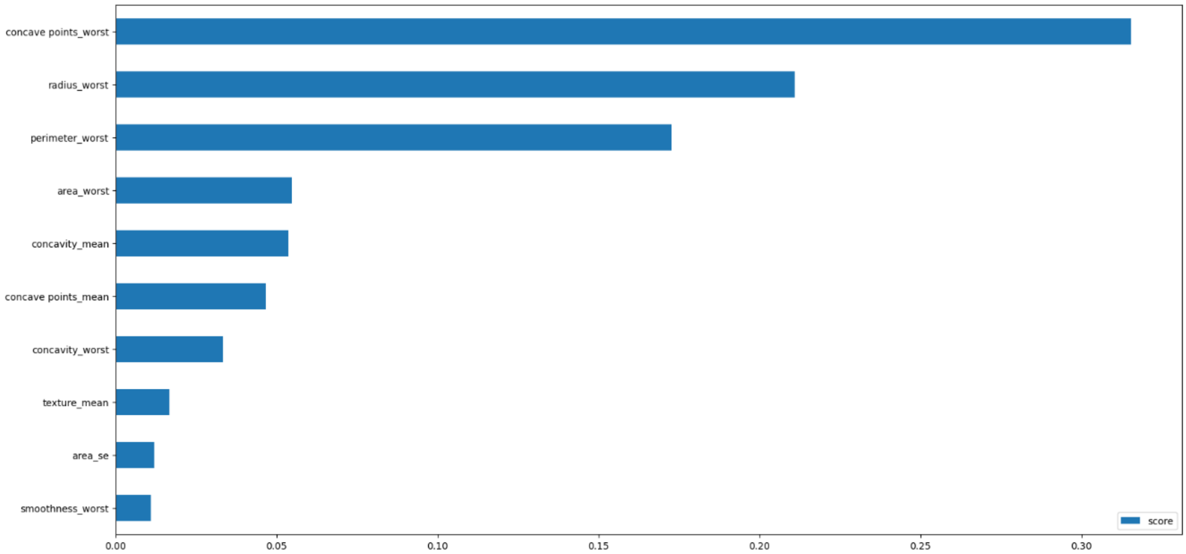


Figure 19: Important features list

Figure 20 illustrates creating a list of 10 most important features and choosing those in final feature set. Also, the code shows train test split of the data to be used for model training and performance evaluation.

```python
imp_columns = data.nlargest(10, columns="score").sort_values('score', ascending=True).index.to_list()
imp_columns
```

```
['smoothness_worst',
 'area_se',
 'texture_mean',
 'concavity_worst',
 'concave points_mean',
 'concavity_mean',
 'area_worst',
 'perimeter_worst',
 'radius_worst',
 'concave points_worst']
```

```python
x = x[imp_columns]
x
```

	smoothness_worst	area_se	texture_mean	concavity_worst	concave points_mean	concavity_mean	area_worst	perimeter_wo
0	0.162200	153.400000	10.380000	0.711900	0.147100	0.300100	2019.000000	184.6000
1	0.123800	74.080000	17.770000	0.241600	0.070170	0.086900	1956.000000	158.8000
2	0.144400	94.030000	21.250000	0.450400	0.127900	0.197400	1709.000000	152.5000
3	0.209800	27.230000	20.380000	0.686900	0.105200	0.241400	567.700000	98.8700
4	0.137400	94.440000	14.340000	0.400000	0.104300	0.198000	1575.000000	152.2000
...	...	...	...	...	...	...	...	
709	0.137862	40.301547	27.495868	0.327036	0.050219	0.108520	922.712133	113.1012
710	0.142983	25.659590	16.024874	0.285916	0.051020	0.081618	814.935143	109.8899
711	0.158447	17.299787	20.903153	0.409592	0.058531	0.094108	746.083507	104.7227
712	0.154742	34.903080	18.428205	0.438002	0.074444	0.133986	1091.050853	125.7038
713	0.174951	46.810645	20.406514	0.512486	0.105888	0.225173	919.690962	118.8139

714 rows × 10 columns

```python
scaler = StandardScaler()
scaler.fit(x)
x = scaler.transform(x)
```

```python
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
```

```python
perfEvaluation = pd.DataFrame()
```

Figure 20: Feature selection and Training and testing data split

7 Image Preprocessing and Augmentation

Figures 21 show the code to create path variable for categories of image and checking for image count.

```
pathbenign = './archive/Dataset_BUSI_with_GT/benign/'
pathmalignant = './archive/Dataset_BUSI_with_GT/malignant/'
pathnormal = './archive/Dataset_BUSI_with_GT/normal/'
```

```
categories = ['Benign', 'Malignant', 'Normal']
print(categories)
```

```
['Benign', 'Malignant', 'Normal']
```

```
#initialization and importing for data analysi
count_bengin=len(os.listdir(pathbenign))
count_malignant=len(os.listdir(pathmalignant))
count_normal=len(os.listdir(pathnormal))
count_bengin, count_malignant, count_normal
```

```
(891, 421, 266)
```

Figure 21: Path setup

Figures 22 show the code to create plot of image count.

```
#plotting graph for cancer cell types count
fig = plt.figure(figsize = (10, 5))
values=[count_bengin, count_malignant, count_normal]
#creating the bar plot
plt.bar(categories, values, color ='blue', width = 0.4)

plt.xlabel("Classes")
plt.ylabel("Number of Images")
plt.show()
```

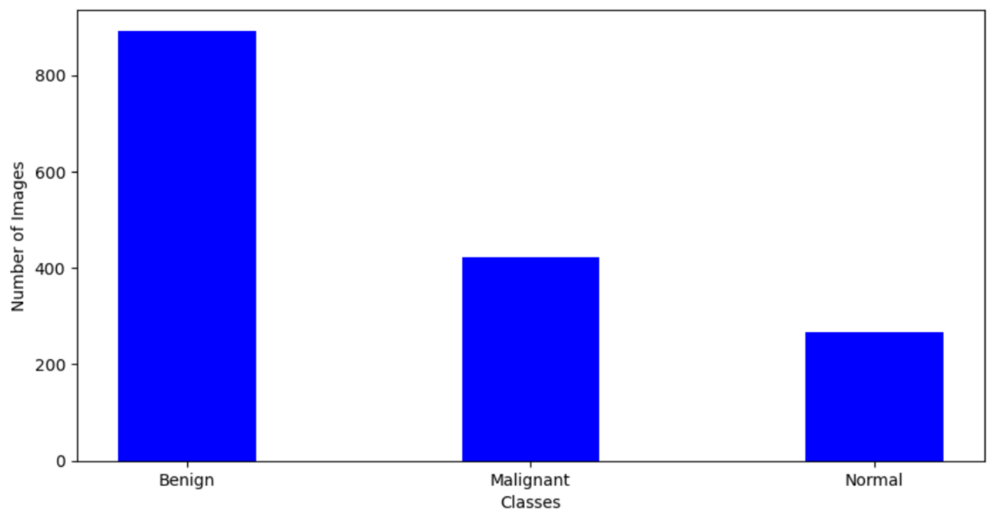


Figure 22: Image count

Figure 23 illustrates the code to create a list of images in each category from the folder path.

.

```
benign = glob.glob(pathbenign+ "*.png")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(benign))
print ('\n'.join(benign[:5]))
```

```
Total of 891  images.
First 5 filenames:
./archive/Dataset_BUSI_with_GT/benign\benign (1).png
./archive/Dataset_BUSI_with_GT/benign\benign (1)_mask.png
./archive/Dataset_BUSI_with_GT/benign\benign (10).png
./archive/Dataset_BUSI_with_GT/benign\benign (10)_mask.png
./archive/Dataset_BUSI_with_GT/benign\benign (100).png
```

```
malignant = glob.glob(pathmalignant + "*.png")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(malignant))
print ('\n'.join(malignant[:5]))
```

```
Total of 421  images.
First 5 filenames:
./archive/Dataset_BUSI_with_GT/malignant\malignant (1).png
./archive/Dataset_BUSI_with_GT/malignant\malignant (1)_mask.png
./archive/Dataset_BUSI_with_GT/malignant\malignant (10).png
./archive/Dataset_BUSI_with_GT/malignant\malignant (10)_mask.png
./archive/Dataset_BUSI_with_GT/malignant\malignant (100).png
```

```
normal = glob.glob(pathnormal + "*.png")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(normal))
print ('\n'.join(normal[:5]))
```

```
Total of 266  images.
First 5 filenames:
./archive/Dataset_BUSI_with_GT/normal\normal (1).png
./archive/Dataset_BUSI_with_GT/normal\normal (1)_mask.png
./archive/Dataset_BUSI_with_GT/normal\normal (10).png
./archive/Dataset_BUSI_with_GT/normal\normal (10)_mask.png
./archive/Dataset_BUSI_with_GT/normal\normal (100).png
```

Figure 23: Image list

The Figure 24, illustrate the code to use ImageDataGenerator to generate augmented images for the deep learning models.

```
path = './archive/Dataset_BUSI_with_GT/'
img_size = 100
```

```
gen = ImageDataGenerator(rescale=1./255)
```

```
train = gen.flow_from_directory(directory=path, target_size=(img_size,img_size), class_mode='sparse')
```

```
Found 1578 images belonging to 3 classes.
```

```
# This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 4, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()

augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

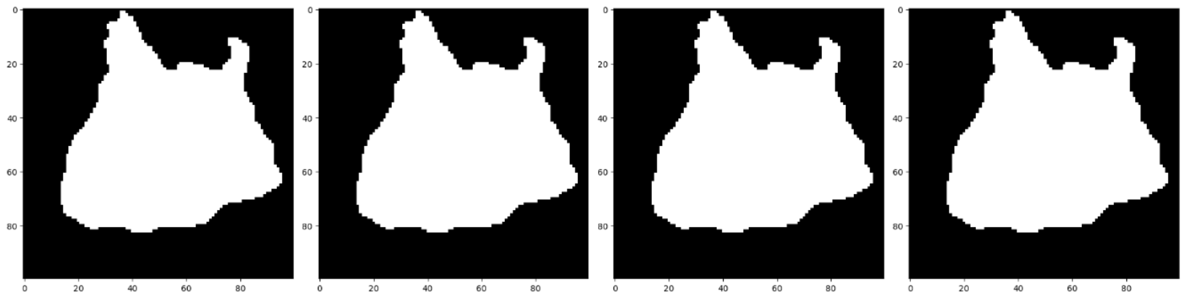


Figure 24: Image Data Generator

Figures 25 show the code to create training data with width and height shift the images.

```
gen = ImageDataGenerator(rescale=1./255,  width_shift_range=0.2, height_shift_range=0.3)
train = gen.flow_from_directory(directory=path, target_size=(img_size,img_size), class_mode='sparse')
```

```
Found 1578 images belonging to 3 classes.
```

```
augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

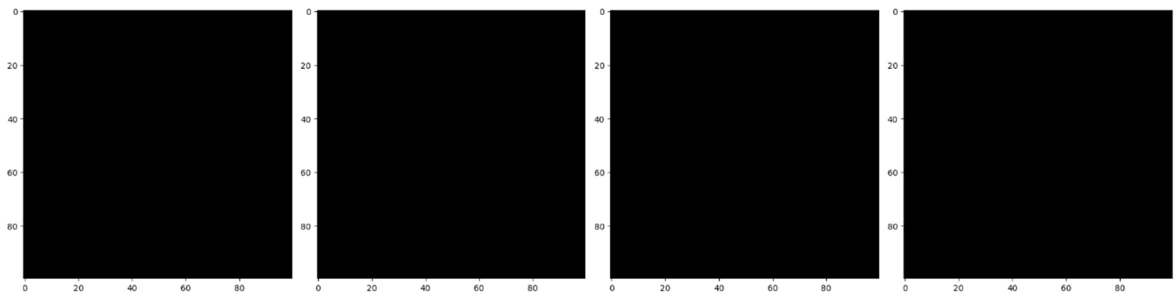


Figure 25: Image Data Generator

Figures 26 show the code to create testing data with rescaling the images.

```
gen = ImageDataGenerator(rescale=1./255)
train = gen.flow_from_directory(directory=path, target_size=(img_size,img_size))
```

```
Found 1578 images belonging to 3 classes.
```

```
augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

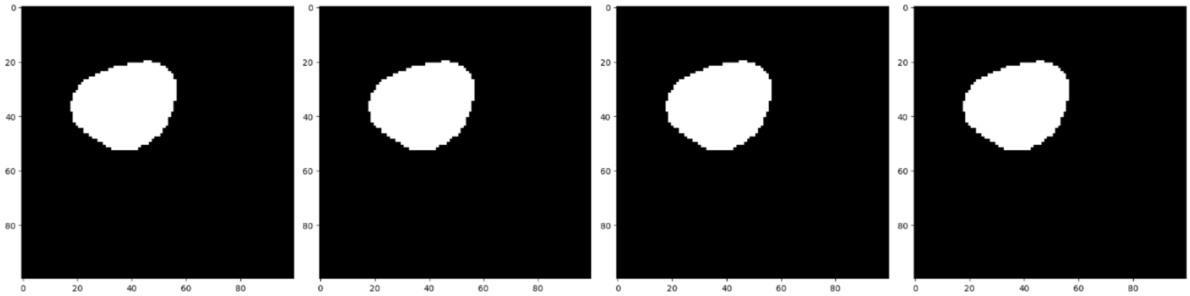


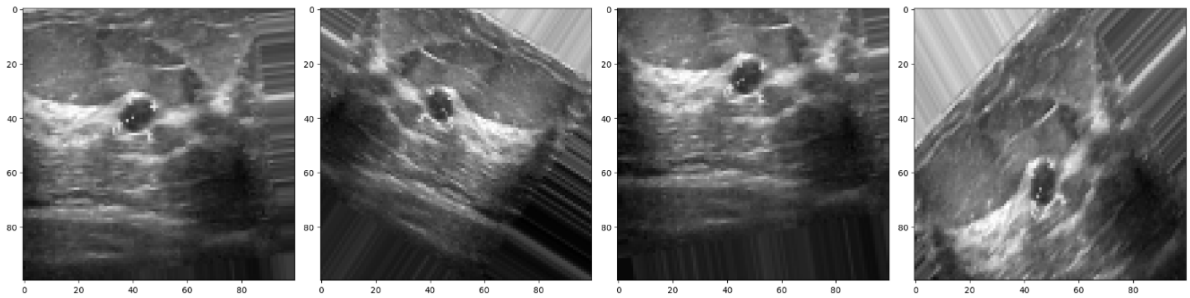
Figure 26: Image Data Generator

Figures 27 show the code to create training and testing data by augmenting the images.

```
batch_size = 32
datagen= ImageDataGenerator(rescale=1./255, validation_split=0.2, rotation_range=40, width_shift_range=0.2,zoom_range=0.2,
                    height_shift_range=0.2, shear_range=0.2, horizontal_flip=True,  fill_mode='nearest')
```

```
train = datagen.flow_from_directory(path, target_size=(img_size, img_size), batch_size=batch_size, subset='training', shuffle=Tru
        classes=categories)
augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

```
Found 1263 images belonging to 3 classes.
```



```
test = datagen.flow_from_directory(path, target_size=(img_size, img_size), batch_size=batch_size, subset='validation',
        shuffle=True, classes=categories)
augmented_images = [test[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

```
Found 315 images belonging to 3 classes.
```

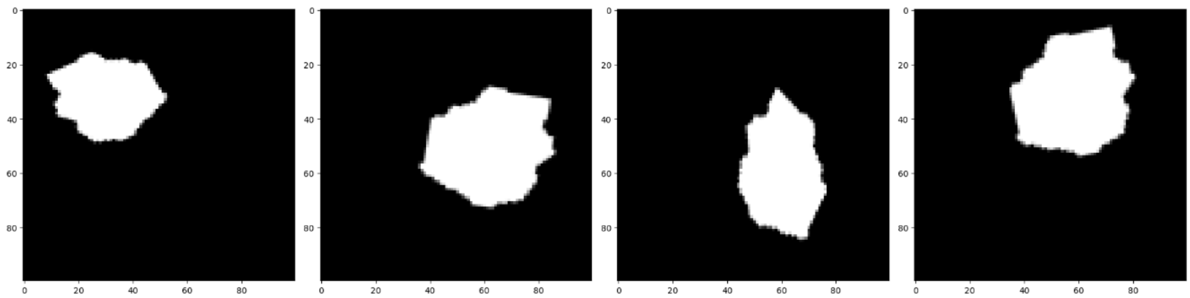


Figure 27: Image Data Generator

8 Machine Learning Models

8.1 Logistic Regression

```
clf = LogisticRegression(random_state=123, max_iter=100).fit(X_train, y_train)
```

```
ypred =  clf.predict(X_test)
```

```
acc = np.round(accuracy_score(y_test, ypred)*100, 2)
acc
```

```
98.88
```

```
f1 = np.round(f1_score(y_test, ypred)*100, 2)
f1
```

```
98.91
```

```
prec = np.round(precision_score(y_test, ypred)*100, 2)
prec
```

```
100.0
```

```
recall = np.round(recall_score(y_test, ypred)*100, 2)
recall
```

```
97.85
```

```
print(classification_report(y_test, ypred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        86
           1       1.00      0.98      0.99        93

    accuracy                           0.99       179
   macro avg       0.99      0.99      0.99       179
weighted avg       0.99      0.99      0.99       179
```

Figure 28: Implementation of Logistic Regression

8.2 Neural Network

```
clf = MLPClassifier(hidden_layer_sizes=(128,), activation='tanh', max_iter=1000).fit(X_train, y_train)
```

```
ypred =  clf.predict(X_test)
```

```
acc = np.round(accuracy_score(y_test, ypred)*100, 2)
acc
```

```
98.32
```

```
f1 = np.round(f1_score(y_test, ypred)*100, 2)
f1
```

```
98.38
```

```
prec = np.round(precision_score(y_test, ypred)*100, 2)
prec
```

```
98.91
```

```
recall = np.round(recall_score(y_test, ypred)*100, 2)
recall
```

```
97.85
```

```
print(classification_report(y_test, ypred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98        86
           1       0.99      0.98      0.98        93

    accuracy                           0.98       179
   macro avg       0.98      0.98      0.98       179
weighted avg       0.98      0.98      0.98       179
```

Figure 29: Implementation of Neural Network

8.3 Logistic Regression JAX

```
y_train = y_train.to_numpy()
```

```
def logistic(r):
    return 1 / (1 + jnp.exp(-r))
```

```
def predict(c, w, X):
    return logistic(jnp.dot(X, w) + c)
```

```
c_0 = 1.50
w_0 = 9.0e-1 * jnp.ones(10)
```

```
ypred = np.round(predict(c_0, w_0, X_test))
```

```
acc = np.round(accuracy_score(y_test, ypred)*100, 2)
acc
```

```
97.77
```

```
f1 = np.round(f1_score(y_test, ypred)*100, 2)
f1
```

```
97.85
```

```
prec = np.round(precision_score(y_test, ypred)*100, 2)
prec
```

```
97.85
```

```
recall = np.round(recall_score(y_test, ypred)*100, 2)
recall
```

```
97.85
```

```
print(classification_report(y_test, ypred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        86
           1       0.98      0.98      0.98        93

    accuracy                           0.98       179
   macro avg       0.98      0.98      0.98       179
weighted avg       0.98      0.98      0.98       179
```

Figure 30: Implementation of Logistic Regression JAX

8.4 Neural Network JAX

```
def relu(x):
    return jnp.maximum(0, x)

def predict(c, w, X):
    outputs = jnp.dot(X, w) + c
    activations = relu(outputs)
    return activations
```

```
def cost(c, w, X, y, eps=3e-2, lmbd=0.1):
    n = y.size
    p = predict(c, w, X)
    p = jnp.clip(p, eps, 1 - eps)  # bound the probabilities within (0,1) to avoid ln(0)
    return -jnp.sum(y * jnp.log(p) + (1 - y) * jnp.log(1 - p)) / n + 0.5 * lmbd * (
        jnp.dot(w, w) + c * c
    )
```

```
%%time
n_iter = 100
eta = 2e-2
tol = 5e-6
w = w_0
c = c_0
new_cost = float(cost(c, w, X_train, y_train))
cost_hist = [new_cost]
for i in range(n_iter):
    c_current = c
    c -= eta * grad(cost, argnums=0)(c_current, w, X_train, y_train)
    w -= eta * grad(cost, argnums=1)(c_current, w, X_train, y_train)
    new_cost = float(cost(c, w, X_train, y_train))
    cost_hist.append(new_cost)
    if (i > 20) and (i % 10 == 0):
        if jnp.abs(cost_hist[-1] - cost_hist[-20]) < tol:
            print(f"Exited loop at iteration {i}")
            break
```

```
CPU times: total: 4.88 s
Wall time: 5.61 s
```

Figure 31: Implementation of Neural Network JAX

```python
ypred = np.round(predict(c, w, X_test))
```

```python
acc = np.round(accuracy_score(y_test, ypred)*100, 2)
acc
```

```
49.72
```

```python
f1 = np.round(f1_score(y_test, ypred,average='weighted')*100, 2)
f1
```

```
51.18
```

```python
prec = np.round(precision_score(y_test, ypred,average='weighted')*100, 2)
prec
```

```
87.97
```

```python
recall = np.round(recall_score(y_test, ypred,average='weighted')*100, 2)
recall
```

```
C:\Users\SHILPA\AppData\Roaming\Python\Python311\site-packages\sklearn\me
Recall is ill-defined and being set to 0.0 in labels with no true samples
or.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
49.72
```

```python
print(classification_report(y_test, ypred))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98        86
         1.0       0.80      0.04      0.08        93
         2.0       0.00      0.00      0.00         0
         3.0       0.00      0.00      0.00         0
         4.0       0.00      0.00      0.00         0
         5.0       0.00      0.00      0.00         0
         6.0       0.00      0.00      0.00         0
         7.0       0.00      0.00      0.00         0
         8.0       0.00      0.00      0.00         0
         9.0       0.00      0.00      0.00         0
        10.0       0.00      0.00      0.00         0
        11.0       0.00      0.00      0.00         0
        12.0       0.00      0.00      0.00         0
        14.0       0.00      0.00      0.00         0
        15.0       0.00      0.00      0.00         0
        16.0       0.00      0.00      0.00         0
        17.0       0.00      0.00      0.00         0
        19.0       0.00      0.00      0.00         0

    accuracy                           0.50       179
   macro avg       0.10      0.06      0.06       179
weighted avg       0.88      0.50      0.51       179
```

Figure 32: Implementation of Neural Network JAX

8.5 CNN

```
model = Sequential()
model.add(Conv2D(64, (3, 3), activation='tanh', input_shape=(img_size, img_size, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.01))
model.add(Flatten())
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(3, activation='sigmoid'))
model.compile(optimizer='sgd', loss='categorical_crossentropy',  metrics=['accuracy'])
```

```
callback = EarlyStopping(monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, shuffle = True, callbacks=[callback])
```

```
Epoch 1/10
40/40 [==============================] - 42s 1s/step - loss: 0.9973 - accuracy: 0.5471 - val_loss: 0.9724 - val_accuracy: 0.565
1
Epoch 2/10
40/40 [==============================] - 21s 527ms/step - loss: 0.9451 - accuracy: 0.5590 - val_loss: 0.9502 - val_accuracy: 0.
5651
Epoch 2: early stopping
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x16d271a8510>
```

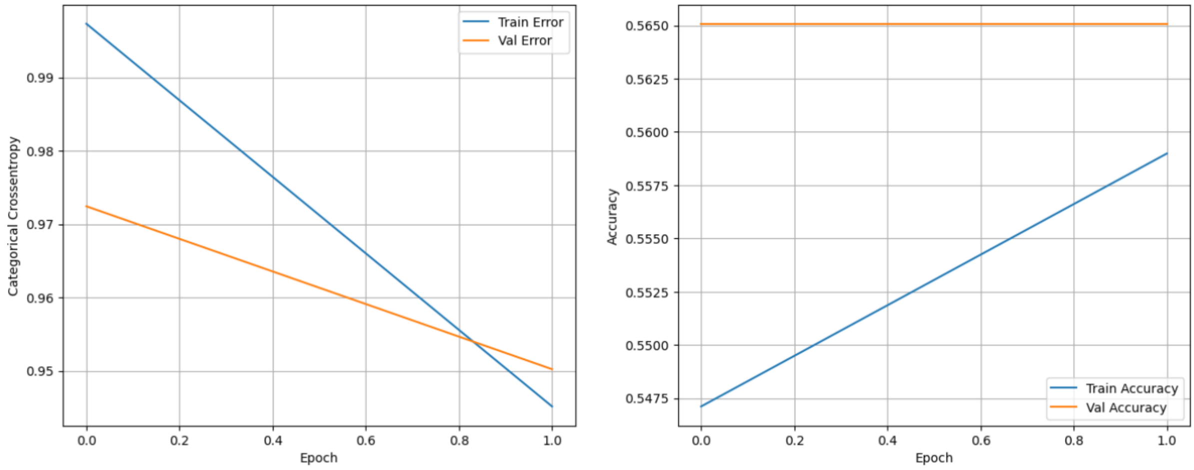


Figure 33: Implementation of CNN

8.6 InceptionNet

```python
# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(img_size, img_size, 3))
```

```python
# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(3, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
    layer.trainable = True
```

```python
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 100, 100, 3)]  0           []

 conv2d_1 (Conv2D)              (None, 49, 49, 32)   864         ['input_1[0][0]']

 batch_normalization (Batch     (None, 49, 49, 32)   96          ['conv2d_1[0][0]']
 Normalization)

 activation (Activation)        (None, 49, 49, 32)   0           ['batch_normalization[0][0]']

 conv2d_2 (Conv2D)              (None, 47, 47, 32)   9216        ['activation[0][0]']

 batch_normalization_1 (Bat     (None, 47, 47, 32)   96          ['conv2d_2[0][0]']
 chNormalization)

 activation_1 (Activation)      (None, 47, 47, 32)   0           ['batch_normalization_1[0][0]'
                                                                 ]

 conv2d_3 (Conv2D)              (None, 47, 47, 64)   18432       ['activation_1[0][0]']
```

```python
callback = EarlyStopping( monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, shuffle = True, callbacks=[callback])
```

```
Epoch 1/10
40/40 [==============================] - 137s 3s/step - loss: 0.9313 - accuracy: 0.6215 - val_loss: 1.9058 - val_accuracy: 0.52
06
Epoch 2/10
40/40 [==============================] - 113s 3s/step - loss: 0.6633 - accuracy: 0.7308 - val_loss: 10.8329 - val_accuracy: 0.5
841
Epoch 3/10
40/40 [==============================] - 108s 3s/step - loss: 0.6297 - accuracy: 0.7458 - val_loss: 65.8379 - val_accuracy: 0.3
492
Epoch 3: early stopping
```

```python
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x16d3101b190>
```

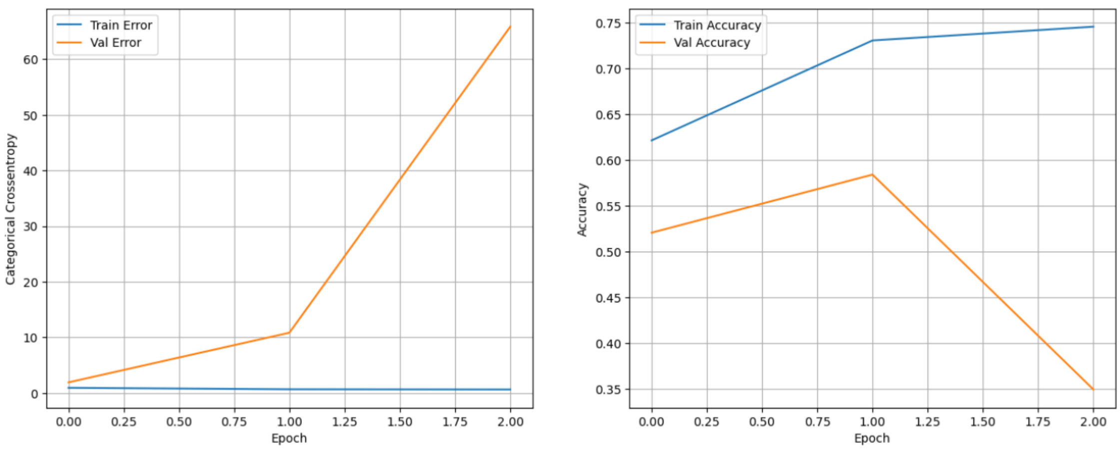


Figure 34: Implementation of InceptioNet

8.7 DenseNet121

```
model = DenseNet121(include_top=False, weights="imagenet", input_shape=(img_size, img_size, 3))
model.summary()
```

```
Model: "densenet121"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 100, 100, 3)]  0         []

 zero_padding2d (ZeroPaddin     (None, 106, 106, 3)  0           ['input_2[0][0]']
 g2D)

 conv1/conv (Conv2D)            (None, 50, 50, 64)   9408        ['zero_padding2d[0][0]']

 conv1/bn (BatchNormalizati     (None, 50, 50, 64)   256         ['conv1/conv[0][0]']
 on)

 conv1/relu (Activation)        (None, 50, 50, 64)   0           ['conv1/bn[0][0]']

 zero_padding2d_1 (ZeroPadd     (None, 52, 52, 64)   0           ['conv1/relu[0][0]']
 ing2D)
```

```
denseNet = model.output
denseNet = Flatten()(denseNet)
denseNet = Dense(256, activation='relu')(denseNet)
denseNet = Dropout(0.02)(denseNet)
output_layer = Dense(3, activation='tanh')(denseNet)

model = Model(inputs=model.input, outputs=output_layer)
```

```
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 100, 100, 3)]  0         []

 zero_padding2d (ZeroPaddin     (None, 106, 106, 3)  0           ['input_2[0][0]']
 g2D)

 conv1/conv (Conv2D)            (None, 50, 50, 64)   9408        ['zero_padding2d[0][0]']

 conv1/bn (BatchNormalizati     (None, 50, 50, 64)   256         ['conv1/conv[0][0]']
 on)

 conv1/relu (Activation)        (None, 50, 50, 64)   0           ['conv1/bn[0][0]']

 zero_padding2d_1 (ZeroPadd     (None, 52, 52, 64)   0           ['conv1/relu[0][0]']
 ing2D)
```

```
callback = EarlyStopping( monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, shuffle = True, callbacks=[callback])
```

```
Epoch 1/10
40/40 [==============================] - 239s 5s/step - loss: 3.9191 - accuracy: 0.5590 - val_loss: 4.2982 - val_accuracy: 0.56
51
Epoch 2/10
40/40 [==============================] - 197s 5s/step - loss: 4.3007 - accuracy: 0.5645 - val_loss: 4.2982 - val_accuracy: 0.56
51
Epoch 2: early stopping
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x16d00083910>
```

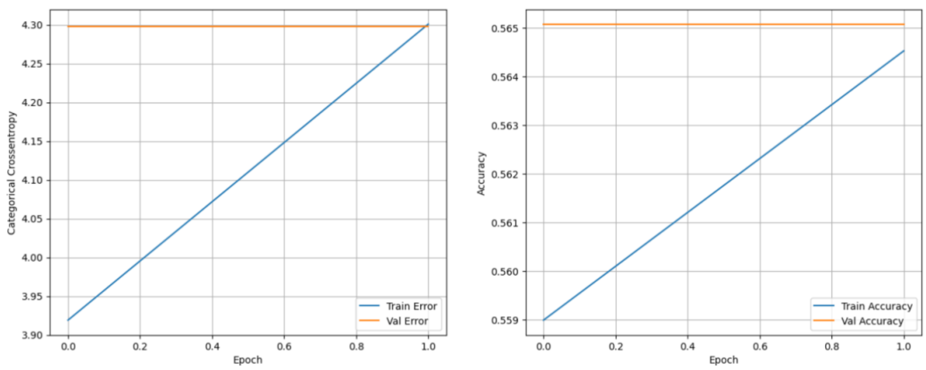


Figure 35: Implementation of DenseNet121

8.8 VGG19

```
model = VGG19(include_top=False,  weights='imagenet',  input_shape=(img_size, img_size, 3))
model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_3 (InputLayer)        [(None, 100, 100, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 100, 100, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 100, 100, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 50, 50, 64)        0         
                                                                 
 block2_conv1 (Conv2D)       (None, 50, 50, 128)       73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 50, 50, 128)       147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 25, 25, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 25, 25, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 25, 25, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 25, 25, 256)       590080    
                                                                 
 block3_conv4 (Conv2D)       (None, 25, 25, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 12, 12, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 12, 12, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 12, 12, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 12, 12, 512)       2359808   
                                                                 
 block4_conv4 (Conv2D)       (None, 12, 12, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 6, 6, 512)         0         
                                                                 
 block5_conv1 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_conv4 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 3, 3, 512)         0         
                                                                 
=================================================================
Total params: 20024384 (76.39 MB)
Trainable params: 20024384 (76.39 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Unfreeze the last few layers for fine-tuning
for layer in model.layers[:-4]:
    layer.trainable = False

# apply Global Average Pooling to the last layer of the pretrained model
x = GlobalAveragePooling2D()(model.output)
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)

model = Model(inputs=model.input, outputs=predictions)
```

Figure 36: Implementation of VGG19

```
model.compile(optimizer = 'adam', loss= 'binary_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 100, 100, 3)]     0

 block1_conv1 (Conv2D)       (None, 100, 100, 64)      1792

 block1_conv2 (Conv2D)       (None, 100, 100, 64)      36928

 block1_pool (MaxPooling2D)  (None, 50, 50, 64)        0

 block2_conv1 (Conv2D)       (None, 50, 50, 128)       73856

 block2_conv2 (Conv2D)       (None, 50, 50, 128)       147584

 block2_pool (MaxPooling2D)  (None, 25, 25, 128)       0

 block3_conv1 (Conv2D)       (None, 25, 25, 256)       295168

 block3_conv2 (Conv2D)       (None, 25, 25, 256)       590080

 block3_conv3 (Conv2D)       (None, 25, 25, 256)       590080

 block3_conv4 (Conv2D)       (None, 25, 25, 256)       590080

 block3_pool (MaxPooling2D)  (None, 12, 12, 256)       0

 block4_conv1 (Conv2D)       (None, 12, 12, 512)       1180160

 block4_conv2 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_conv3 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_conv4 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 6, 6, 512)         0

 block5_conv1 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv2 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv3 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv4 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 3, 3, 512)         0

 global_average_pooling2d_4  (None, 512)               0
  (GlobalAveragePooling2D)

 flatten_3 (Flatten)         (None, 512)               0

 dense_7 (Dense)             (None, 512)               262656

 dense_8 (Dense)             (None, 3)                 1539

=================================================================
Total params: 20288579 (77.39 MB)
Trainable params: 7343619 (28.01 MB)
Non-trainable params: 12944960 (49.38 MB)
_________________________________________________________________
```

```
callback = EarlyStopping( monitor='val_accuracy', patience=1, verbose=1, mode='max')
history = model.fit(train, epochs=10, validation_data=test, callbacks=[callback])
```

```
Epoch 1/10
40/40 [==============================] - 151s 4s/step - loss: 0.5393 - accuracy: 0.6041 - val_loss: 0.8241 - val_accuracy: 0.71
75
Epoch 2/10
40/40 [==============================] - 181s 5s/step - loss: 0.3603 - accuracy: 0.7387 - val_loss: 0.3684 - val_accuracy: 0.72
38
Epoch 3/10
40/40 [==============================] - 179s 4s/step - loss: 0.3104 - accuracy: 0.7593 - val_loss: 0.3232 - val_accuracy: 0.78
10
Epoch 4/10
40/40 [==============================] - 185s 5s/step - loss: 0.2836 - accuracy: 0.8044 - val_loss: 0.3063 - val_accuracy: 0.78
```

Figure 37: Implementation of VGG19

```python
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Categorical Crossentropy')
ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
ax1.grid()
ax1.legend()
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
ax2.grid()
ax2.legend()
```

```
<matplotlib.legend.Legend at 0x16d46533910>
```

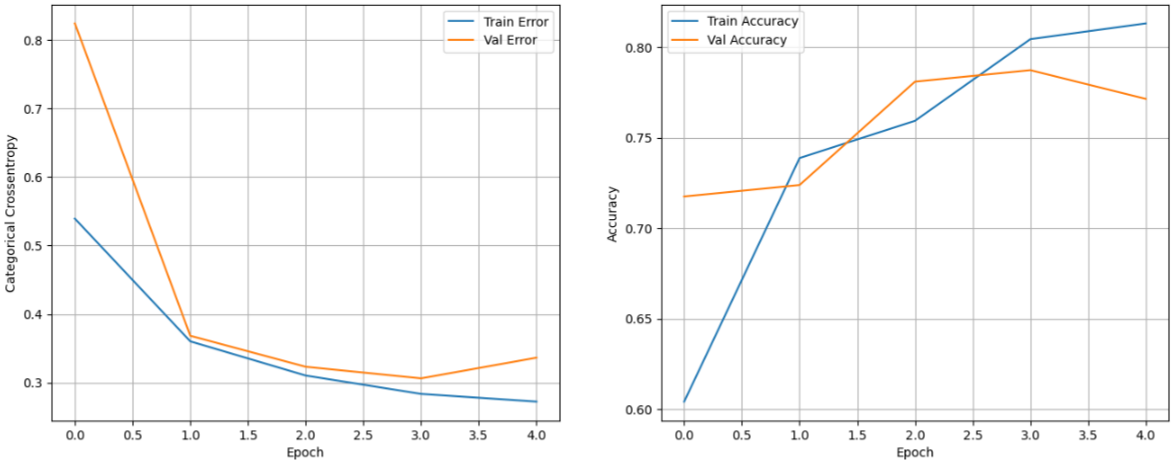


Figure 38: Implementation of VGG19

9 Model result

This section explains the performance of the models.

9.1 Model Scores

```
modelScores.columns = ['Models', 'Accuracy']
```

```
modelScores
```

	Models	Accuracy
0	CNN	56.507939
0	InceptionNet	30.793652
0	Dense Net	56.507939
0	VGG19	76.507938

Figure 39: Model Performance Image

```
plt.figure(figsize=(8,5))
plt.bar(modelScores['Models'], modelScores['Accuracy'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3],
 [Text(0, 0, 'CNN'),
  Text(1, 0, 'InceptionNet'),
  Text(2, 0, 'Dense Net'),
  Text(3, 0, 'VGG19')])
```

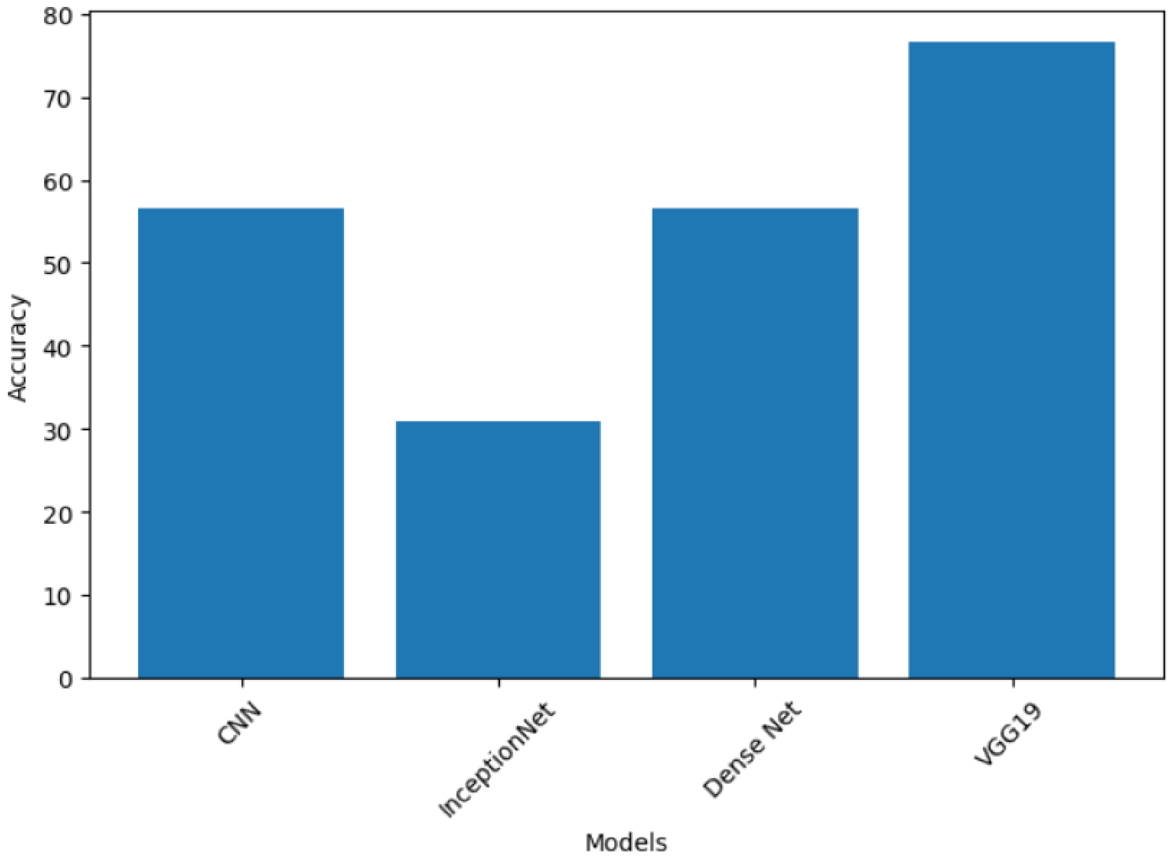


Figure 40: Model Performance Image

```python
perfEvaluation.columns =['Model', 'Accuracy', 'F1-Score', 'Precision', 'Recall']
perfEvaluation
```

	Model	Accuracy	F1-Score	Precision	Recall
0	Logistic Regression ML	98.88	98.91	100.00	97.85
0	Neural Network ML	98.32	98.38	98.91	97.85
0	Logistic Regression JAX	97.77	97.85	97.85	97.85
0	Neural Network JAX	49.72	51.18	87.97	49.72

```python
plt.plot(perfEvaluation['Model'], perfEvaluation['Accuracy'])
plt.bar(perfEvaluation['Model'], perfEvaluation['Accuracy'])
plt.xlabel('Models')
plt.ylabel('Scores')
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3],
 [Text(0, 0, 'Logistic Regression ML'),
  Text(1, 0, 'Neural Network ML'),
  Text(2, 0, 'Logistic Regression JAX'),
  Text(3, 0, 'Neural Network JAX')])
```

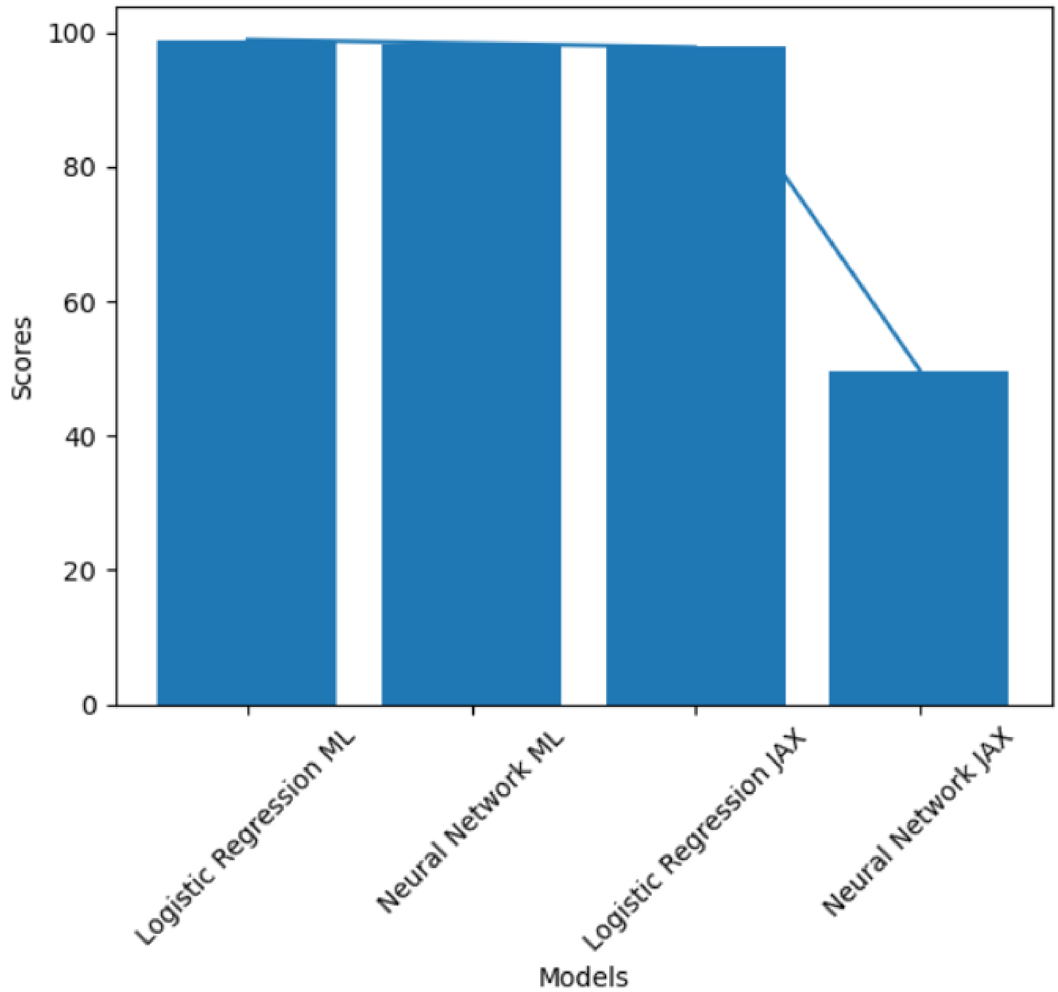


Figure 41: Model Performance Data

```
plt.plot(perfEvaluation['Model'], perfEvaluation['F1-Score'])
plt.bar(perfEvaluation['Model'], perfEvaluation['F1-Score'])
plt.xlabel('Models')
plt.ylabel('Scores')
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3],
 [Text(0, 0, 'Logistic Regression ML'),
  Text(1, 0, 'Neural Network ML'),
  Text(2, 0, 'Logistic Regression JAX'),
  Text(3, 0, 'Neural Network JAX')])
```

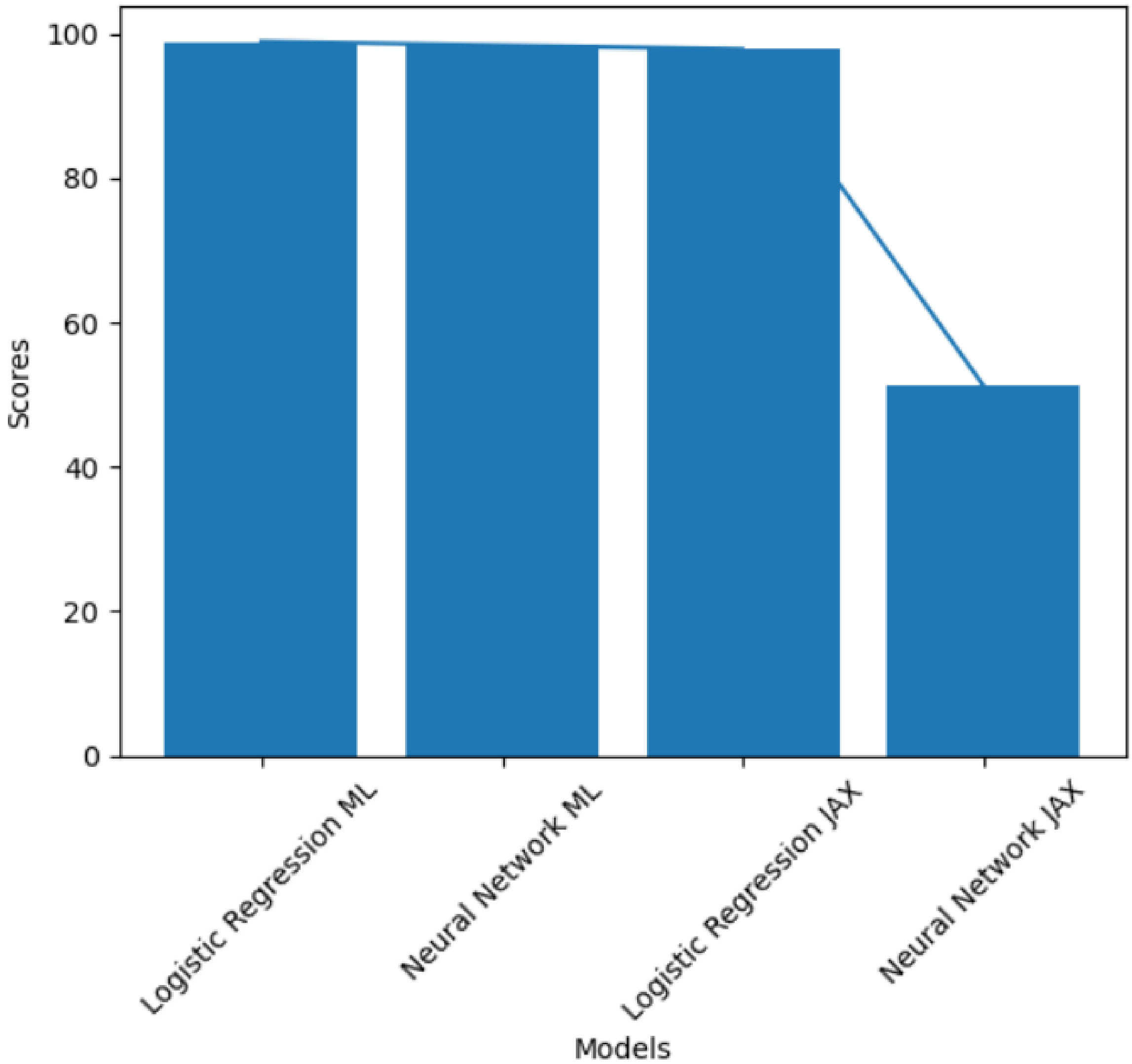


Figure 42: Model Performance Data

```
plt.plot(perfEvaluation['Model'], perfEvaluation['Precision'])
plt.bar(perfEvaluation['Model'], perfEvaluation['Precision'])
plt.xlabel('Models')
plt.ylabel('Scores')
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3],
 [Text(0, 0, 'Logistic Regression ML'),
  Text(1, 0, 'Neural Network ML'),
  Text(2, 0, 'Logistic Regression JAX'),
  Text(3, 0, 'Neural Network JAX')])
```

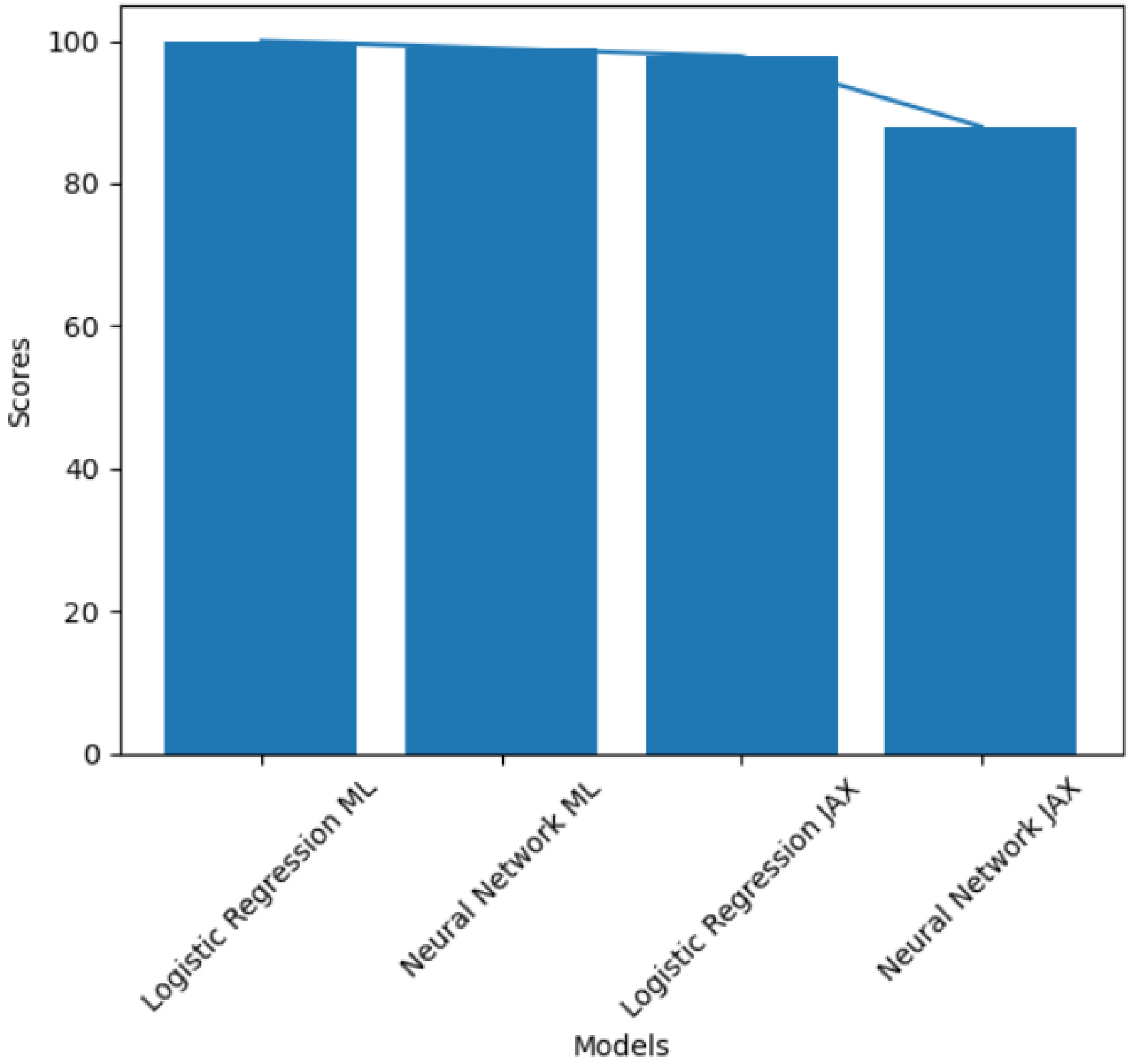


Figure 43: Model Performance Data

References

UCI Machine Learning Repository

Breast Ultrasound Images Dataset (kaggle.com)

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm

https://www.geeksforgeeks.org/introduction-convolution-neural-network/