Data Analysis and Machine Learning

Assignment EE-514

5th December 2022

Udai Sharma

22261555

Introduction

The assignment mainly comprises of 2 parts which are further divided into subsections. The following provides an insight into the main idea the question puts ahead of us and the technique used to tackle the problem for an optimized solution.

Part I:

The first part primarily focuses on data wrangling and statistical machine learning techniques. Based on the ExtraSensory dataset, this assignment's focus is on leveraging sensor data to forecast human activity such as walking using attributes of watch acceleration and raw acceleration. The dataset contains information retrieved using smartphone and smartwatch sensors placed on 60 users. Each user is uniquely identified by an UUID (Universally unique identifier). The users involved in the research problem belonged to distinct classifications and carried a variety of mobile devices and high frequency motion-reactive sensors that were sampled in low frequency (once in a minute).

The 1st part started with importing libraries such as pandas, numpy, matplotlib, sklearn, statistics and pathlib into the jupyter notebook and then loading data of a given user UUID and selecting an attribute. Looking at the accelerometer attribute which are given by watch acceleration (watch_acceleration) and raw acceleration (raw_acc). We input our user data into the Logistic regression model and test it on five more users to validate the model's power to predict whether the user is walking or not. Following that, the data was split in an 80:20 ratio. The data from the training set takes up 80% and is used to train the model by feeding it into the algorithm of our model. The validation set which takes up the 20% part of the data is used to assess the performance of our model followed by a support vector classifier (svc) alongside changed hyperparameter which were 'sigmoid' and 'rbf' kernel. For each model used, a classification report was generated and a ROC curve was also plotted for the logistic regression model and the support vector classifier.

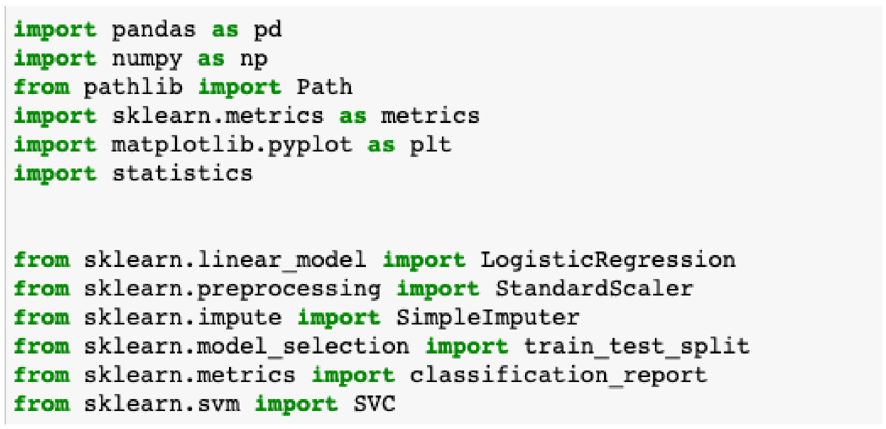
Part II:

Part II of the assignment involves training a deep convolution neural network called ResNet50 on low-resolution images of Mars' surface. Fans and blotches are the two parts that require our attention. To begin, we train the model for 5 epochs in order to generate and report the validation and training losses. The "epochs" hyperparameter specifies how many times the learning algorithm will run over the entire training dataset. Every sample in the training dataset has had one epoch to update the internal model parameters. To gain a better understanding of the deep CNN, the learning curve after training is plotted.

Part I: Extrasensory

1. Importing Libraries

Python libraries are essential in areas such as data science, machine learning, data manipulation applications, and so on. The library incorporates modules that provide access to system functionality such as file I/O that would otherwise be absent.



```
import pandas as pd
import numpy as np
from pathlib import Path
import sklearn.metrics as metrics
import matplotlib.pyplot as plt
import statistics


from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.svm import SVC
```

- *from sklearn.model_selection import train_test_split*

  To assess how well our machine learning model performs, we must classify a dataset into train and test sets. The train set is used to fit the model, and its statistics are known. The second set is identified as the test data set, and it is only used for predictions.

- *from sklearn.metrics import classification_report*

  A classification report is a report that encompasses metrics about how well a machine learning model performed. It provides us with the potential to obtain results such as f1-score, recall, precision and assists in positive and negative prediction as well. It also aids us in our calculation of accuracy, macro and weighted averages

- *from sklearn.svm import SVC*

The SVC (Support Vector Classifier) is an SVM-based classifier that can be used in classification problems. It is also known as C-SVC because it optimizes the margin in hyperplane using the C hyper-parameter.

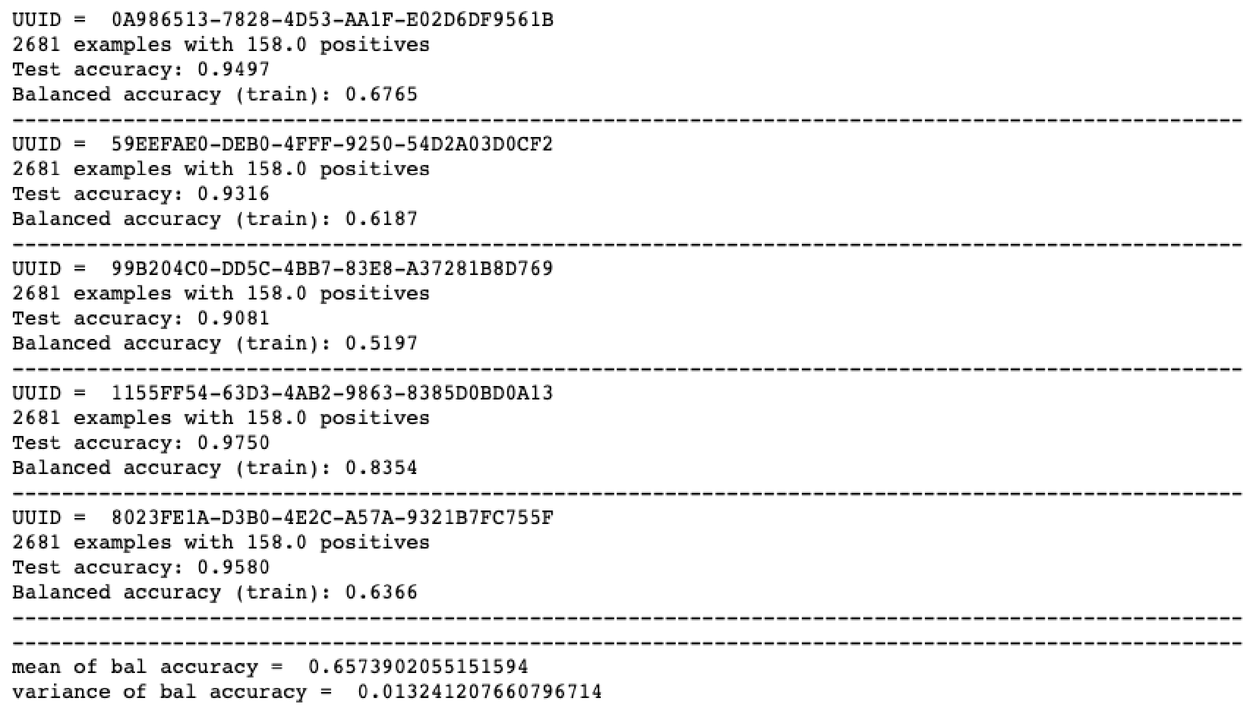
2. **Improving Test set**

   The model was trained on a single user and then tested on 5 users individually. Further, ***mean*** and ***variance*** of the balanced accuracy for the 5 users were obtaine as ***0.6573*** and ***0.0132*** respectively. After running the model through its paces, it was discovered that the balancing accuracy of all five user data points was above 0.5%, indicating that the developed model functioned admirably. However, the third user's data was on monotonous with a value of 0.5197, indicating that the data was not entirely accurate.

```
user_list=
['0A986513-7828-4D53-AA1F-E02D6DF9561B','59EEFAE0-DEB0-4FFF-9250-54D2A03D0CF2','99B204C0-DD5C-4BB7-83E8-A37281B8D769','1155FF54-63D3-4AB2-9863-8385D0BD0A13','8023FE1A-D3B0-4E2C-A57A-9321B7FC755F']
acc_list=[]

def make_list(x) :
  acc_list.append(x)

for user in user_list:
  df_test = load_data_for_user(user)
  X_test, y_test = get_features_and_target(df_test, acc_sensors, target_column)
  print("UUID = ",user)
  print(f'{y_train.shape[0]} examples with {y_train.sum()} positives')
  X_test = imputer.transform(scaler.transform(X_test))
  print(f'Test accuracy: {clf.score(X_test, y_test):0.4f}')
  y_pred = clf.predict(X_test)
```

```
    print(f'Balanced accuracy (train): {metrics.balanced_accuracy_score(y_test,
y_pred):0.4f}')
    x = metrics.balanced_accuracy_score(y_test, y_pred)
    make_list(x)
    print('-'*100)

print('-'*100)
#print(acc_list)
y = sum(acc_list)
y_mean = y/len(acc_list)
print("mean of bal accuracy = ",y_mean)

y_var = statistics.variance(acc_list)
print("variance of bal accuracy = ",y_var)
```

Result:

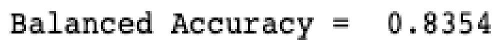


```
UUID =  0A986513-7828-4D53-AA1F-E02D6DF9561B
2681 examples with 158.0 positives
Test accuracy: 0.9497
Balanced accuracy (train): 0.6765
----------------------------------------------------------------------------------------------------
UUID =  59EEFAE0-DEB0-4FFF-9250-54D2A03D0CF2
2681 examples with 158.0 positives
Test accuracy: 0.9316
Balanced accuracy (train): 0.6187
----------------------------------------------------------------------------------------------------
UUID =  99B204C0-DD5C-4BB7-83E8-A37281B8D769
2681 examples with 158.0 positives
Test accuracy: 0.9081
Balanced accuracy (train): 0.5197
----------------------------------------------------------------------------------------------------
UUID =  1155FF54-63D3-4AB2-9863-8385D0BD0A13
2681 examples with 158.0 positives
Test accuracy: 0.9750
Balanced accuracy (train): 0.8354
----------------------------------------------------------------------------------------------------
UUID =  8023FE1A-D3B0-4E2C-A57A-9321B7FC755F
2681 examples with 158.0 positives
Test accuracy: 0.9580
Balanced accuracy (train): 0.6366
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
mean of bal accuracy =  0.6573902055151594
variance of bal accuracy =  0.013241207660796714
```

3. Validation Data and increasing the training data

```
merged_csv = pd.concat([pd.read_csv(data_dir + '/' + (user + '.features_labels.csv'))
for user in user_list])
df_merged = pd.DataFrame(merged_csv)
df_merged.to_csv("combine_csv.csv", index=False, encoding='utf-8-sig')
df_mergedset = pd.read_csv(data_dir + '/' + "combine_csv.csv" )
```

Result :



```
Balanced Accuracy =  0.8354
```

We can use balanced accuracy to evaluate the performance of a classification model. Balanced accuracy is calculated as: (Sensitivity + Specificity) / 2. where,

Sensitivity is the percentage of positive cases that the model can detect ("true positive rate"), while specificity is the percentage of negative cases that the model can detect ("true negative rate").

This metric is especially beneficial when the two classes are imbalanced, implying that one class appears remarkably more than the other.

The model's balanced accuracy comes out to be 0.5958.

It is worth noting that the model's ability to correctly classify observations decreases as the balanced accuracy decreases.

In this case, the balanced accuracy is quite average, indicating that the logistic regression model does an average job of predicting using the data.

4. Model Selection and Model Testing:

SVM is a supervised machine learning algorithm that can be used for both classification and regression.

Then, in each case, it is used to train on a Logistic Regression model once with C-parameters of 1.0 and 0.6 to see the optimisation in variance, and a classification report is generated.

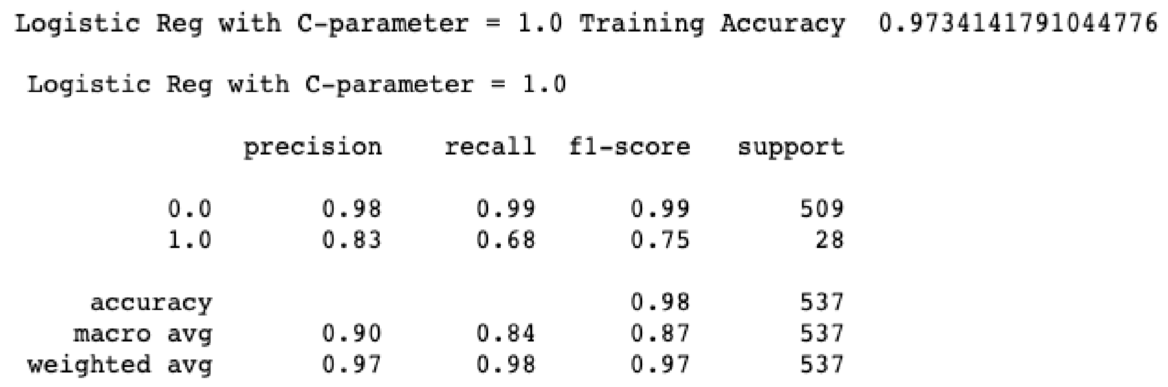
We generate classification reports for two Support Vector Classifiers, rbf and sigmoid, using hyper-parameter kernels.

Sigmoid Kernel: This function is equivalent to a two-layer neural network perceptron model, and it is used as an activation function for artificial neurons.

Rbf Kernel: The radial basis function kernel, or RBF kernel, is a popular kernel function in machine learning that is used in a variety of kernelized learning algorithms.

```
clf = LogisticRegression(solver='liblinear', max_iter=1000, C=1.0)

clf.fit(X_train, y_train)

print('Logistic Reg with C-parameter = 1.0 Training accuracy : ',clf.score(X_train, y_train))

print('Logistic Reg with C-parameter = 1.0\n' )

print(classification_report(y_val,y_pred))
```

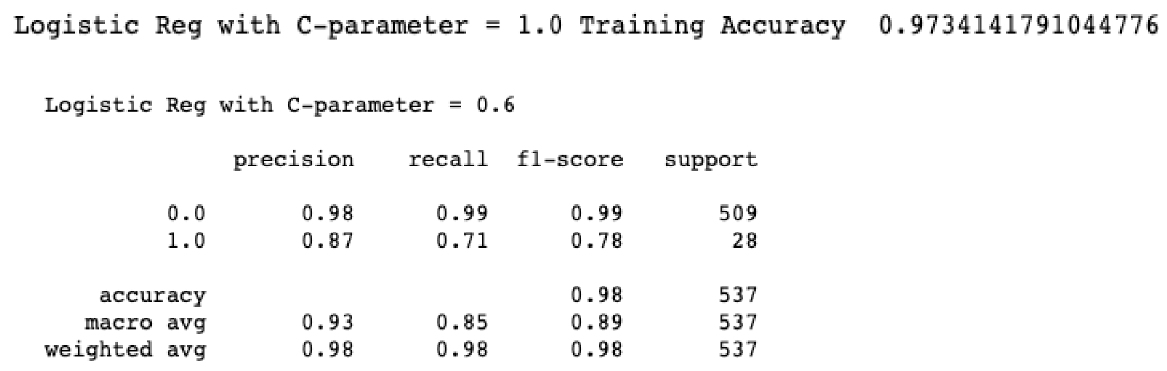
Result:



```
Logistic Reg with C-parameter = 1.0 Training Accuracy  0.9734141791044776

 Logistic Reg with C-parameter = 1.0

              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99       509
         1.0       0.83      0.68      0.75        28

    accuracy                           0.98       537
   macro avg       0.90      0.84      0.87       537
weighted avg       0.97      0.98      0.97       537
```

```
clf = LogisticRegression(solver='liblinear', max_iter=1000, C=0.6)
clf.fit(X_train, y_train)
print(f'Logistic Reg with C=0.6 Training Accuracy : {clf.score(X_train,y_train):0.4f}')
print('Logistic Reg with C-parameter = 0.6\n' )
print(classification_report(y_val,y_pred))
```

Result:



```
Logistic Reg with C-parameter = 1.0 Training Accuracy  0.9734141791044776

  Logistic Reg with C-parameter = 0.6

              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99       509
         1.0       0.87      0.71      0.78        28

    accuracy                           0.98       537
   macro avg       0.93      0.85      0.89       537
weighted avg       0.98      0.98      0.98       537
```

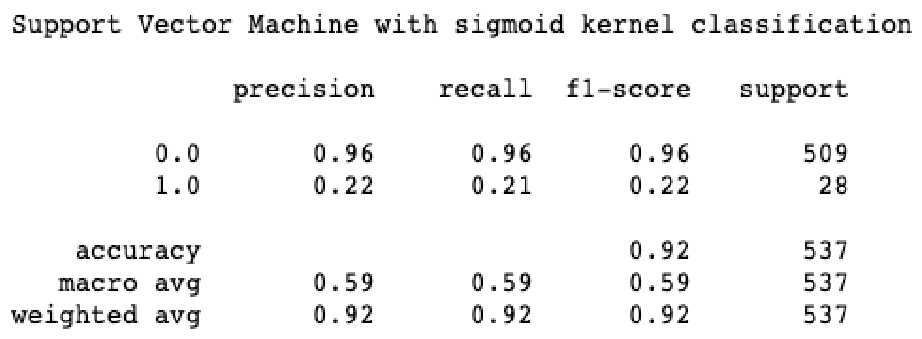
The following table represents the Balanced Accuracy and Training Accuracy result against the Hyperparameter ‘C’ :

C-Parameter	1.0	0.6
Balanced Accuracy	0.8354	0.8542
Training Accuracy	0.9729	0.9734

```
clf = SVC(C=0.1,kernel = 'sigmoid', gamma= 1)
clf.fit(X_train, y_train)
X_val = scaler.fit_transform(X_val)
X_val = imputer.fit_transform(X_val)
```

```
y_pred = clf.predict(X_val)
print(f'Balanced Accuracy =  {metrics.balanced_accuracy_score(y_val, y_pred):0.4f}')
print(f'Support Vector Machine with sigmoid kernel classification\n')
print(classification_report(y_val,y_pred))
```

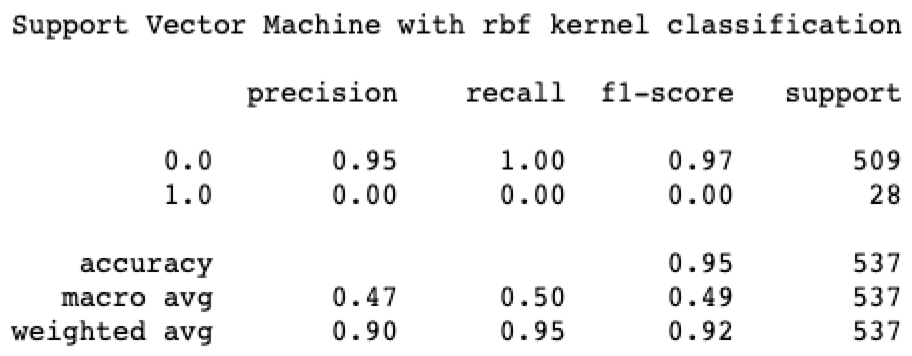
Result:



```
Support Vector Machine with sigmoid kernel classification

              precision    recall  f1-score   support

         0.0       0.96      0.96      0.96       509
         1.0       0.22      0.21      0.22        28

    accuracy                           0.92       537
   macro avg       0.59      0.59      0.59       537
weighted avg       0.92      0.92      0.92       537
```

```
clf = SVC(C=0.1,kernel = 'rbf', gamma= 1)
clf.fit(X_train, y_train)
X_val = scaler.fit_transform(X_val)
X_val = imputer.fit_transform(X_val)
y_pred = clf.predict(X_val)
print(f'Balanced Accuracy =  {metrics.balanced_accuracy_score(y_val, y_pred):0.4f}')
```

Result:



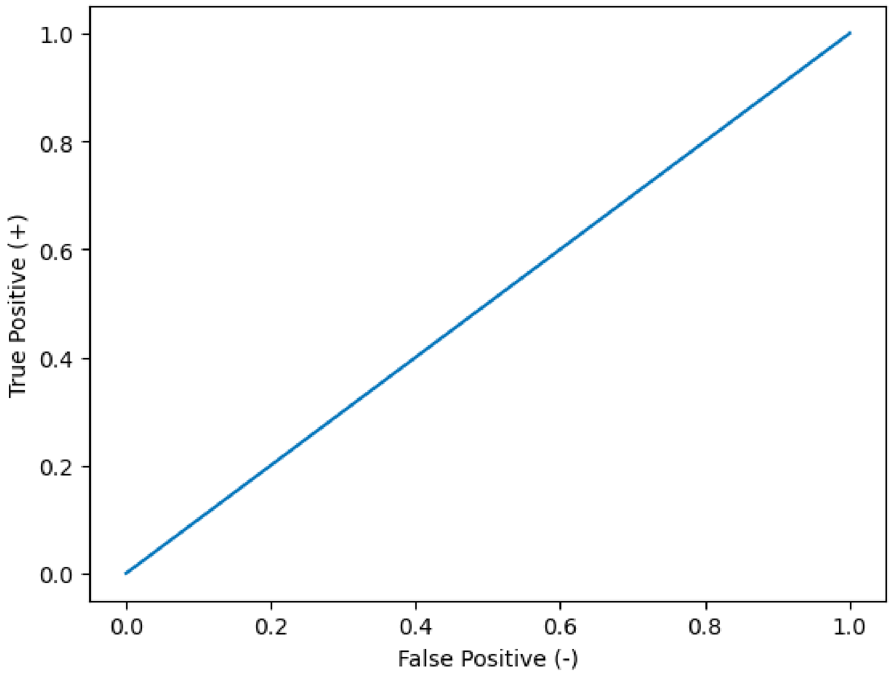
```
Support Vector Machine with rbf kernel classification

              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97       509
         1.0       0.00      0.00      0.00        28

    accuracy                           0.95       537
   macro avg       0.47      0.50      0.49       537
weighted avg       0.90      0.95      0.92       537
```

5. Plotting the ROC curve

Plotted the ROC curve for Logistic Regression C = 1.0 and SVC with kernel = ‘sigmoid’. Also, calculated the value of AUC in each case.

```
fpr, tpr, _ = metrics.roc_curve(y_val,  y_pred)

auc = metrics.roc_auc_score(y_val, y_pred)


plt.plot(fpr,tpr)

plt.ylabel('True Positive (+)')

plt.xlabel('False Positive (-)')

plt.show()
```

Result:



Part II: Planet Four

Residual neural networks (RNNs) are artificial neural networks (ANNs) that build networks by stacking residual blocks. ResNet is an abbreviation for Residual Network, which is a type of convolutional neural network.

ResNet-50 is a convolutional neural network with 50 layers.

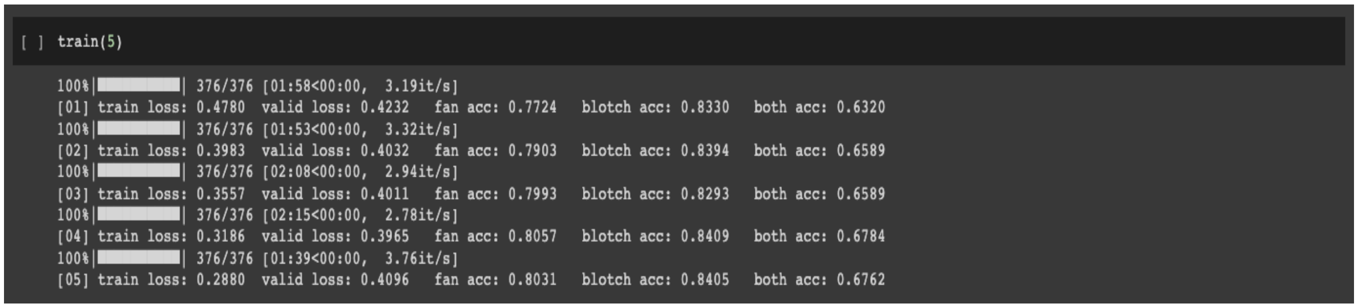
The introduction of new technologies has resulted in profound changes in the field of computer vision in recent years. As a result of these advancements, computer vision models can now outperform humans in solving various image recognition, object detection, face recognition, image classification, and other problems.

```
avg_train_losses = []
avg_valid_losses = []
valid_accuracies = []
traindrop_list = []
validdrop_list = []

def initiate_list(x,y):
   traindrop_list.append(x)
   validdrop_list.append(y)

def train_iteration(optimizer):
   model.train()
   train_drops = []
   for batch, target in tqdm.tqdm(train_loader):

      batch = batch.to(device)
```

```
        target = target.to(device)

        optimizer.zero_grad()

        predictions = model(batch)

        loss = criterion(predictions, target)

        loss.backward()

        optimizer.step()

        train_losses.append(float(loss.item()))

    train_losses = np.array(train_losses)
    return train_losses


def validate():
    model.eval()

    valid_losses = []
    y_true, y_prob = [], []

    with torch.no_grad():
        for batch, target in valid_loader:

            batch = batch.to(device)
            target = target.to(device)

            predictions = model(batch)

            loss = criterion(predictions, target)

            torch.sigmoid_(predictions)

            valid_losses.append(float(loss.item()))
            y_true.extend(target.cpu().numpy())
            y_prob.extend(predictions.cpu().numpy())

    y_true = np.array(y_true)
    y_prob = np.array(y_prob)
    y_pred = y_prob > 0.5
    valid_losses = np.array(valid_losses)

    fan_accuracy = metrics.accuracy_score(y_true[:,0], y_pred[:,0])
    blotch_accuracy = metrics.accuracy_score(y_true[:,1], y_pred[:,1])
    exact_accuracy = np.all(y_true == y_pred, axis=1).mean()

    valid_loss = valid_losses.mean()

    return valid_loss, fan_accuracy, blotch_accuracy, exact_accuracy


def train(epochs, first_epoch=1):
    for epoch in range(first_epoch, epochs+first_epoch):
        train_loss = train_for_epoch(optimizer)



        valid_loss, fan_accuracy, blotch_accuracy, both_accuracy = validate()
        print(f'[{epoch:02d}] train loss: {train_loss.mean():0.04f}  '
            f'valid loss: {valid_loss:0.04f}  ',
            f'fan acc: {fan_accuracy:0.04f}  ',
            f'blotch acc: {blotch_accuracy:0.04f}  ',
            f'both acc: {both_accuracy:0.04f}'
        )

        initiate_list(train_loss.mean(),valid_loss)

        avg_train_losses.append(train_loss.mean())
        avg_valid_losses.append(valid_loss)
        valid_accuracies.append((fan_accuracy, blotch_accuracy, both_accuracy))

    print('traindrop_list = ', traindrop_list)
    print('validdrop_list = ', validdrop_list)

    torch.save(model, f'checkpoints/baseline_{epoch:03d}.pkl')
```

Result:

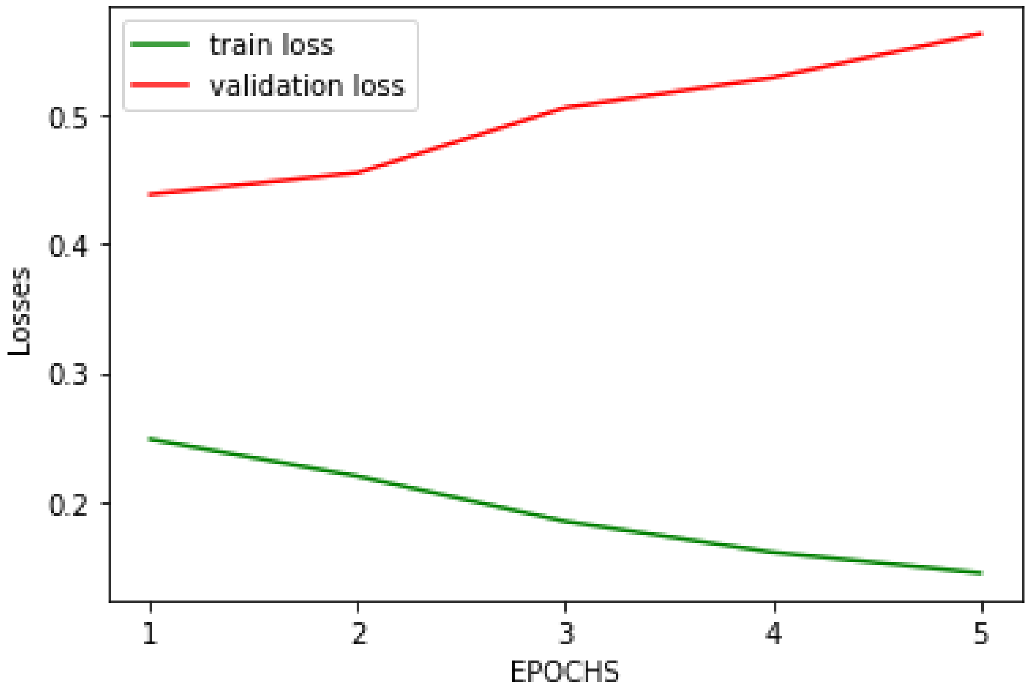
The number of passes ‘n’ = 5 is fed to the train() function and iterated for 5 epochs to obtain the following output,



```
[ ] train(5)

    100%|██████████| 376/376 [01:58<00:00,  3.19it/s]
    [01] train loss: 0.4780  valid loss: 0.4232   fan acc: 0.7724   blotch acc: 0.8330   both acc: 0.6320
    100%|██████████| 376/376 [01:53<00:00,  3.32it/s]
    [02] train loss: 0.3983  valid loss: 0.4032   fan acc: 0.7903   blotch acc: 0.8394   both acc: 0.6589
    100%|██████████| 376/376 [02:08<00:00,  2.94it/s]
    [03] train loss: 0.3557  valid loss: 0.4011   fan acc: 0.7993   blotch acc: 0.8293   both acc: 0.6589
    100%|██████████| 376/376 [02:15<00:00,  2.78it/s]
    [04] train loss: 0.3186  valid loss: 0.3965   fan acc: 0.8057   blotch acc: 0.8409   both acc: 0.6784
    100%|██████████| 376/376 [01:39<00:00,  3.76it/s]
    [05] train loss: 0.2880  valid loss: 0.4096   fan acc: 0.8031   blotch acc: 0.8405   both acc: 0.6762
```

The learning curve is plotted using validation loss

```
epoch_list =[1,2,3,4,5]
plt.plot(epoch_list,traindrop_list)
plt.plot(epoch_list,validdrop_list)
plt.xticks(epoch_list)
plt.xlabel("EPOCHS")
plt.ylabel("Losses")
plt.show()
```

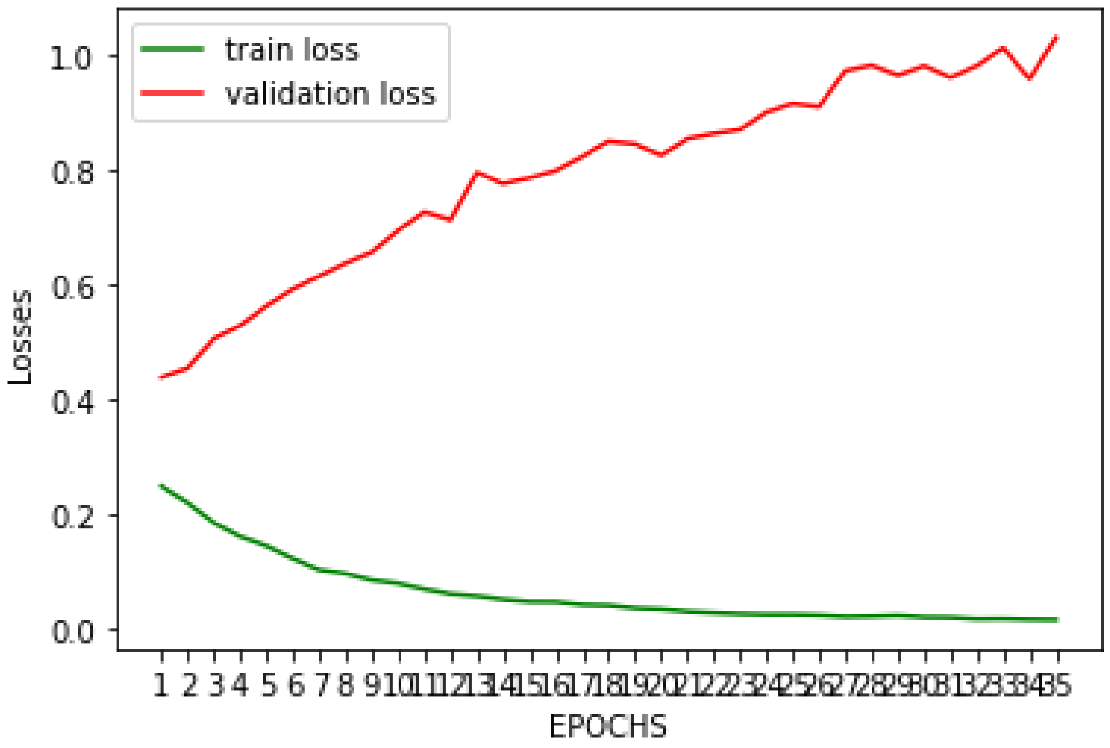


The number of passes ‘n’ = 30 is fed to the train() function and iterated for 5 epochs to obtain the following output,

```
train(30)
```

The learning curve is plotted using validation loss

```
epoch_list =
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,2
8,29,30,31,32,33,34,35]

plt.plot(epoch_list, traindrop_list, color = 'green', label = 'train
loss')

plt.plot(epoch_list, validdrop_list, color = 'red', label = 'validation
loss')

plt.xticks(epoch_list)

plt.xlabel("EPOCHS")

plt.ylabel("Losses")

plt.legend()

plt.show()
```



train loss
validation loss
Losses
EPOCHS
1.0
0.8
0.6
0.4
0.2
0.0