# EE514: DATA ANALYTICS AND MACHINE LEARNING

## ASSIGNMENT

**Spandan Banerjee**

**22267215**

# INTRODUCTION

## Part 1

Part 1 of the assignment consists of data taken from sensors of mobile phones and a personal smart watch of 60 users each identified by a unique user id (UUID). Users were of diverse ethnicity and mostly graduate/under graduate students. The data was captured by the devices at an interval of 1 minute. Most of the data contains labels which were decided by the users. The sensors used were mostly high-frequency motion reactive sensors, location services, audio, watch compasses and phone state indicators. The data had been cleaned and labelled for further use, which we have used in the assignment.

In the notebook, we have initially trained/tested a logistic regression model on one user where we take the accelerometer readings and try to predict if the user is walking or not then further extended it to 5 users to see how accurate our model is. Subsequently the data was split in an 80-20 ratio for making a validation set and a training set. Then the data was tested with changed C-parameter values on a logistic regression model followed by a Support Vector Classifier with changed hyper-parameters which were a linear kernel and a sigmoid kernel. Classification report was generated for each model used. An ROC curve was also plotted for logistic regression model and support vector classifier.

## Part 2

Part 2 of the assignment deals with a deep convolution neural network called ResNet50 in which we train on low-resolution images from the surface of Mars. The two parts which we need to focus on are fans and blotches. Initially we train the model for 5 epochs to generate and report the validation loss and training loss. The hyperparameter called "epochs" determines how many times the learning algorithm will run over the whole training dataset. Every sample in the training dataset has had a chance to update the internal model parameters once during an epoch. Learning curve after training is plotted to gain further insight about the deep CNN.

## PART 1: THE EXTRA SENSORY DATA SET

**Import statements used**:

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

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

*from **sklearn.metrics** import **classification_report** :*

Used to generate a classification report (test metrics) of the models. It gives us the precision, recall, f1-score and support of 0s and 1s (positive and negative predictions). It also helps us to calculate the accuracy, macro and weighted averages.

*from **sklearn.model_selection** import **train_test_split** :*

Helps in splitting the data for testing and validation purposes.

*from **sklearn.svm** import **SVC**:*

Importing the Support Vector Classifier, this is the chosen alternate model for this assignment.

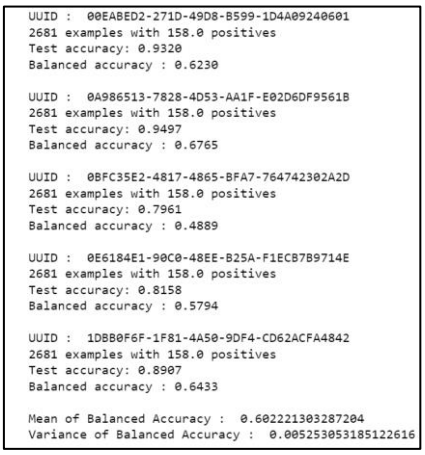
**Improving the test set:**

After training the model on one user, training accuracy and balanced accuracy was obtained then the model was tested on another user. Training accuracy was found to be 0.941 in this case while balanced accuracy was found to be 0.83 which shows the model has fit the training data.

In the below code, the model was tested on each individually. Moreover, we calculated balanced and test accuracy for each UUID. It was observed that balanced accuracy was below 0.5 in some cases which suggests that model does not fit our data well.

```
# Selecting 5 different users and testing the model on each individually

# Evaluating mean and variance of the balanced accuracy


u_str = ['00EABED2-271D-49D8-B599-1D4A09240601', '0A986513-7828-4D53-AA1F-
E02D6DF9561B','0BFC35E2-4817-4865-BFA7-764742302A2D','0E6184E1-90C0-48EE-B25A-
F1ECB7B9714E','1DBB0F6F-1F81-4A50-9DF4-CD62ACFA4842']

x_list = []


def store_list(x) :          # function to store list
    x_list.append(x)


for u in u_str:
    df_test = load_data_for_user(u)
    X_test, y_test = get_features_and_target(df_test, acc_sensors, target_column)
    print("UUID : ", u)
    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 = float(format(metrics.balanced_accuracy_score(y_test, y_pred)))
    store_list(x)
    print(" ")


x_sum = sum(x_list)

x_mean = x_sum/len(x_list)

print("Mean of Balanced Accuracy : ",x_mean)


var = statistics.variance(x_list)

print("Variance of Balanced Accuracy : ",var)
```

Output:




Mean and variance of the balanced accuracy was found to 0.602 and 0.005 respectively.

**Validation data and increasing the training data:**

We combine the 5 users and test the model.

```
#Combining the 5 users and putting in data frame df_combined

u_str = ['00EABED2-271D-49D8-B599-1D4A09240601','0A986513-7828-4D53-AA1F-
E02D6DF9561B','0BFC35E2-4817-4865-BFA7-764742302A2D','0E6184E1-90C0-48EE-B25A-
F1ECB7B9714E','1DBB0F6F-1F81-4A50-9DF4-CD62ACFA4842']

combined_csv = pd.concat([pd.read_csv(data_dir + '/' + (u + '.features_labels.csv')) for u in u_str])

df_csv = pd.DataFrame(combined_csv)

df_csv.to_csv("combine_csv.csv", index=False, encoding='utf-8-sig')
```

Output:

```
2681 examples with 158.0 positives
Test accuracy: 0.8710
Balanced accuracy : 0.5958
```

We can measure a classification model's performance using balanced accuracy.

It is determined by:

(Sensitivity + Specificity) / 2 = balanced accuracy.

where:

Sensitivity: The proportion of positive cases the model is able to identify, or the "real positive rate."

Specificity: The proportion of negative cases that the model is able to identify, or the "real negative rate."

This statistic is especially helpful when there is an imbalance between the two classes, meaning that one class appears substantially more frequently than the other.

In this case, the balanced accuracy is 0.5958 which means the model performs averagely on the data.

To split data into validation set and training set (80-20 split),

```
X_train, X_val, y_train, y_val = train_test_split(X_test, y_test, test_size=0.20, random_state=1)
```

## Model Selection and Model Testing:

Then, it is used to train on a Logistic Regression model once with C-parameter as 1.0 and 0.5 to see the optimisation in variance and a classification report is generated in each case.

```
clf = LogisticRegression(solver='liblinear', max_iter=1000, C=1.0)
clf.fit(X_train, y_train)

print('LR with C = 1.0 Classification Report\n')
print(classification_report(y_val, y_pred))
```

```
LR with C = 1.0 Classification Report

              precision    recall  f1-score   support

         0.0       0.92      0.97      0.94      3949
         1.0       0.71      0.44      0.54       622

    accuracy                           0.90      4571
   macro avg       0.81      0.71      0.74      4571
weighted avg       0.89      0.90      0.89      4571
```

```
clf = LogisticRegression(solver='liblinear', max_iter=1000, C=0.5)
clf.fit(X_train, y_train)
```

```
LR with C = 0.5 Classification Report

              precision    recall  f1-score   support

         0.0       0.92      0.97      0.94      3949
         1.0       0.70      0.47      0.57       622

    accuracy                           0.90      4571
   macro avg       0.81      0.72      0.75      4571
weighted avg       0.89      0.90      0.89      4571
```

Below is a table which gives us the Balanced and Training Accuracy based on two different C-parameters:

| **C - Parameter** | 1.0 | 0.5 |
|---|---|---|
| **Balanced Accuracy** | 0.7061 | 0.7211 |
| **Training Accuracy** | 0.9025 | 0.9025 |

A new model is chosen which is Support Vector Classifier. In order to avoid overfitting, support vector classifiers purposefully misclassify a few training data. As C increases, our tolerance for margin violations grows, and the margin expands, resulting in a more straightforward model with a higher risk of bias and reduced variation. The margin gets smaller as C drops, which causes overfitting that lowers bias and raises variance. In the bias-variance trade-off, C is used as the tuning parameter for model selection.

We again generate the classification report for two Support Vector Classifier with the hyper-parameter kernel as linear and sigmoid.

```
clf = SVC(C=0.1, kernel='linear', gamma= 1)
clf.fit(X_train, y_train)
X_val = scaler.transform(X_val)
X_val = imputer.transform(X_val)
y_pred = clf.predict(X_val)
print('SVM with linear kernel Classification Report\n')
print(classification_report(y_val, y_pred))
```

```
SVM with linear kernel Classification Report

              precision    recall  f1-score   support

         0.0       0.91      0.98      0.94      3949
         1.0       0.74      0.36      0.48       622

    accuracy                           0.90      4571
   macro avg       0.82      0.67      0.71      4571
weighted avg       0.88      0.90      0.88      4571
```

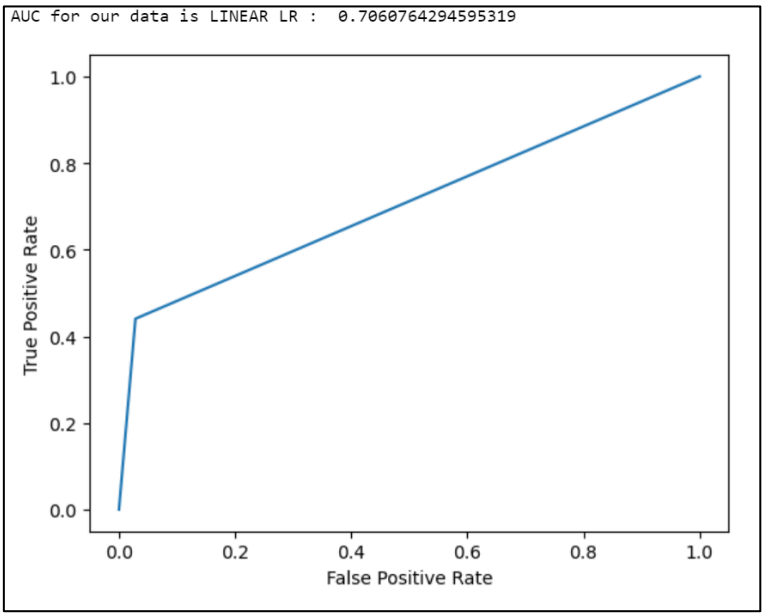
```
clf = SVC(C= .1, kernel='sigmoid', gamma= 1)
clf.fit(X_train, y_train) X_val = scaler.transform(X_val)
X_val = imputer.transform(X_val)
y_pred = clf.predict(X_val)
print('SVM with sigmoid kernel Classification Report\n')
print(classification_report(y_val, y_pred))
```

```
SVM with sigmoid kernel Classification Report

              precision    recall  f1-score   support

         0.0       0.88      0.93      0.90      3949
         1.0       0.31      0.22      0.26       622

    accuracy                           0.83      4571
   macro avg       0.60      0.57      0.58      4571
weighted avg       0.80      0.83      0.82      4571
```

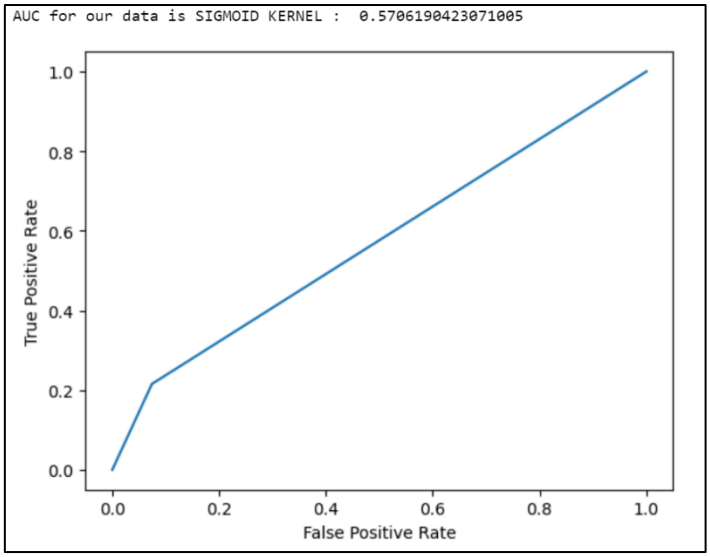
**Plotting the ROC Curve:**

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

```
#ROC Curve LR
fpr, tpr, _ = metrics.roc_curve(y_val,  y_pred)
auc = metrics.roc_auc_score(y_val, y_pred)
print("AUC for our data is LINEAR LR : ",auc)
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




```
#ROC Curve SIGMOID KERNEL
fpr, tpr, _ = metrics.roc_curve(y_val,  y_pred)
auc = metrics.roc_auc_score(y_val, y_pred)
print("AUC for our data is SIGMOID KERNEL : ",auc)
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




AUC-ROC curve gives us the performance measurement for the classification problems at various threshold settings. ROC is the probability curve and AUC represents the degree or measure of separability. Higher AUC means, the model is better at predicting 0 classes as 0 and 1 classes as 1. Our logistic regression model gives us an AUC = 0.706 which means it is around 70% accurate whereas for SVC sigmoid kernel we get an AUC = 0.5706 which is around 50% accurate, just average.

# PART 2: PLANET FOUR

Residual Networks or ResNet is a convolutional neural network which is more than 50-layers deep. It is a common neural network that serves as the foundation for many computer vision applications. The main innovation with ResNet was that it allows to train very deep neural networks with more than 150 layers.

After loading the data and setting up the environment, certain modifications were made to store the validation loss and training loss into a list so that a learning curve can be plotted.

```
avg_train_losses = []
avg_valid_losses = []
valid_accuracies = []
trainloss_list = []
validloss_list = []


def make_list(x,y):
    trainloss_list.append(x)
    validloss_list.append(y)


def train_for_epoch(optimizer):
    model.train()
    train_losses = []
    for batch, target in tqdm.tqdm(train_loader):
        # data to GPU
        batch = batch.to(device)
        target = target.to(device)

        # reset optimizer
        optimizer.zero_grad()

        # forward pass
        predictions = model(batch)
        #breakpoint()

        # calculate loss
        loss = criterion(predictions, target)

        # backward pass
        loss.backward()

        # parameter update
        optimizer.step()
```

```
        # track loss
        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:

            # move data to the device
            batch = batch.to(device)
            target = target.to(device)

            # make predictions
            predictions = model(batch)

            # calculate loss
            loss = criterion(predictions, target)

            # logits -> probabilities
            torch.sigmoid_(predictions)

            # track losses and 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)
```

```
    # calculate validation accuracy from y_true and y_pred
    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()

    # calculate the mean validation loss
    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
        train_loss = train_for_epoch(optimizer)

        # validation
        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}'
        )

        make_list(train_loss.mean(),valid_loss)
       # update learning curves
        avg_train_losses.append(train_loss.mean())
        avg_valid_losses.append(valid_loss)
        valid_accuracies.append((fan_accuracy, blotch_accuracy, both_accuracy))

print('trainloss_list = ', trainloss_list)
print('validloss_list = ', validloss_list)

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

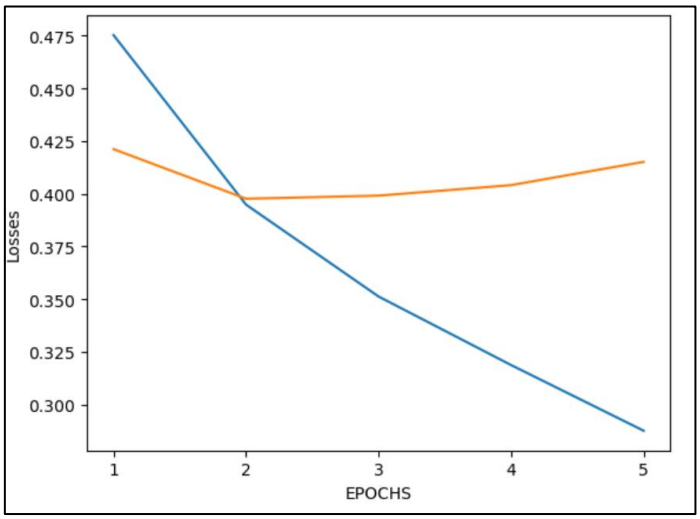
We pass n = 5 in the train() function to iterate it for 5 epochs and we get the following output,

```
train(5)
```

```
100%|██████████| 376/376 [27:41<00:00,  4.42 s/it]
[01] train loss: 0.4753  valid loss: 0.4212   fan acc: 0.7791   blotch acc: 0.8383   both acc: 0.6402
100%|██████████| 376/376 [27:21<00:00,  4.37 s/it]
[02] train loss: 0.3949  valid loss: 0.3977   fan acc: 0.8004   blotch acc: 0.8420   both acc: 0.6675
100%|██████████| 376/376 [28:00<00:00,  4.47 s/it]
[03] train loss: 0.3513  valid loss: 0.3991   fan acc: 0.7915   blotch acc: 0.8431   both acc: 0.6657
100%|██████████| 376/376 [27:10<00:00,  4.34 s/it]
[04] train loss: 0.3187  valid loss: 0.4041   fan acc: 0.7978   blotch acc: 0.8428   both acc: 0.6705
100%|██████████| 376/376 [27:01<00:00,  4.31 s/it]
[05] train loss: 0.2876  valid loss: 0.4152   fan acc: 0.7963   blotch acc: 0.8450   both acc: 0.6702
```

For the learning curve we plot it using validation loss, training loss VS number of epochs,

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

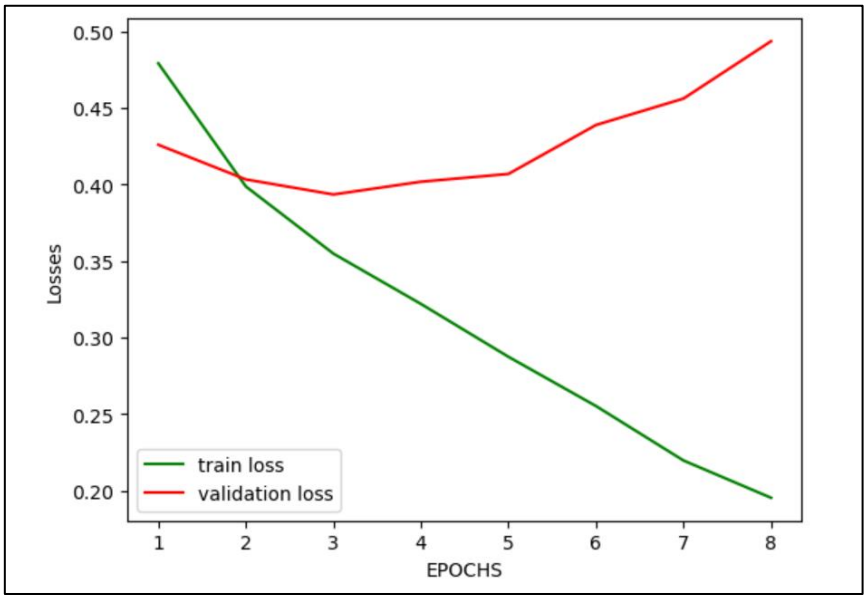


We pass n = 8 in for the second run and observe the following output,

```
train(8)
100%|██████████| 376/376 [26:57<00:00,  4.30 s/it]
[01] train loss: 0.4792  valid loss: 0.4260   fan acc: 0.7802   blotch acc: 0.8319   both acc: 0.6380
100%|██████████| 376/376 [25:04<00:00,  4.00 s/it]
[02] train loss: 0.3988  valid loss: 0.4034   fan acc: 0.7866   blotch acc: 0.8413   both acc: 0.6586
100%|██████████| 376/376 [25:09<00:00,  4.01 s/it]
[03] train loss: 0.3549  valid loss: 0.3935   fan acc: 0.8064   blotch acc: 0.8420   both acc: 0.6754
100%|██████████| 376/376 [25:05<00:00,  4.00 s/it]
[04] train loss: 0.3221  valid loss: 0.4019   fan acc: 0.8094   blotch acc: 0.8267   both acc: 0.6683
100%|██████████| 376/376 [24:40<00:00,  3.94 s/it]
[05] train loss: 0.2875  valid loss: 0.4070   fan acc: 0.8121   blotch acc: 0.8409   both acc: 0.6818
100%|██████████| 376/376 [24:53<00:00,  3.97 s/it]
[06] train loss: 0.2553  valid loss: 0.4389   fan acc: 0.7967   blotch acc: 0.8356   both acc: 0.6627
```

```
100%|██████████| 376/376 [24:53<00:00,  3.97 s/it]
[07] train loss: 0.2197  valid loss: 0.4562   fan acc: 0.8027   blotch acc: 0.8248   both acc: 0.6593
100%|██████████| 376/376 [24:36<00:00,  3.93 s/it]
[08] train loss: 0.1954  valid loss: 0.4937   fan acc: 0.7956   blotch acc: 0.8274   both acc: 0.6601
```

```
epoch_list =[1,2,3,4,5,6,7,8]
plt.plot(epoch_list,trainloss_list, color = 'green', label = "train loss")
plt.plot(epoch_list,validloss_list, color = 'red', label = "validation loss")
plt.xticks(epoch_list)
plt.xlabel("EPOCHS")
plt.ylabel("Losses")
plt.legend()
plt.show()
```



Since the validation loss is going upwards and training loss is going downwards and the gap between them is increasing, we can conclude that our model is underfitting. To make the model better we need to reduce the gap between validation and training loses.

We train using ResNet50 with weights as imagenet1k_v2 to train for 3 epochs.

```
model = models.resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)
model.fc = nn.Linear(2048, 2)
model.to(device)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.05)
```

```
train(3)
100%|██████████| 376/376 [45:59<00:00,  7.34 s/it]
[01] train loss: 0.7373  valid loss: 0.8152   fan acc: 0.6219   blotch acc: 0.8083   both acc: 0.4399
100%|██████████| 376/376 [43:39<00:00,  6.97 s/it]
[02] train loss: 0.5985  valid loss: 0.6108   fan acc: 0.6207   blotch acc: 0.8083   both acc: 0.4395
100%|██████████| 376/376 [47:40<00:00,  7.61 s/it]
[03] train loss: 0.5861  valid loss: 0.5800   fan acc: 0.6249   blotch acc: 0.8083   both acc: 0.4433
```

For getting the learning curve, we do the following

```
epoch_list =[1,2,3]
plt.plot(epoch_list,trainloss_list, color = 'green', label = "train loss")
plt.plot(epoch_list,validloss_list, color = 'red', label = "validation loss")
plt.xticks(epoch_list)
plt.xlabel("EPOCHS")
plt.ylabel("Losses")
plt.legend()
plt.show()
```