Speech Emotion Recognition

Name 1: Abdelrahman Salem Mohamed

ID 1: 6309

Name 2: Reem Abdelhalim

ID 2: 6114

Name 3: Salma Ahmed

ID 3: 6126

Problem Statement

Speech is the most natural way of expressing ourselves as humans. It is only natural then to extend this communication medium to computer applications. We define speech emotion recognition (SER) systems as a collection of methodologies that process and classify speech signals to detect the embedded emotions. Below we will show the needed steps to achieve the goal of the assignment

Procedures

1- Download and explore the CREMA dataset

Which contain 6 different classes {sad, happy, angry, fear, disguised, neutral}

```
def read_data(root):
  data = []

  directory = os.fsencode(root)
  for file in os.listdir(directory):
      filename = os.fsdecode(file)
      filename= root + filename
      data.append([filename,filename[24:27]])
  return pd.DataFrame(data, columns = ['File_path', 'Class'])
```

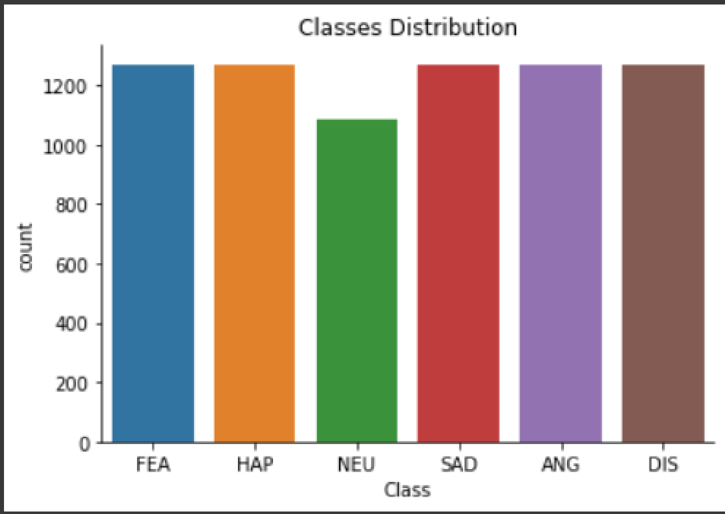
```
data_df = read_data("/content/Crema/")
```

```
data_df.head()
```

	File_path	Class
0	/content/Crema/1082_IWL_FEA_XX.wav	FEA
1	/content/Crema/1062_TSI_HAP_XX.wav	HAP
2	/content/Crema/1013_IEO_FEA_MD.wav	FEA
3	/content/Crema/1034_DFA_NEU_XX.wav	NEU
4	/content/Crema/1012_ITH_NEU_XX.wav	NEU

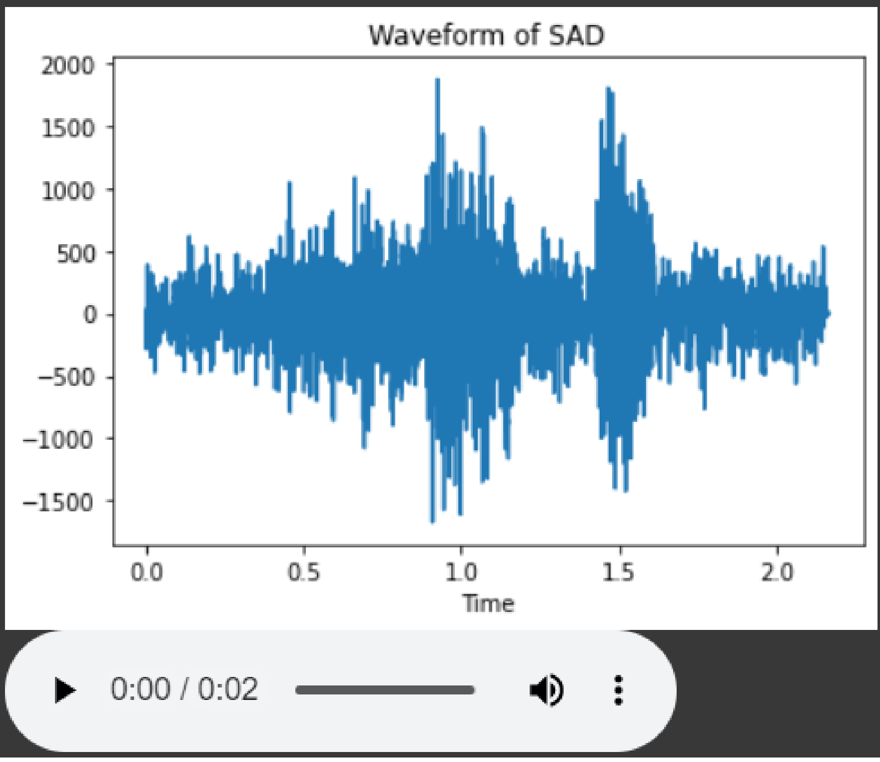
And here the classes distribution over the dataset

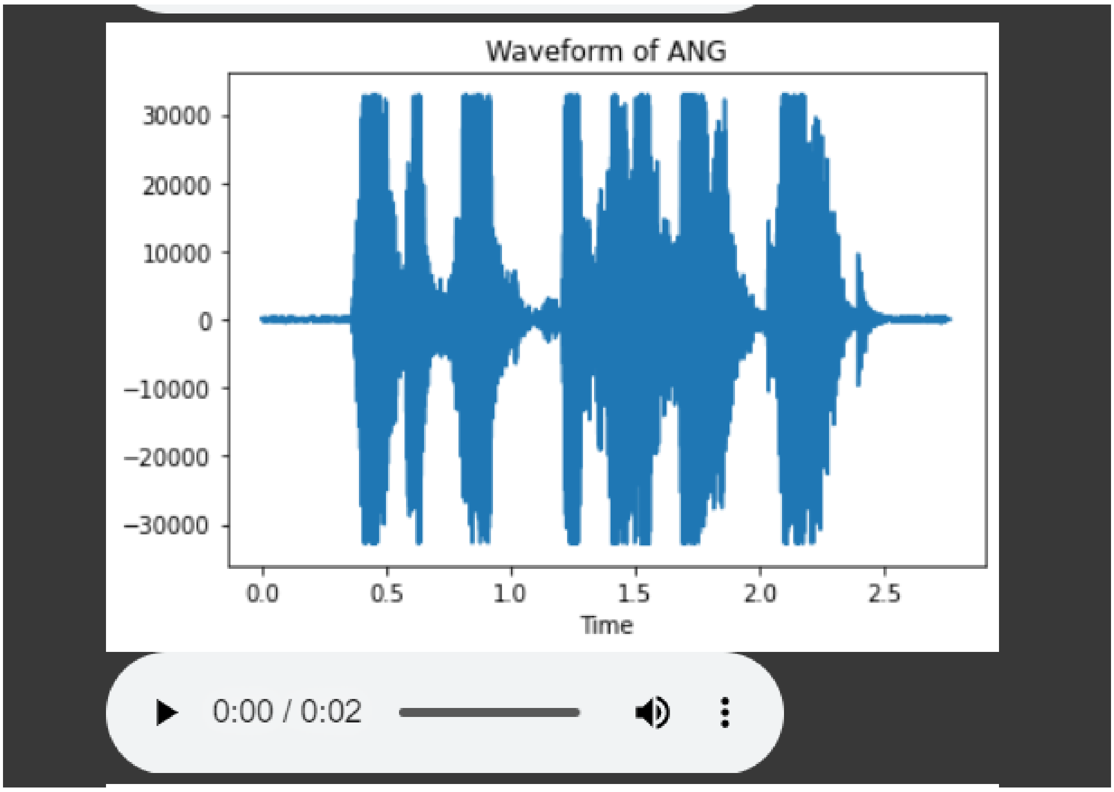
```
plt.title("Classes Distribution")
sns.countplot(x = data_df["Class"])
sns.despine(top = True, right = True, left = False, bottom = False)
```



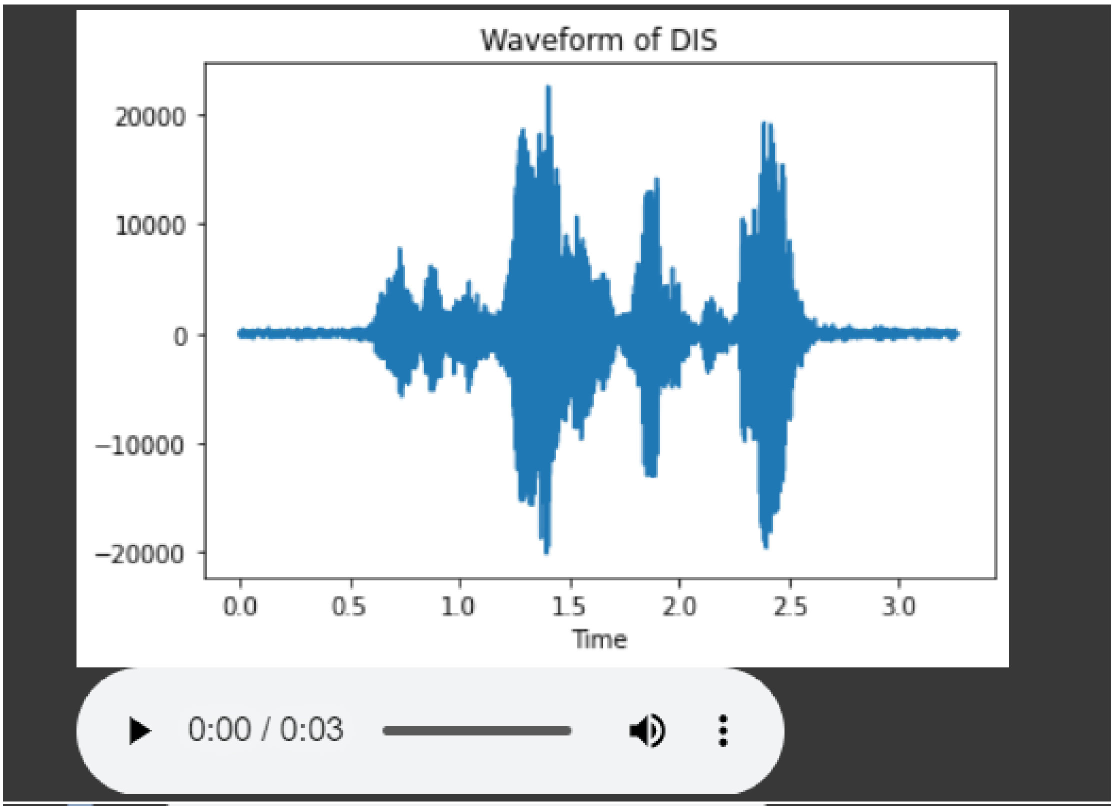
As we see the distribution has little unbalanced due to the neutral class but we believed it is accepted due to the dataset limitation

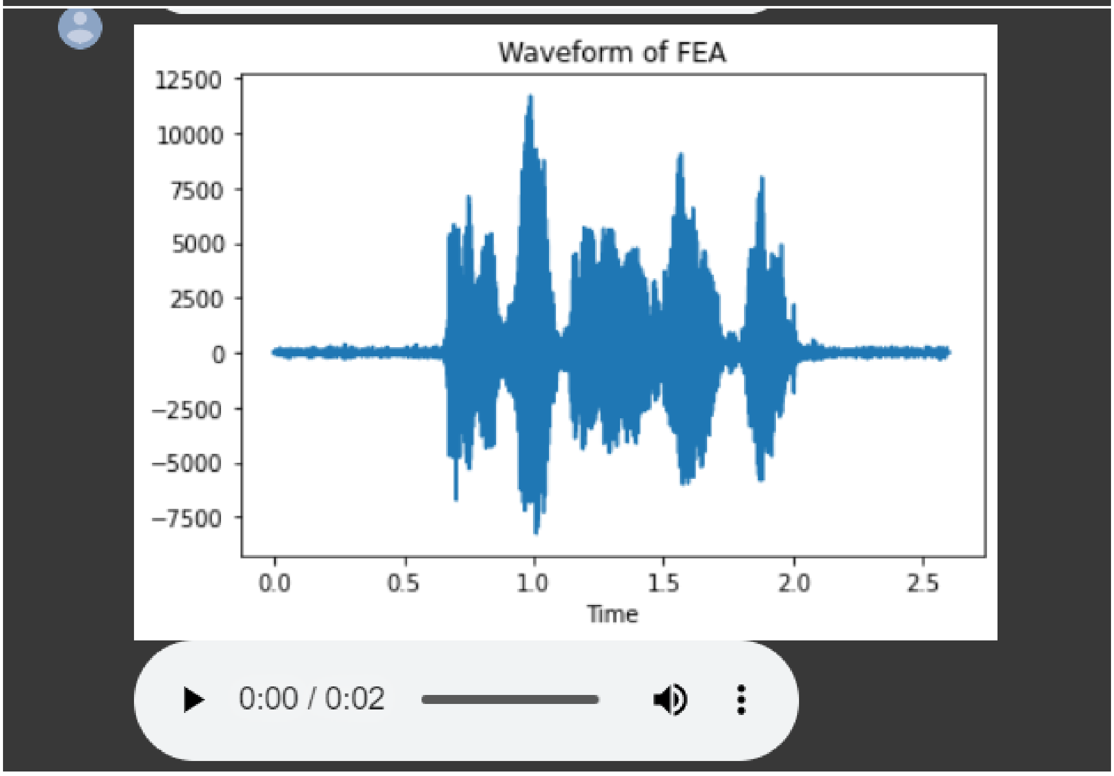
2- Visualize and plot the time domain of the audios

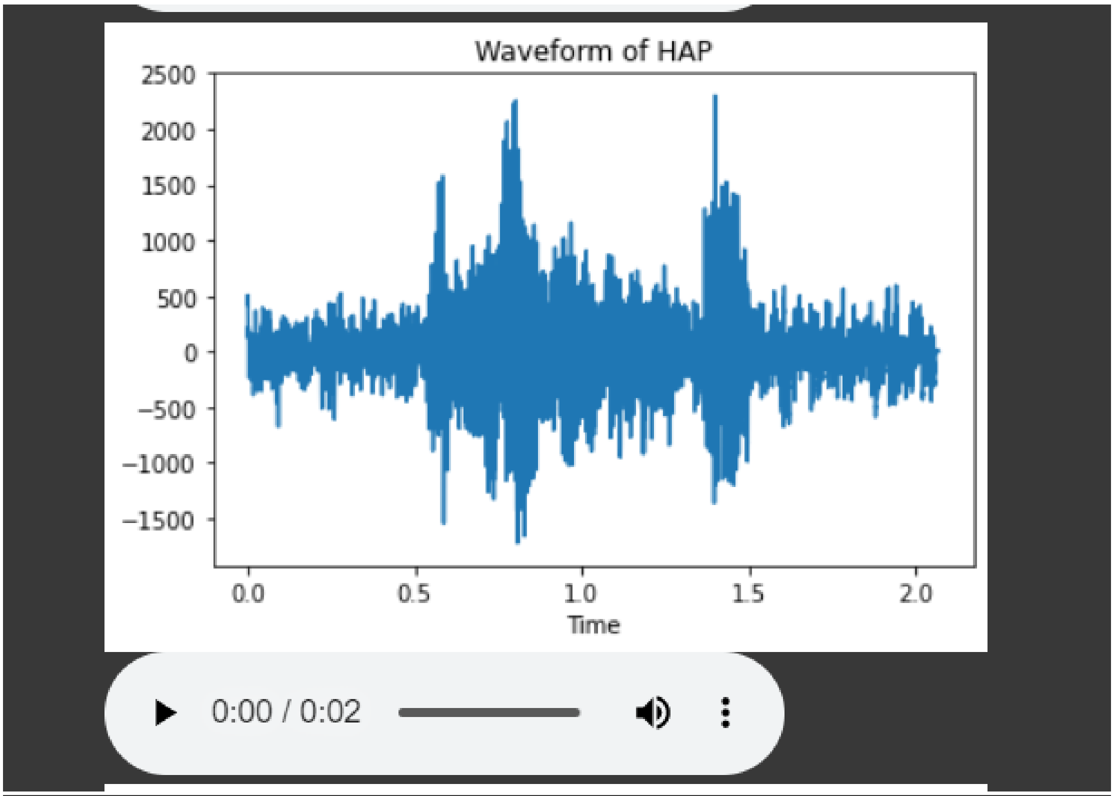


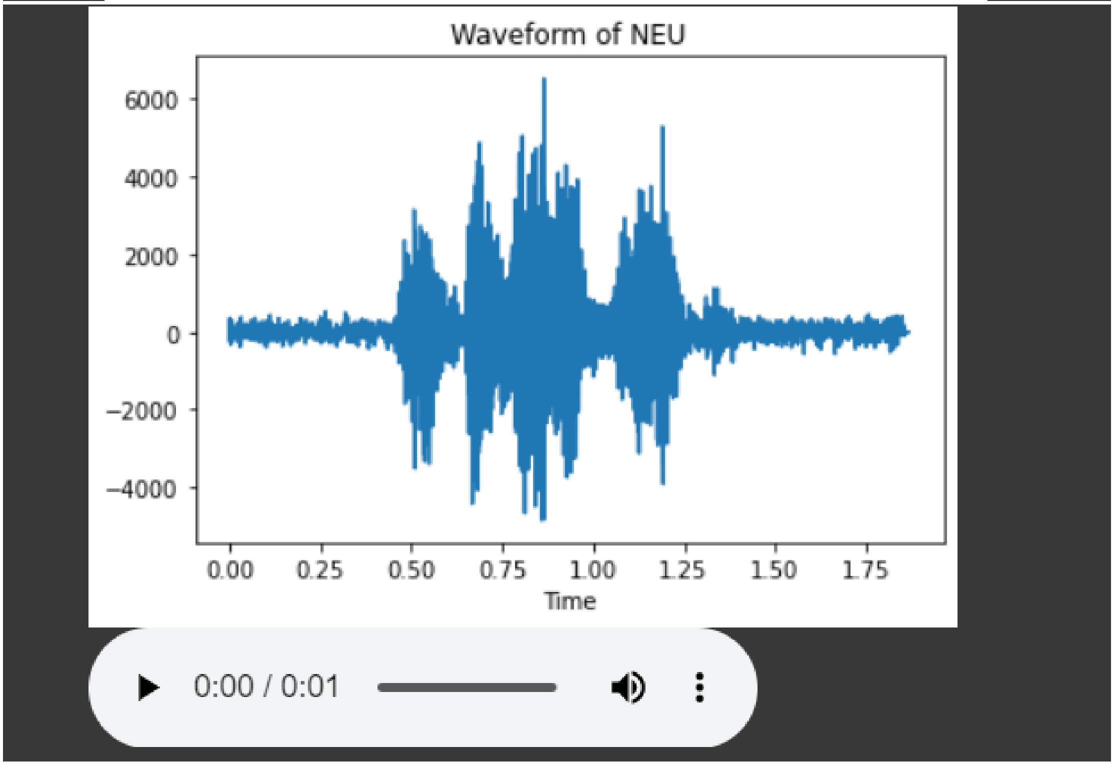


Waveform of ANG
30000
20000
10000
0
−10000
−20000
−30000
0.0
0.5
1.0
1.5
2.0
2.5
Time
0:00 / 0:02


Waveform of DIS
20000
10000
0
−10000
−20000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Time
0:00 / 0:03


Waveform of FEA
12500
10000
7500
5000
2500
0
−2500
−5000
−7500
0.0
0.5
1.0
1.5
2.0
2.5
Time
0:00 / 0:02


Waveform of HAP
2500
2000
1500
1000
500
0
−500
−1000
−1500
0.0
0.5
1.0
1.5
2.0
Time
0:00 / 0:02


Waveform of NEU
6000
4000
2000
0
−2000
−4000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Time
0:00 / 0:01

3- Dataset Preprocessing

During out dataset exploring we noticed that the audios are not equal in length which present unequal samples, we decide to perform audio equaling where finding the smallest audio and dividing all the audio according to it and discard any remaining samples

```
def audio_equalling(data_df):
  samples = []
  labels = []
  min_len = np.Inf               # minimum length =27959 when sr=22050
  equalled_samples = []

  # to get the length of smallest audio file
  for i in range(len(data_df)):
    samples.append(librosa.load(data_df['File_path'][i],sr=SAMPLING_RATE)[0])
    min_len = min(min_len,len(samples[i]))

  # add zero padding vector equal to difference between maximum audio and audio length
  # samples = np.array(samples)
  # for i in range(len(samples)):
  #   samples[i] = np.concatenate((samples[i],np.zeros(max_len - len(samples[i]))),axis=0)

  # divde the samples according to the smallest audio length and clip the incompleted samples
  # to do zero padding for the neglicting part
  for i in range(len(samples)):
    it = len(samples[i]) // min_len
    label = extract_label(data_df["Class"][i])
    for j in range(it):
      equalled_samples.append(samples[i][j*min_len+1:(j+1)*min_len])
      labels.append(label)

  return equalled_samples,np.array(labels)
```

4- Feature Spaces

a- ZCR and Energy

concatenating the audio signal samples with it's ZCR and energy constructing first feature space

First Feature Space: ZCR and Energy --> (we can add more features)

```
def custom_features(x):
  zcrs = sum(np.diff(np.sign(x)) != 0)
  absolute_x = abs(x)
  energy = np.mean(absolute_x*absolute_x)     # this takes less computation time
  return np.array([zcrs,energy])
```

b- Mel Spectrogram

feature space based on converting the audio signals to 2D images that represents the audio in another space

Second Feature Space: Mel Spectorgram

```
[ ]  def mel_spectrogram(x):
       S = librosa.feature.melspectrogram(y=x)
       return S
```

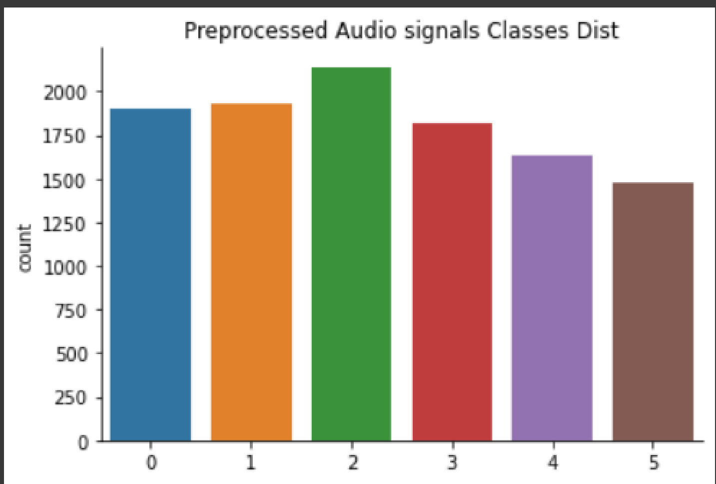
5- Splitting The dataset

Splits the audio signals dataset into 70% for training and validation and 30% for testing and make it balanced as possible

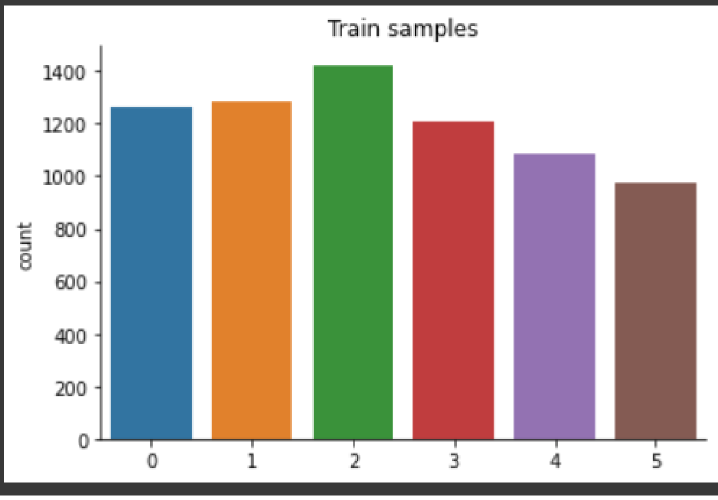
```
# TODO dataset after pre-processing is unbalanced
train_features, test_features, train_labels, test_labels = train_test_split(d1_reshaped,labels, test_size=0.3,
                                                                              random_state=42,stratify=labels)
train_features, val_features, train_labels, val_labels = train_test_split(train_features, train_labels, test_size=0.05,
                                                                            random_state=42,stratify=train_labels)
```

The distribution of classes accros each data split is almost the same

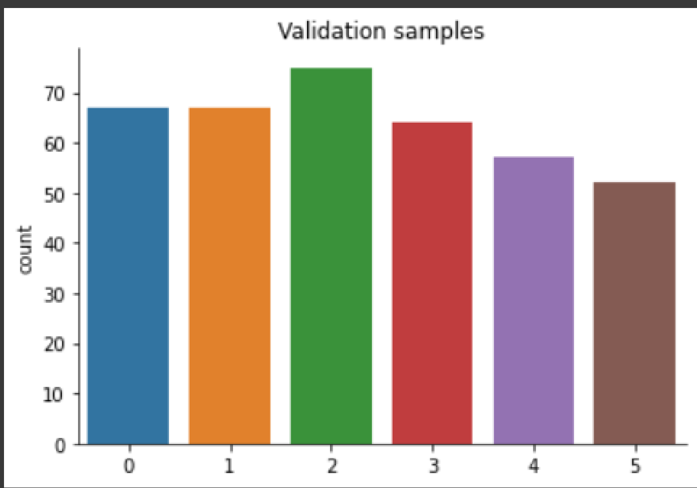
```
plt.title("Preprocessed Audio signals Classes Dist")
sns.countplot(x = labels)
sns.despine(top = True, right = True, left = False, bottom = False)
```



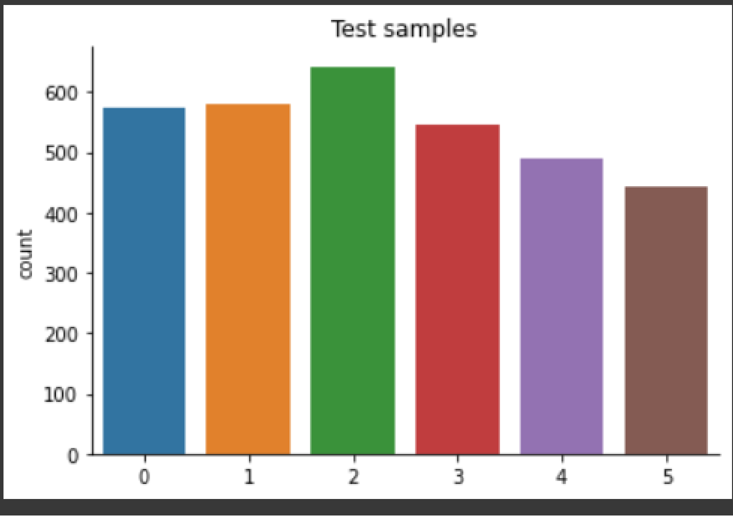
```
[ ] plt.title("Train samples")
    sns.countplot(x = train_labels)
    sns.despine(top = True, right = True, left = False, bottom = False)
```




```
plt.title("Validation samples")
sns.countplot(x = val_labels)
sns.despine(top = True, right = True, left = False, bottom = False)
```

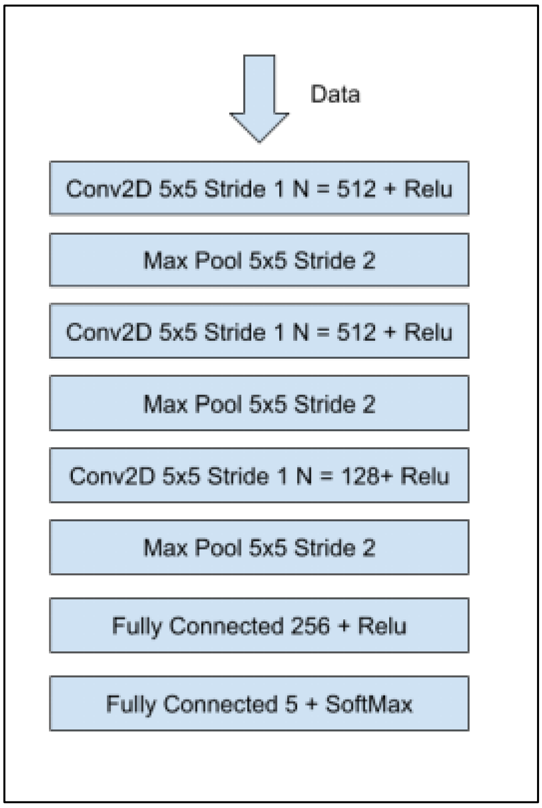


```python
plt.title("Test samples")
sns.countplot(x = test_labels)
sns.despine(top = True, right = True, left = False, bottom = False)
```



6- CNN model

We used keras first for just testing the dataset before using Pytorch we actually followed the assignment requirements architecture with little additions



a- For Spectrogram Feature space

we have added dropout just before the last dense layer

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=512, kernel_size=(5,3),stride=1)
        self.pool1 = nn.MaxPool2d(kernel_size=(5,3), stride=2)

        self.conv2 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=(5,3),stride=1)
        self.pool2 = nn.MaxPool2d(kernel_size=(5,3), stride=2)

        self.conv3 = nn.Conv2d(in_channels=512, out_channels=128, kernel_size=(5,3),stride=1)
        self.pool3 = nn.MaxPool2d(kernel_size=(5,3), stride=2)

        self.fc1 = nn.Linear(in_features=4608,out_features=256)
        self.dropout1 = nn.Dropout(0.2)
        self.fc2 = nn.Linear(in_features=256,out_features=6)
```

```
    # Defining the forward pass
    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = self.pool3(F.relu(self.conv3(x)))
        x = torch.flatten(x,1)
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = F.log_softmax(self.fc2(x),dim=0)
        return x
```

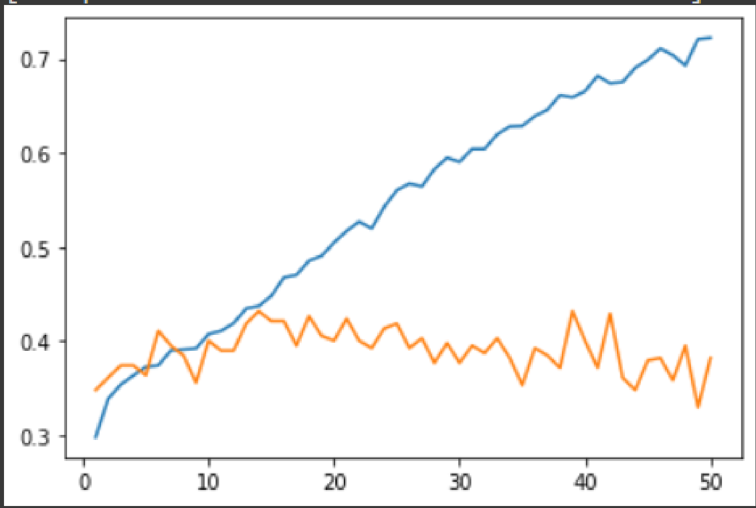
Train Loop

We configured the batch size with 8, learning rate with 0.0001 and 50 epochs but due to the model overfitted so we adjusted the number of epochs

```
Epochs: 40 | Train Loss:  0.106 | Train Accuracy:  0.666 | Val Loss:  0.425 | Val Accuracy:  0.401
100%|██████████| 907/907 [01:22<00:00, 11.02it/s]
Epochs: 41 | Train Loss:  0.101 | Train Accuracy:  0.682 | Val Loss:  0.499 | Val Accuracy:  0.372
100%|██████████| 907/907 [01:22<00:00, 11.02it/s]
Epochs: 42 | Train Loss:  0.108 | Train Accuracy:  0.674 | Val Loss:  0.415 | Val Accuracy:  0.429
100%|██████████| 907/907 [01:22<00:00, 10.98it/s]
Epochs: 43 | Train Loss:  0.106 | Train Accuracy:  0.675 | Val Loss:  0.418 | Val Accuracy:  0.361
100%|██████████| 907/907 [01:22<00:00, 10.98it/s]
Epochs: 44 | Train Loss:  0.100 | Train Accuracy:  0.690 | Val Loss:  0.474 | Val Accuracy:  0.348
100%|██████████| 907/907 [01:22<00:00, 10.99it/s]
Epochs: 45 | Train Loss:  0.097 | Train Accuracy:  0.699 | Val Loss:  0.449 | Val Accuracy:  0.380
100%|██████████| 907/907 [01:22<00:00, 10.99it/s]
Epochs: 46 | Train Loss:  0.092 | Train Accuracy:  0.711 | Val Loss:  0.512 | Val Accuracy:  0.382
100%|██████████| 907/907 [01:22<00:00, 10.98it/s]
Epochs: 47 | Train Loss:  0.096 | Train Accuracy:  0.704 | Val Loss:  0.570 | Val Accuracy:  0.359
100%|██████████| 907/907 [01:22<00:00, 10.98it/s]
Epochs: 48 | Train Loss:  0.102 | Train Accuracy:  0.693 | Val Loss:  0.544 | Val Accuracy:  0.395
100%|██████████| 907/907 [01:22<00:00, 10.98it/s]
Epochs: 49 | Train Loss:  0.089 | Train Accuracy:  0.721 | Val Loss:  0.606 | Val Accuracy:  0.330
100%|██████████| 907/907 [01:22<00:00, 11.02it/s]
Epochs: 50 | Train Loss:  0.093 | Train Accuracy:  0.722 | Val Loss:  0.542 | Val Accuracy:  0.382
```

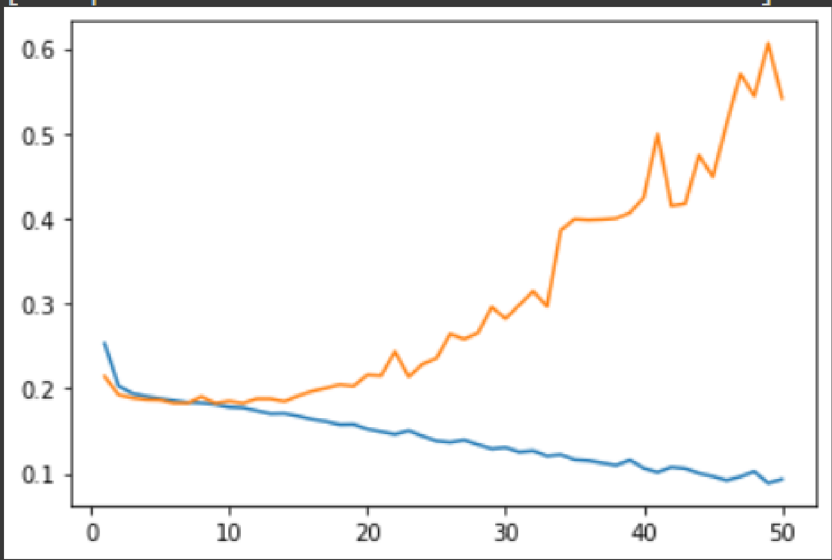
```
[ ] x_axis = np.linspace(1, EPOCHS, EPOCHS, endpoint=True)
    plt.plot(x_axis, history['train_acc'])
    plt.plot(x_axis, history['val_acc'])
```

```
[<matplotlib.lines.Line2D at 0x7f0e50c22c50>]
```




```
plt.plot(x_axis, history['train_loss'])
plt.plot(x_axis, history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f0e506fc950>]
```

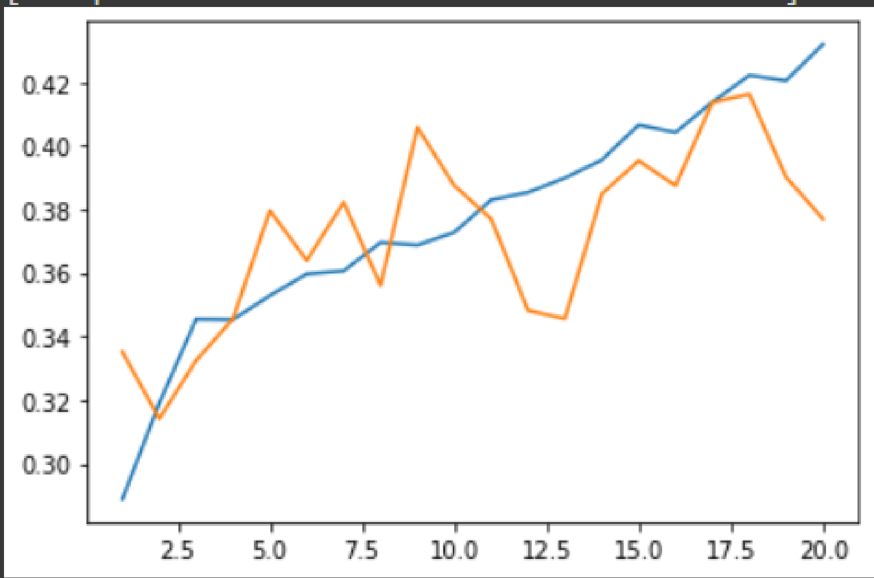


When we setup the epochs to 20 the validation accuracy and train accuracy nearly the same and the model not overfitted

```
Epochs: 13 | Train Loss:  0.181 | Train Accuracy:  0.390 | Val Loss:  0.185 | Val Accuracy:  0.346
100%|██████████| 907/907 [01:22<00:00, 11.01it/s]
Epochs: 14 | Train Loss:  0.180 | Train Accuracy:  0.396 | Val Loss:  0.191 | Val Accuracy:  0.385
100%|██████████| 907/907 [01:22<00:00, 11.01it/s]
Epochs: 15 | Train Loss:  0.181 | Train Accuracy:  0.407 | Val Loss:  0.184 | Val Accuracy:  0.395
100%|██████████| 907/907 [01:22<00:00, 11.01it/s]
Epochs: 16 | Train Loss:  0.179 | Train Accuracy:  0.404 | Val Loss:  0.184 | Val Accuracy:  0.387
100%|██████████| 907/907 [01:22<00:00, 11.02it/s]
Epochs: 17 | Train Loss:  0.176 | Train Accuracy:  0.414 | Val Loss:  0.185 | Val Accuracy:  0.414
100%|██████████| 907/907 [01:22<00:00, 11.02it/s]
Epochs: 18 | Train Loss:  0.175 | Train Accuracy:  0.422 | Val Loss:  0.193 | Val Accuracy:  0.416
100%|██████████| 907/907 [01:22<00:00, 11.02it/s]
Epochs: 19 | Train Loss:  0.173 | Train Accuracy:  0.420 | Val Loss:  0.193 | Val Accuracy:  0.390
100%|██████████| 907/907 [01:22<00:00, 11.00it/s]
Epochs: 20 | Train Loss:  0.171 | Train Accuracy:  0.432 | Val Loss:  0.191 | Val Accuracy:  0.377
```

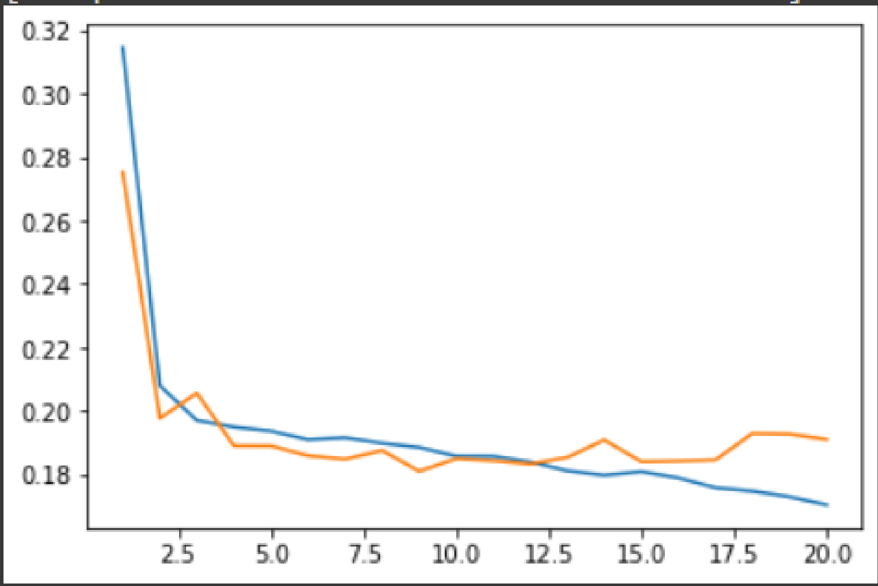
```python
x_axis = np.linspace(1, EPOCHS, EPOCHS, endpoint=True)
plt.plot(x_axis, history['train_acc'])
plt.plot(x_axis, history['val_acc'])
```

```
[<matplotlib.lines.Line2D at 0x7f0e50627950>]
```



```
[ ] plt.plot(x_axis, history['train_loss'])
    plt.plot(x_axis, history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f0e50592c90>]
```

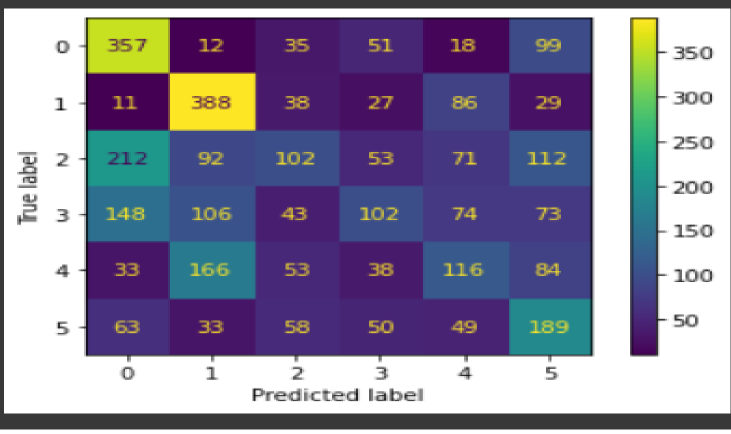


Testing and Results

```
[ ] y_pred,y_true,total_acc = test_model(net,test_dataloader)
    f1_score(y_true, y_pred, average=None)
```

```
Test Accuracy:  0.383
array([0.51146132, 0.56395349, 0.21009269, 0.23529412, 0.25663717,
       0.36770428])
```

```
[ ] confusion_matrix(y_true, y_pred)
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
    plt.show()
```



b- ZCR and Energy Feature Space

We added one more dense layer at the end with dropout

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=512, kernel_size=5,stride=1)
        self.pool1 = nn.MaxPool1d(kernel_size=5, stride=2)

        self.conv2 = nn.Conv1d(in_channels=512, out_channels=512, kernel_size=5,stride=1)
        self.pool2 = nn.MaxPool1d(kernel_size=5, stride=2)

        self.conv3 = nn.Conv1d(in_channels=512, out_channels=128, kernel_size=5,stride=1)
        self.pool3 = nn.MaxPool1d(kernel_size=5, stride=2)

        # self.dropout = nn.Dropout(0.25)
        self.fc1 = nn.Linear(in_features=446464,out_features=256)
        self.dropout = nn.Dropout(0.2)
        self.fc2 = nn.Linear(in_features=256, out_features=128)
        self.fc3 = nn.Linear(in_features=128,out_features=6)
```

```
# Defining the forward pass
def forward(self, x):
    # print(x.shape)
    x = self.pool1(F.relu(self.conv1(x)))
    # print(x.shape)
    x = self.pool2(F.relu(self.conv2(x)))
    # print(x.shape)
    x = self.pool3(F.relu(self.conv3(x)))
    # print(x.shape)
    x = torch.flatten(x,1)
    # print(x.shape)
    x = F.relu(self.fc1(x))
    x = self.dropout(x)
    # print(x.shape)
    x = F.relu(self.fc2(x))
    x = F.log_softmax(self.fc3(x), dim=-1)
    # print(x.shape)
    return x
```

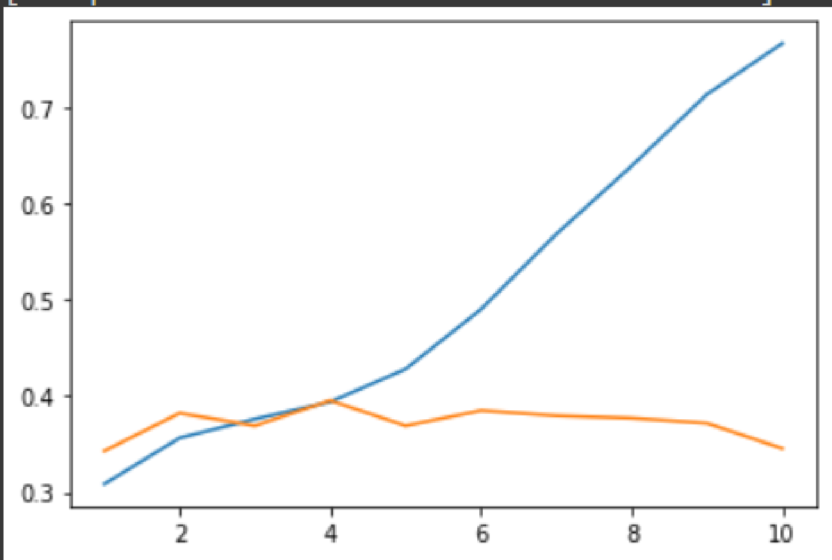
Train loop

At first, we use 10 epochs and learning rate =0.0001 and batch size=8 but the model overfitted

```
[ ]  model, summary = train_model(net, train_dataloader, val_dataloader)

100%|██████████| 907/907 [05:29<00:00,  2.75it/s]
Epochs: 1 | Train Loss:  0.203 | Train Accuracy:  0.309 | Val Loss:  0.190 | Val Accuracy:  0.343
100%|██████████| 907/907 [05:28<00:00,  2.76it/s]
Epochs: 2 | Train Loss:  0.192 | Train Accuracy:  0.356 | Val Loss:  0.184 | Val Accuracy:  0.382
100%|██████████| 907/907 [05:28<00:00,  2.76it/s]
Epochs: 3 | Train Loss:  0.187 | Train Accuracy:  0.376 | Val Loss:  0.186 | Val Accuracy:  0.369
100%|██████████| 907/907 [05:28<00:00,  2.76it/s]
Epochs: 4 | Train Loss:  0.182 | Train Accuracy:  0.394 | Val Loss:  0.185 | Val Accuracy:  0.395
100%|██████████| 907/907 [05:28<00:00,  2.76it/s]
Epochs: 5 | Train Loss:  0.174 | Train Accuracy:  0.428 | Val Loss:  0.185 | Val Accuracy:  0.369
100%|██████████| 907/907 [05:28<00:00,  2.76it/s]
Epochs: 6 | Train Loss:  0.158 | Train Accuracy:  0.490 | Val Loss:  0.196 | Val Accuracy:  0.385
100%|██████████| 907/907 [05:28<00:00,  2.76it/s]
Epochs: 7 | Train Loss:  0.136 | Train Accuracy:  0.568 | Val Loss:  0.229 | Val Accuracy:  0.380
100%|██████████| 907/907 [05:29<00:00,  2.75it/s]
Epochs: 8 | Train Loss:  0.116 | Train Accuracy:  0.639 | Val Loss:  0.252 | Val Accuracy:  0.377
100%|██████████| 907/907 [05:29<00:00,  2.75it/s]
Epochs: 9 | Train Loss:  0.094 | Train Accuracy:  0.713 | Val Loss:  0.336 | Val Accuracy:  0.372
100%|██████████| 907/907 [05:29<00:00,  2.75it/s]
Epochs: 10 | Train Loss:  0.078 | Train Accuracy:  0.766 | Val Loss:  0.348 | Val Accuracy:  0.346
```

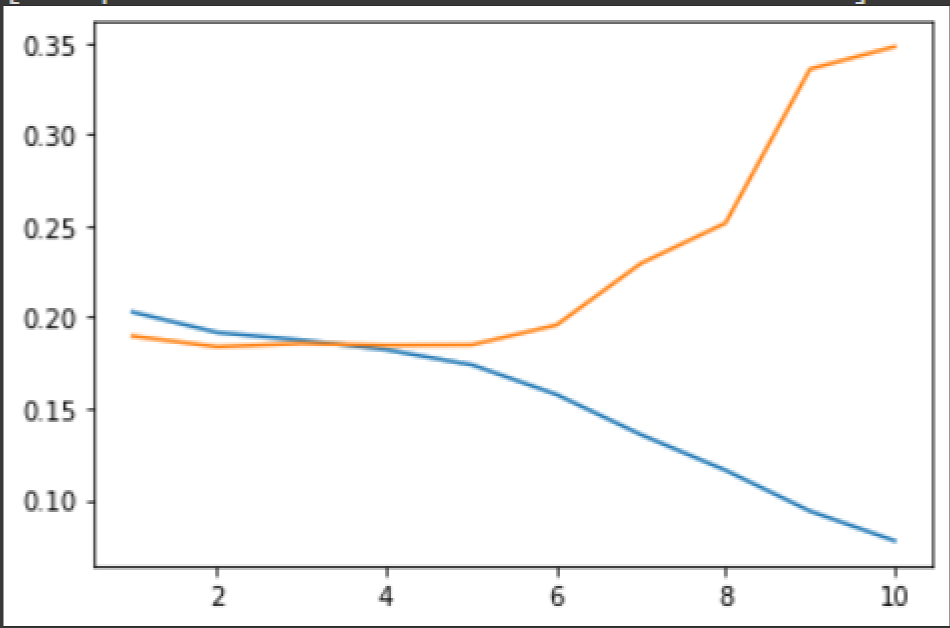
```
x_axis = np.linspace(1, EPOCHS, EPOCHS, endpoint=True)
plt.plot(x_axis, summary['train_acc'])
plt.plot(x_axis, summary['val_acc'])
```

```
[<matplotlib.lines.Line2D at 0x7fe83973c250>]
```



```
plt.plot(x_axis, summary['train_loss'])
plt.plot(x_axis, summary['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7fe838b14d50>]
```

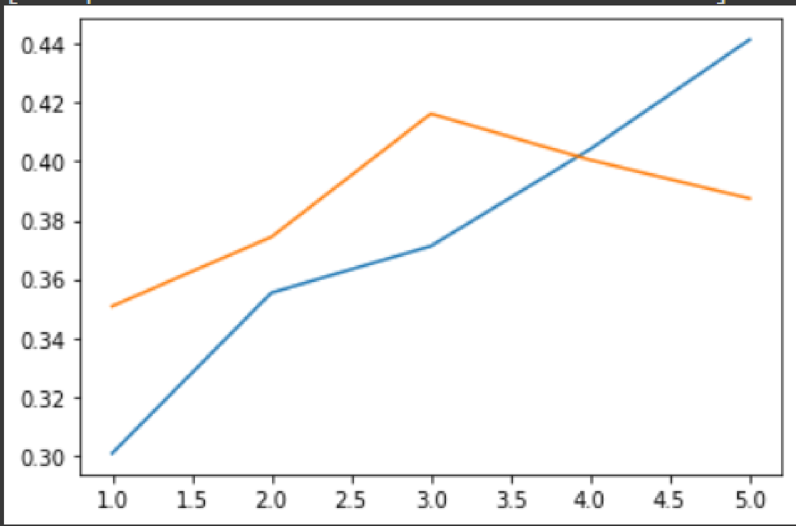


As we can see the model start to overfitted after epoch 5 so we trained it only for 5 epochs and the results became better

```
[ ] EPOCHS = 5
    model, summary = train_model(net, train_dataloader, val_dataloader)

    100%|██████████| 907/907 [05:30<00:00,  2.74it/s]
    Epochs: 1 | Train Loss:  0.204 | Train Accuracy:  0.301 | Val Loss:  0.190 | Val Accuracy:  0.351
    100%|██████████| 907/907 [05:29<00:00,  2.76it/s]
    Epochs: 2 | Train Loss:  0.192 | Train Accuracy:  0.355 | Val Loss:  0.185 | Val Accuracy:  0.374
    100%|██████████| 907/907 [05:28<00:00,  2.76it/s]
    Epochs: 3 | Train Loss:  0.187 | Train Accuracy:  0.371 | Val Loss:  0.180 | Val Accuracy:  0.416
    100%|██████████| 907/907 [05:28<00:00,  2.76it/s]
    Epochs: 4 | Train Loss:  0.182 | Train Accuracy:  0.404 | Val Loss:  0.179 | Val Accuracy:  0.401
    100%|██████████| 907/907 [05:29<00:00,  2.76it/s]
    Epochs: 5 | Train Loss:  0.172 | Train Accuracy:  0.441 | Val Loss:  0.186 | Val Accuracy:  0.387
```

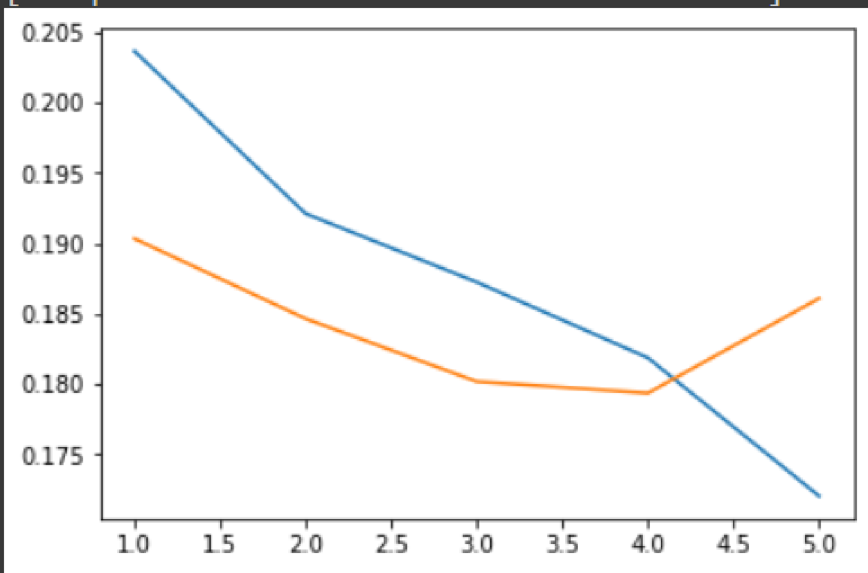
```python
x_axis = np.linspace(1, EPOCHS, EPOCHS, endpoint=True)
plt.plot(x_axis, summary['train_acc'])
plt.plot(x_axis, summary['val_acc'])
```

```
[<matplotlib.lines.Line2D at 0x7fe838d16f90>]
```




```python
plt.plot(x_axis, summary['train_loss'])
plt.plot(x_axis, summary['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7fe830bc3410>]
```

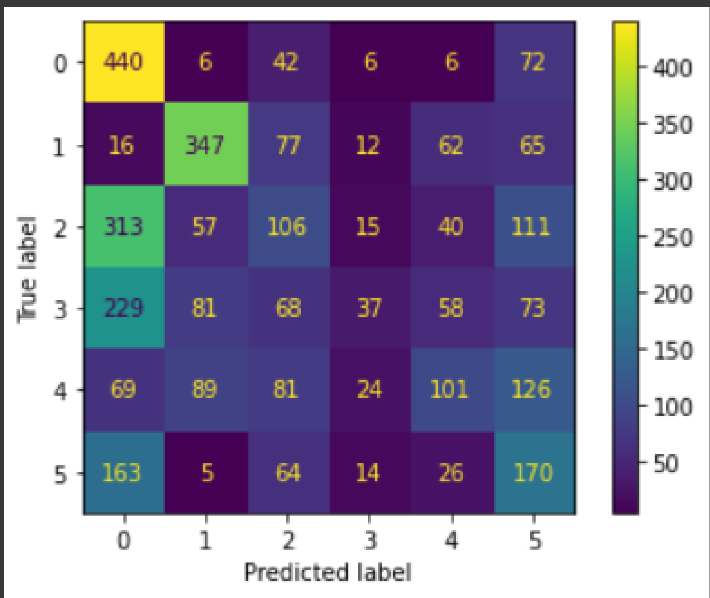


Test and Results

```
[ ] y_pred,y_true,total_acc = test_model(net,test_dataloader)

    Test Accuracy:  0.367
```

```
from sklearn.metrics import f1_score

f1_score(y_true, y_pred, average=None)

array([0.48834628, 0.59621993, 0.1962963 , 0.11314985, 0.25798212,
       0.3210576 ])
```

```
[ ] from sklearn.metrics import ConfusionMatrixDisplay
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
    plt.show()
```



7-Code

[Notebook1]

[Notebook2]