CSE 676: Deep Learning

FINAL PROJECT REPORT

Exploring Emotions: Employing Deep Learning for Facial Expression Recognition

TEAM MEMBERS	ASSIGNMENT PART	CONTRIBUTION (%)
VAMSIKRISHNA JAYARAMAN	Coding, Report	33.33% (1/3rd)
NAGESHWARAN GANDHIRAJ	Coding, Report	33.33% (1/3rd)
GOUTHAM REDDY ENUGULA	Coding, Report	33.33% (1/3rd)

The objective of this project is to develop an intelligent computer system through deep learning, capable of precisely identifying facial expressions. The FER2013 dataset is used for this purpose. The primary aim is to create a robust system that can effectively navigate the complexities involved in interpreting various emotions in real-world scenarios. This project has implications in mental health monitoring, offering a potential tool for early detection and intervention based on facial cues.

Objectives:

- Train a deep learning model on FER2013 for facial expression recognition.
- Assess classification performance across different emotions.
- Enhance accuracy and explore model adaptability to diverse facial expressions and settings by experimenting with four image recognition architectures.
- Compare performance on various parameters and select the most effective model based on the evaluation results.

DATASET DESCRIPTION:

The Facial Expression Recognition Challenge dataset is a comprehensive collection of facial images tagged with appropriate emotions and organized into three essential columns. The first column, "Emotion," comprises numerical representations of the various emotional states depicted in the photos. These emotions might include happiness, sadness, surprise, rage, and others. The second column, marked "Usage," distinguishes between data intended for training and data allocated for testing. This section oversees the integrity of model evaluation and performance assessment. Finally, the "Pixels" column contains the grayscale intensity values of each pixel within the facial images, which are most likely stored in a manner suited for computational analysis, such as flattened arrays or matrix representations. With almost 35,000 rows, this dataset is a valuable resource for training and assessing machine learning algorithms charged with understanding human emotions through facial expressions.

Such activities have a wide range of applications across sectors, including human-computer interface, healthcare diagnostics, and security systems.

Dataset Display:

	emotion	Usage	Pixels
0	0	Training	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1	0	Training	151 150 147 155 148 133 111 140 170 174 182 15...
2	2	Training	231 212 156 164 174 138 161 173 182 200 106 38...
3	4	Training	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
4	6	Training	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...
...	...	...	...
35882	6	PrivateTest	50 36 17 22 23 29 33 39 34 37 37 37 39 43 48 5...
35883	3	PrivateTest	178 174 172 173 181 188 191 194 196 199 200 20...
35884	0	PrivateTest	17 17 16 23 28 22 19 17 25 26 20 24 31 19 27 9...
35885	3	PrivateTest	30 28 28 29 31 30 42 68 79 81 77 67 67 71 63 6...
35886	2	PrivateTest	19 13 14 12 13 16 21 33 50 57 71 84 97 108 122...

35887 rows × 3 columns

Basic Statistics:

```
1 data.describe()
```

	emotion
count	35887.000000
mean	3.323265
std	1.873819
min	0.000000
25%	2.000000
50%	3.000000
75%	5.000000
max	6.000000

In the above figure, describe() method is used for generating the descriptive analysis for the dataset. It will calculate and display the summary statistics for all the numerical features in the dataset.

DATA CLEANING / PREPROCESSING:

1. **Converting Pixel Value from string to Numpy array:**

```
data['Pixels'] = data['Pixels'].apply(lambda x: np.array(x.split(), dtype = np.uint8))
```

```
data.head(7)
```

	emotion	Usage	Pixels
0	0	Training	[70, 80, 82, 72, 58, 58, 60, 63, 54, 58, 60, 4...
1	0	Training	[151, 150, 147, 155, 148, 133, 111, 140, 170, ...
2	2	Training	[231, 212, 156, 164, 174, 138, 161, 173, 182, ...
3	4	Training	[24, 32, 36, 30, 32, 23, 19, 20, 30, 41, 21, 2...
4	6	Training	[4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 15, 23...
5	2	Training	[55, 55, 55, 55, 55, 54, 60, 68, 54, 85, 151, ...
6	4	Training	[20, 17, 19, 21, 25, 38, 42, 42, 46, 54, 56, 6...

This converts the pixel values stored as strings in the 'Pixels' column of the DataFrame into NumPy arrays of unsigned 8-bit integers, making the data ready for further processing.

2. **Dividing the Dataset:**

```
data['Usage'].value_counts()
```

```
Training       28709
PublicTest      3589
PrivateTest     3589
Name: Usage, dtype: int64
```

```
train_dataset = data[data['Usage'] == 'Training'].drop(columns=['Usage'])
val_dataset = data[data['Usage'] == 'PublicTest'].drop(columns=['Usage'])
test_dataset = data[data['Usage'] == 'PrivateTest'].drop(columns=['Usage'])
```

Dividing the original dataset into three separate datasets: one for training, one for validation, and one for testing. Each dataset is stripped of the 'Usage' column, as it's no longer necessary for modeling purposes after the split.

3. **Finding the Mean and Standard Deviation value:**

Mean and standard deviation is calculated for all sets of data. With this mean and standard deviation we can normalize the pixels.

- **For Training Dataset:**

```
pixel_columns = train_dataset['Pixels']

all_pixels = [pixel for sublist in pixel_columns for pixel in sublist]

total_mean_tr = np.mean(all_pixels)
total_std_tr = np.std(all_pixels)
print("Total mean value of all pixels in training :", total_mean_tr)
print("Total std value of all pixels in training :", total_std_tr)
```

```
Total mean value of all pixels in training : 129.47433955331468
Total std value of all pixels in training : 65.02727348443116
```

- **For Validation Dataset:**

```
pixel_columns = val_dataset['Pixels']

all_pixels = [pixel for sublist in pixel_columns for pixel in sublist]

total_mean_val = np.mean(all_pixels)
total_std_val = np.std(all_pixels)
print("Total mean value of all pixels in Validation:", total_mean_val)
print("Total std value of all pixels in training :", total_std_val)
```

```
Total mean value of all pixels in Validation: 128.98103217586143
Total std value of all pixels in training : 65.0060051006662
```

- **For Testing Dataset:**

```
pixel_columns = test_dataset['Pixels']

all_pixels = [pixel for sublist in pixel_columns for pixel in sublist]

total_mean_test = np.mean(all_pixels)
total_std_test = np.std(all_pixels)
print("Total mean value of all pixels in test:", total_mean_test)
print("Total std value of all pixels in training :", total_std_test)
```

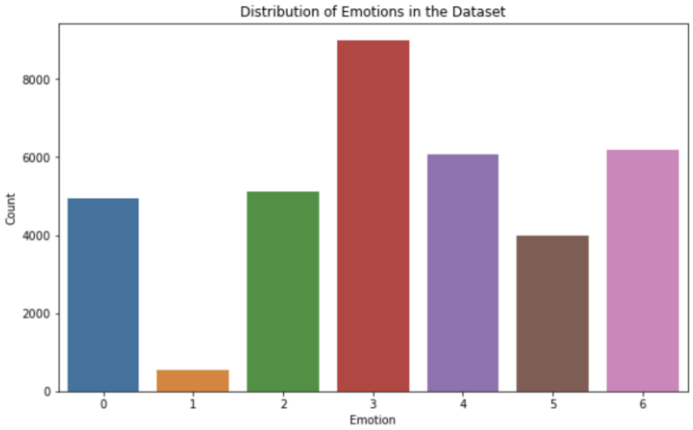
```
Total mean value of all pixels in test: 129.08289168678988
Total std value of all pixels in training : 65.35151723799241
```

Visualizations:

Distribution of dataset:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.countplot(x='emotion', data = data)
plt.title('Distribution of Emotions in the Dataset')
plt.xlabel('Emotion')
plt.ylabel('Count')
plt.show()
```



Sample Images:

```
sample_indices = np.random.choice(len(data), 5, replace=False)
emotions = {0: 'Angry', 1: 'Disgust', 2: 'Fear', 3: 'Happy', 4: 'Sad', 5: 'Surprise', 6: 'Neutral'}
fig, axes = plt.subplots(1,5, figsize=(15, 5))
op = []
for i, index in enumerate(sample_indices):
    emotion = data['emotion'][index]
    pixels = data['Pixels'][index]
    image = pixels.reshape(48, 48)

    axes[i].imshow(image, cmap='gray')
    op.append(emotions[emotion])
    axes[i].axis('off')

plt.show()
print(op)
```

```
['Surprise', 'Angry', 'Happy', 'Angry', 'Surprise']
```

Models:

Base Model:

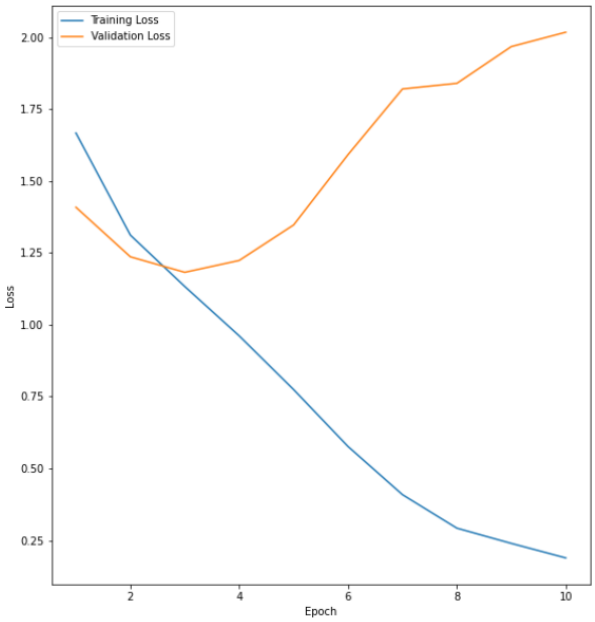
The network's architecture consists of several layers with a substantial number of trainable parameters, amounting to over 40 million. Our model employs a multilayered approach, including convolutional layers, ReLU activation

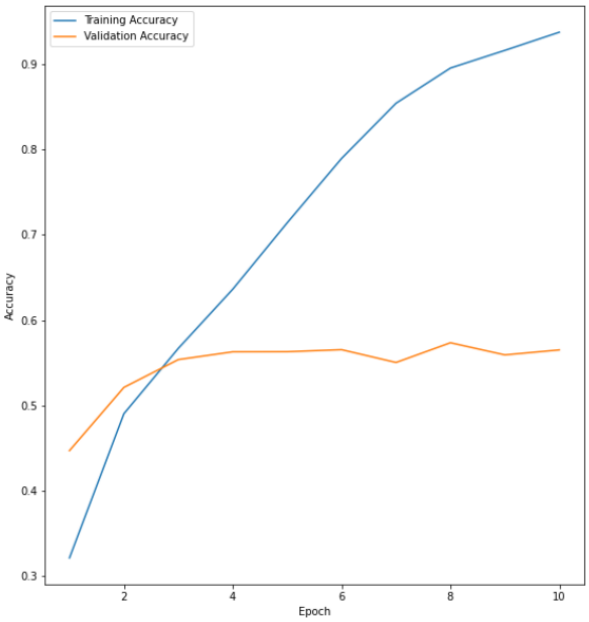
functions, pooling layers, and fully connected layers, along with dropout for regularization. Such a composition is standard in deep learning for capturing intricate patterns within the data.

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
BaseModel                                --
├─Sequential: 1-1                        --
│    └─Conv2d: 2-1                       1,792
│    └─ReLU: 2-2                         --
│    └─MaxPool2d: 2-3                    --
│    └─Conv2d: 2-4                       73,856
│    └─ReLU: 2-5                         --
│    └─MaxPool2d: 2-6                    --
│    └─Conv2d: 2-7                       295,168
│    └─ReLU: 2-8                         --
│    └─Conv2d: 2-9                       590,080
│    └─ReLU: 2-10                        --
│    └─MaxPool2d: 2-11                   --
│    └─Conv2d: 2-12                      1,180,160
│    └─ReLU: 2-13                        --
│    └─Conv2d: 2-14                      2,359,808
│    └─ReLU: 2-15                        --
│    └─MaxPool2d: 2-16                   --
├─Sequential: 1-2                        --
│    └─Linear: 2-17                      18,878,464
│    └─ReLU: 2-18                        --
│    └─Dropout: 2-19                     --
│    └─Linear: 2-20                      16,781,312
│    └─ReLU: 2-21                        --
│    └─Dropout: 2-22                     --
│    └─Linear: 2-23                      28,679
==========================================================================
Total params: 40,189,319
Trainable params: 40,189,319
Non-trainable params: 0
==========================================================================
```

Loss and Accuracy Graph:

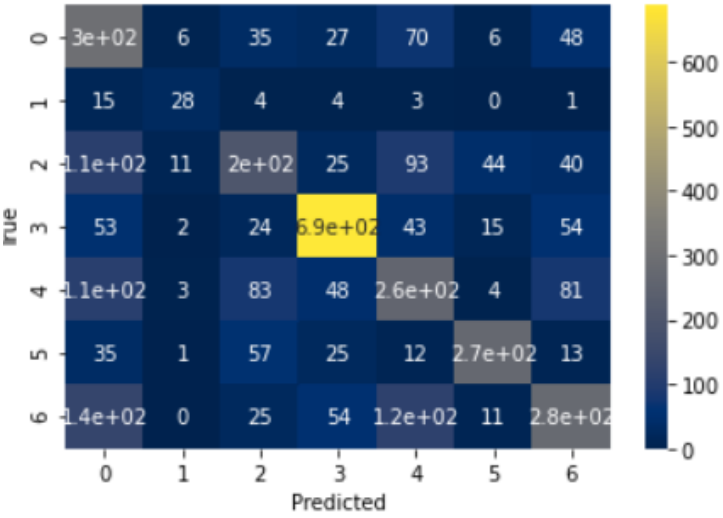
Throughout the training over 10 epochs, the model's training loss consistently decreased, implying successful learning. However, an increase in validation loss suggests that the model may be too closely fitted to the training data, failing to generalize well to unseen data. Additionally, while the training accuracy approached perfection, the validation accuracy plateaued at a much lower value, reinforcing concerns of overfitting. So, our plan after this checkpoint is to apply some regularization techniques to the model.





Confusion Matrix:

The confusion matrix exposed substantial misclassifications between specific classes. The mix-up between classes 2 and 3 was particularly noticeable, hinting at potential similarities in features that the model struggles to differentiate.



Accuracy Precision, Recall and F1 Score:

```
Accuracy: 56.56%
Precision: 0.5800385034035689
Recall: 0.5656171635553079
F1: 0.5675131020863855
```

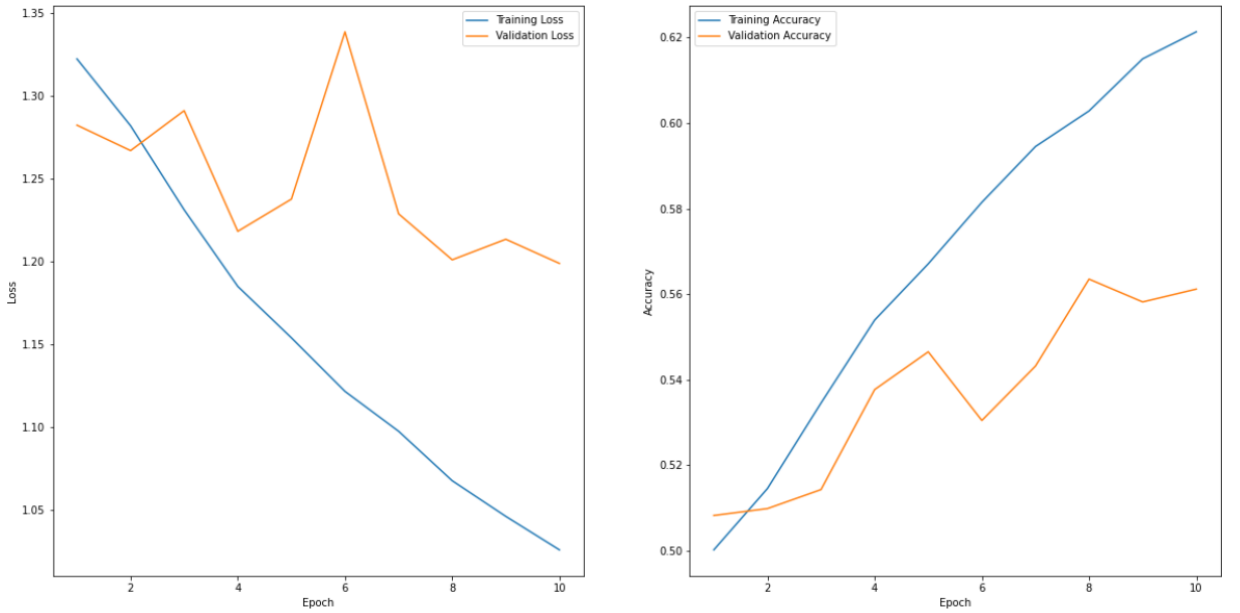
For Base Model we have tuned with the below Hyper Parameters

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 0.001, weight_decay = 0.001)
```

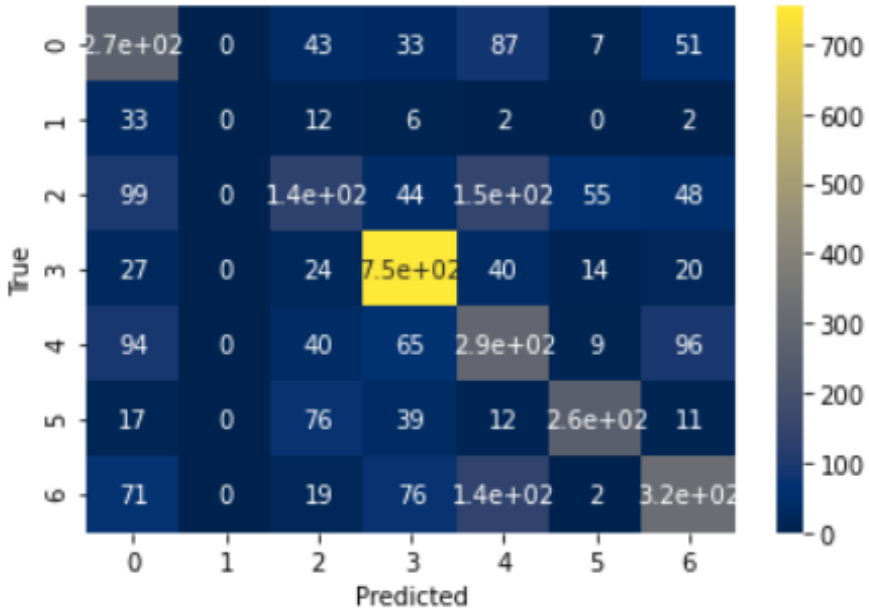
Now the Resultant Output is

```
Accuracy: 56.51%
Precision: 0.5544317087455066
Recall: 0.5650599052660908
F1: 0.5542045585389876
```

The below are the Accuracy Graphs



Confusion Matrix:

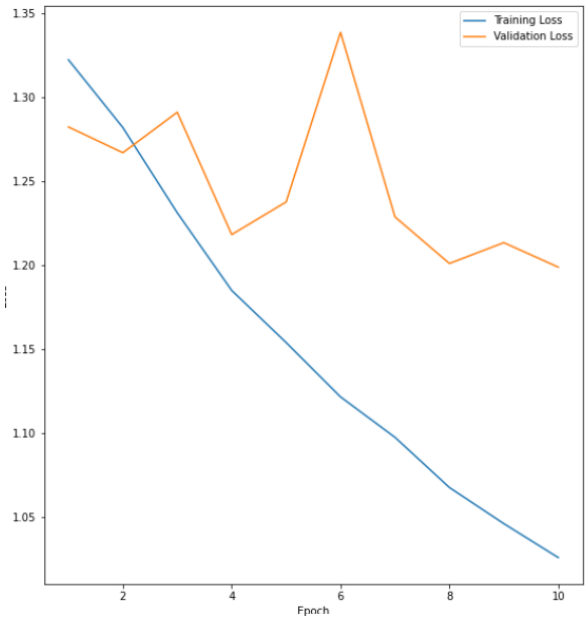


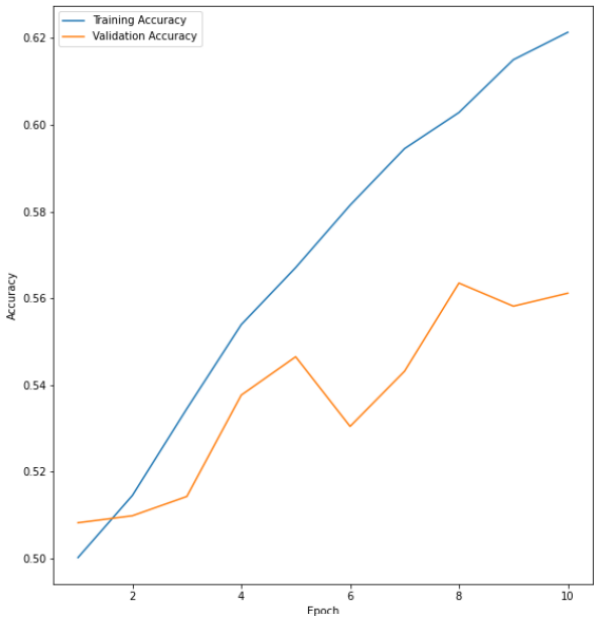
VGG13 Model:

I set up VGG13 with multiple convolutional layers, followed by ReLU activation functions and max-pooling layers. Ending with fully connected layers, the model boasted an impressive 34.6 million parameters.

```
==========================================================================
Layer (type:depth-idx)                             Param #
==========================================================================
VGG13                                              --
├─Sequential: 1-1                                  --
│    └─Conv2d: 2-1                                 1,792
│    └─ReLU: 2-2                                   --
│    └─Conv2d: 2-3                                 36,928
│    └─ReLU: 2-4                                   --
│    └─MaxPool2d: 2-5                              --
│    └─Conv2d: 2-6                                 73,856
│    └─ReLU: 2-7                                   --
│    └─Conv2d: 2-8                                 147,584
│    └─ReLU: 2-9                                   --
│    └─MaxPool2d: 2-10                             --
│    └─Conv2d: 2-11                                295,168
│    └─ReLU: 2-12                                  --
│    └─Conv2d: 2-13                                590,080
│    └─ReLU: 2-14                                  --
│    └─MaxPool2d: 2-15                             --
│    └─Conv2d: 2-16                                1,180,160
│    └─ReLU: 2-17                                  --
│    └─Conv2d: 2-18                                2,359,808
│    └─ReLU: 2-19                                  --
│    └─MaxPool2d: 2-20                             --
│    └─Conv2d: 2-21                                2,359,808
│    └─ReLU: 2-22                                  --
│    └─Conv2d: 2-23                                2,359,808
│    └─ReLU: 2-24                                  --
│    └─MaxPool2d: 2-25                             --
├─Sequential: 1-2                                  --
│    └─Linear: 2-26                                8,392,704
│    └─ReLU: 2-27                                  --
│    └─Linear: 2-28                                16,781,312
│    └─ReLU: 2-29                                  --
│    └─Linear: 2-30                                28,679
==========================================================================
Total params: 34,607,687
Trainable params: 34,607,687
Non-trainable params: 0
```

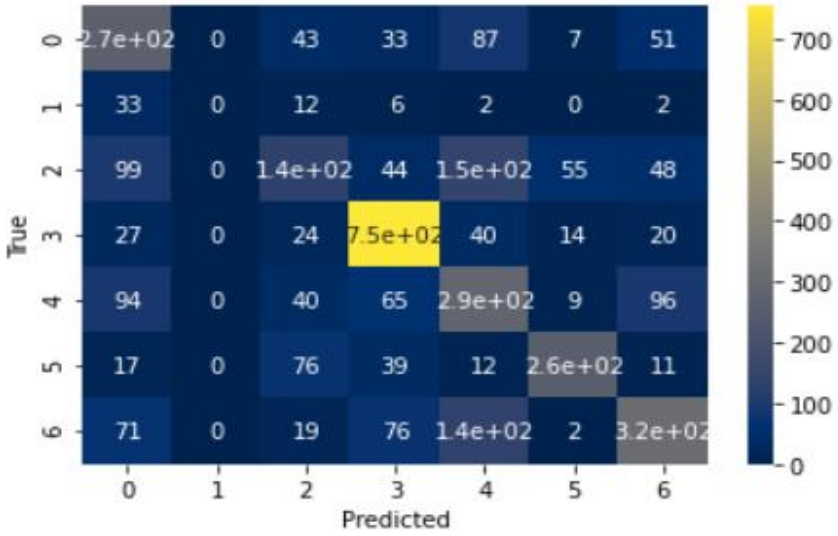
Loss and Accuracy Graph:






```
Accuracy: 56.51%
Precision: 0.5544317087455066
Recall: 0.5650599052660908
F1: 0.5542045585389876
```

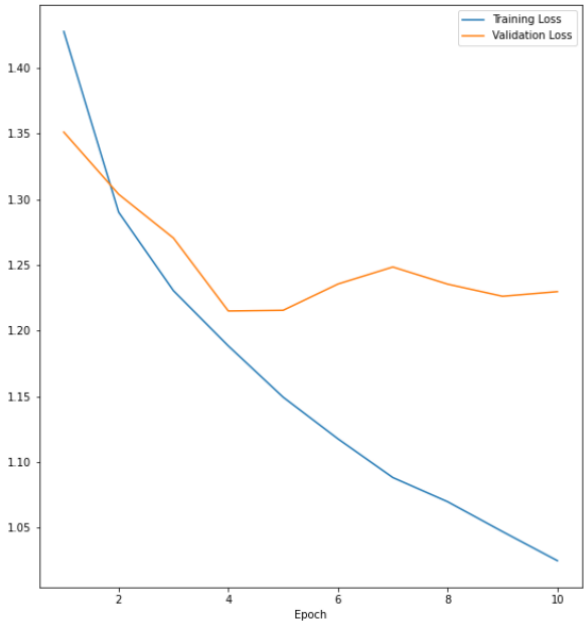
Confusion Matrix:

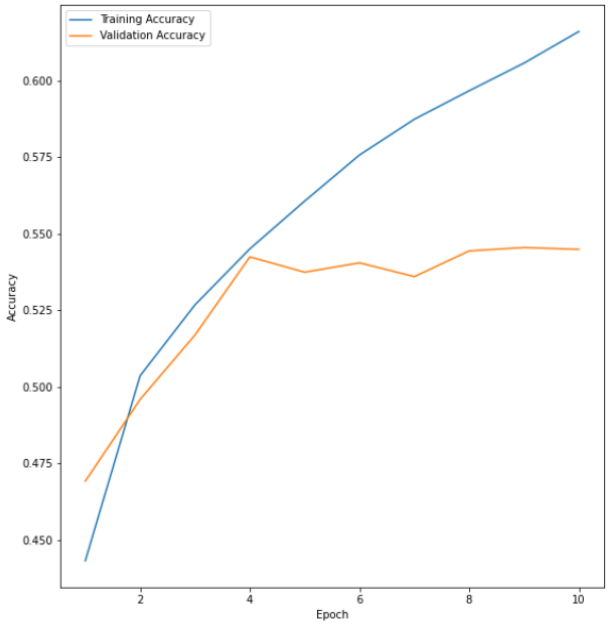


VGG model Trained with Hyper Parameters

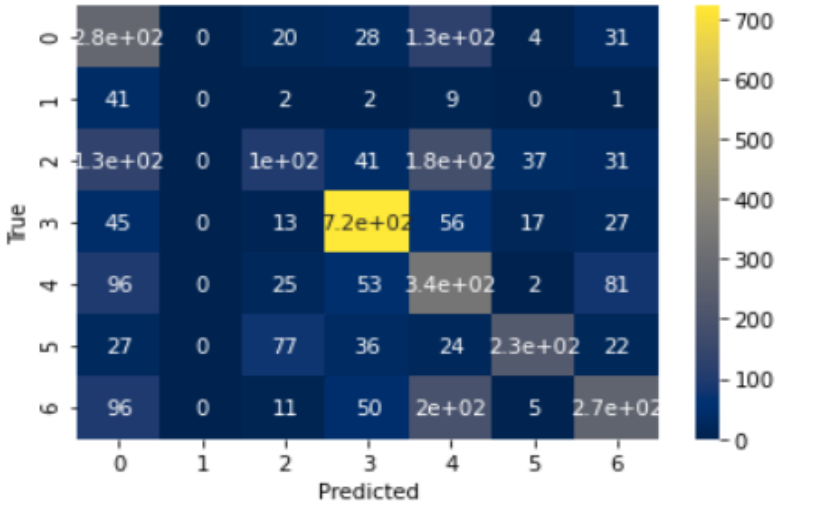
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_vgg1.parameters(), lr=0.001, weight_decay = 0.001)
```

Accuracy: 54.03%
Precision: 0.5547116680775224
Recall: 0.540261911395932
F1: 0.5312878880846599





Confusion Matrix:



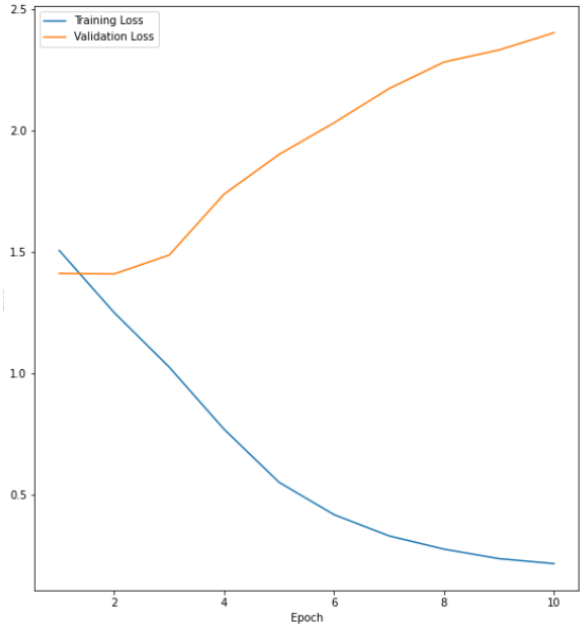
Resnet:

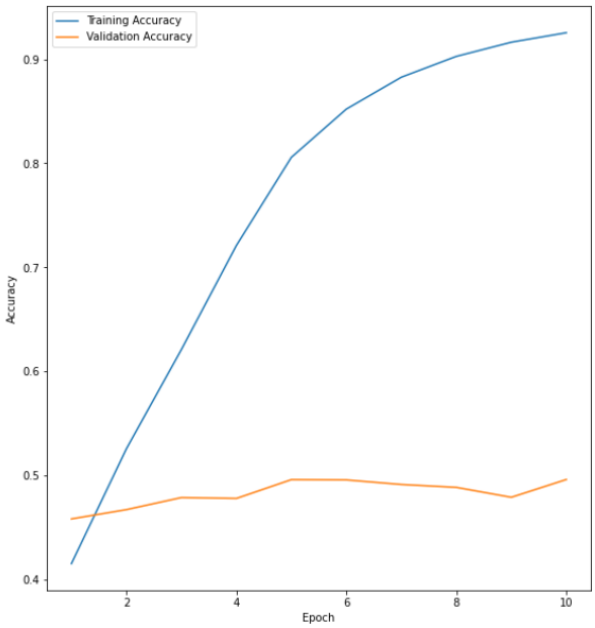
Model:

```
==========================================================================
Layer (type:depth-idx)                             Param #
==========================================================================
ResNetB                                            --
├─Conv2d: 1-1                                      9,408
├─BatchNorm2d: 1-2                                 128
├─ReLU: 1-3                                        --
├─MaxPool2d: 1-4                                   --
├─Sequential: 1-5                                  --
│    └─BasicBlock: 2-1                             --
│    │    └─Conv2d: 3-1                            36,864
│    │    └─BatchNorm2d: 3-2                       128
│    │    └─ReLU: 3-3                              --
│    │    └─Conv2d: 3-4                            36,864
│    │    └─BatchNorm2d: 3-5                       128
│    └─BasicBlock: 2-2                             --
│    │    └─Conv2d: 3-6                            36,864
│    │    └─BatchNorm2d: 3-7                       128
│    │    └─ReLU: 3-8                              --
│    │    └─Conv2d: 3-9                            36,864
│    │    └─BatchNorm2d: 3-10                      128
├─Sequential: 1-6                                  --
│    └─BasicBlock: 2-3                             --
│    │    └─Conv2d: 3-11                           73,728
│    │    └─BatchNorm2d: 3-12                      256
│    │    └─ReLU: 3-13                             --
│    │    └─Conv2d: 3-14                           147,456
│    │    └─BatchNorm2d: 3-15                      256
│    │    └─Sequential: 3-16                       8,448
│    └─BasicBlock: 2-4                             --
│    │    └─Conv2d: 3-17                           147,456
│    │    └─BatchNorm2d: 3-18                      256
│    │    └─ReLU: 3-19                             --
│    │    └─Conv2d: 3-20                           147,456
│    │    └─BatchNorm2d: 3-21                      256
├─Sequential: 1-7                                  --
│    └─BasicBlock: 2-5                             --
│    │    └─Conv2d: 3-22                           294,912
│    │    └─BatchNorm2d: 3-23                      512
│    │    └─ReLU: 3-24                             --
│    │    └─Conv2d: 3-25                           589,824
│    │    └─BatchNorm2d: 3-26                      512
│    │    └─Sequential: 3-27                       33,280
│    └─BasicBlock: 2-6                             --
│    │    └─Conv2d: 3-28                           589,824
│    │    └─BatchNorm2d: 3-29                      512
│    │    └─ReLU: 3-30                             --
│    │    └─Conv2d: 3-31                           589,824
│    │    └─BatchNorm2d: 3-32                      512
├─Sequential: 1-8                                  --
│    └─BasicBlock: 2-7                             --
│    │    └─Conv2d: 3-33                           1,179,648
│    │    └─BatchNorm2d: 3-34                      1,024
│    │    └─ReLU: 3-35                             --
│    │    └─Conv2d: 3-36                           2,359,296
│    │    └─BatchNorm2d: 3-37                      1,024
│    │    └─Sequential: 3-38                       132,096
│    └─BasicBlock: 2-8                             --
│    │    └─Conv2d: 3-39                           2,359,296
│    │    └─BatchNorm2d: 3-40                      1,024
│    │    └─ReLU: 3-41                             --
│    │    └─Conv2d: 3-42                           2,359,296
│    │    └─BatchNorm2d: 3-43                      1,024
├─AdaptiveAvgPool2d: 1-9                           --
├─Linear: 1-10                                     3,591
==========================================================================
Total params: 11,180,103
Trainable params: 11,180,103
Non-trainable params: 0
==========================================================================
```

We implemented a ResNetB variant with modifications suitable for our specific task, as evidenced by the provided model summary. The network consists of several convolutional (Conv2d), batch normalization (BatchNorm2d), and Rectified Linear Unit (ReLU) layers, as well as pooling layers (MaxPool2d) and fully connected layers (Linear).

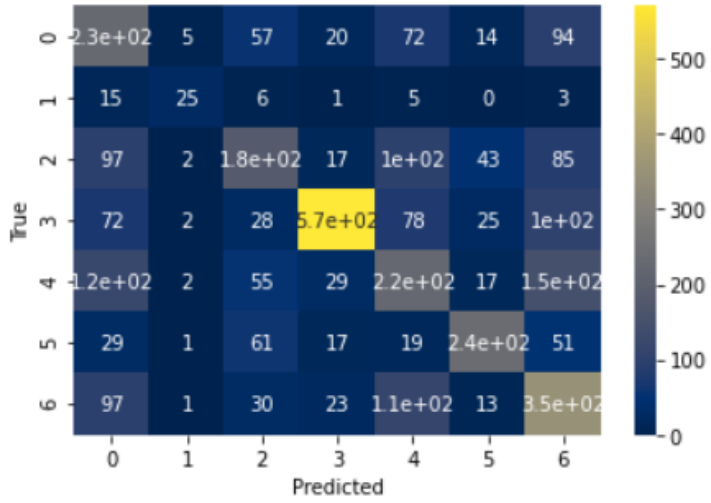
Loss and Accuracy Graph:





Accuracy: 50.63%
Precision: 0.540202535811535
Recall: 0.5062691557536918
F1: 0.5156901734201564

Confusion Matrix:

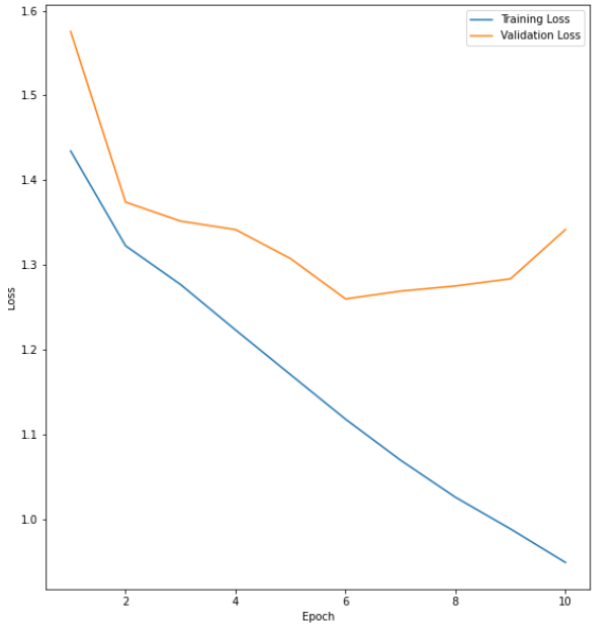


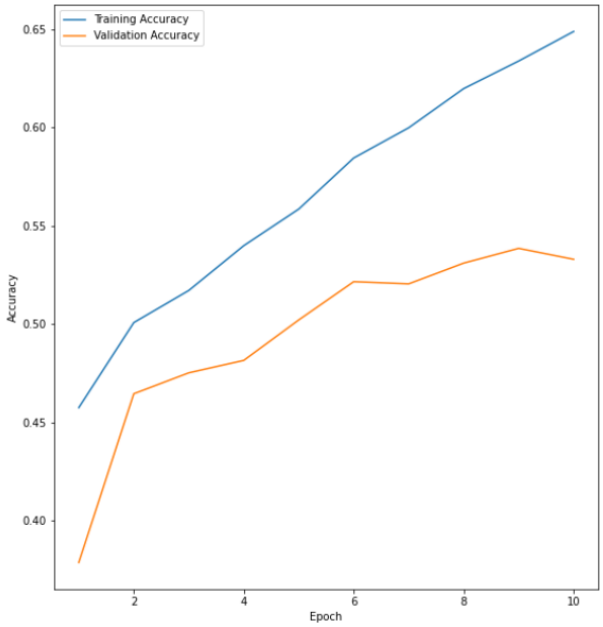
We have trained with Hyper Parameters

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(modelResnet0.parameters(), lr = 0.001, weight_decay = 0.001)
```

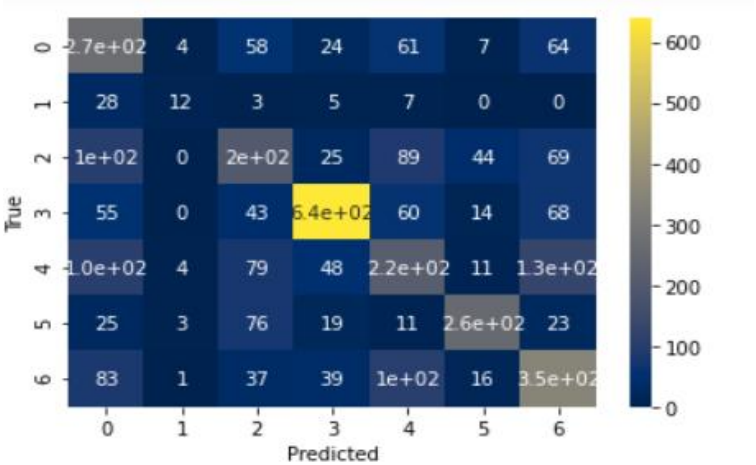
Accuracy: 54.30%
Precision: 0.5566150543633681
Recall: 0.5430482028420173
F1: 0.5459982597362945

Accuracy Graphs





Confusion Matrix

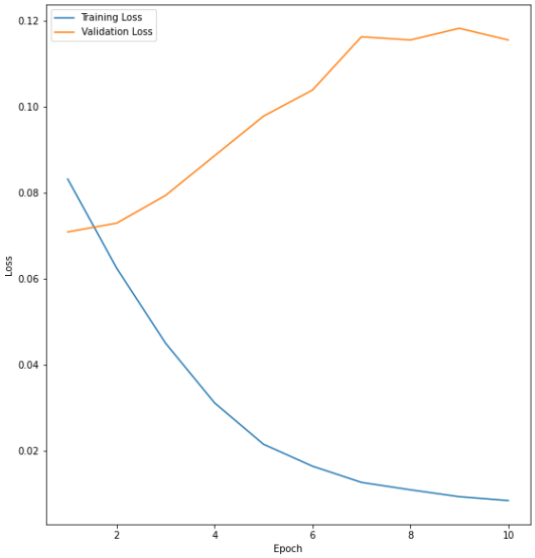


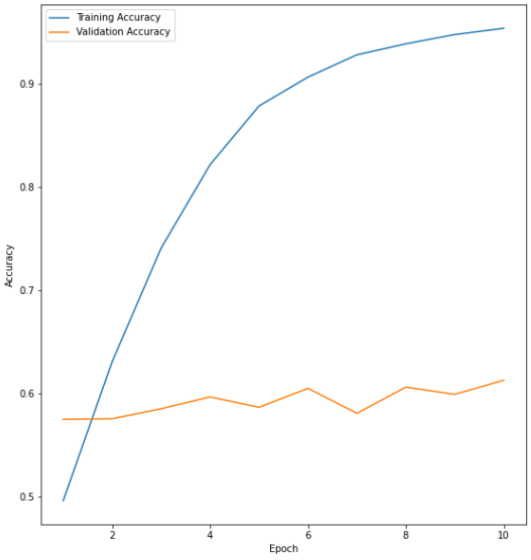
Now we Implemented new Model
RESNET-50

We have implemented new model Resnet 50 this made our model to increase the Accuracy as shown below.

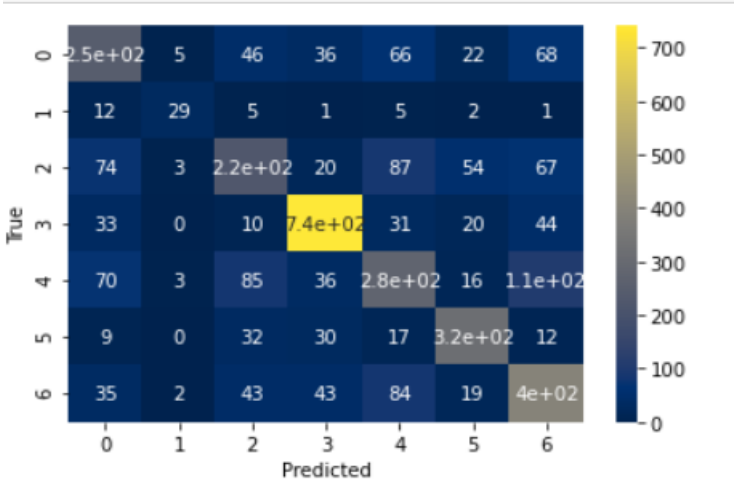
```
Accuracy: 62.25%
Precision: 0.6173411703639246
Recall: 0.6224575090554472
F1: 0.6186796475942096
```

Accuracy Graphs



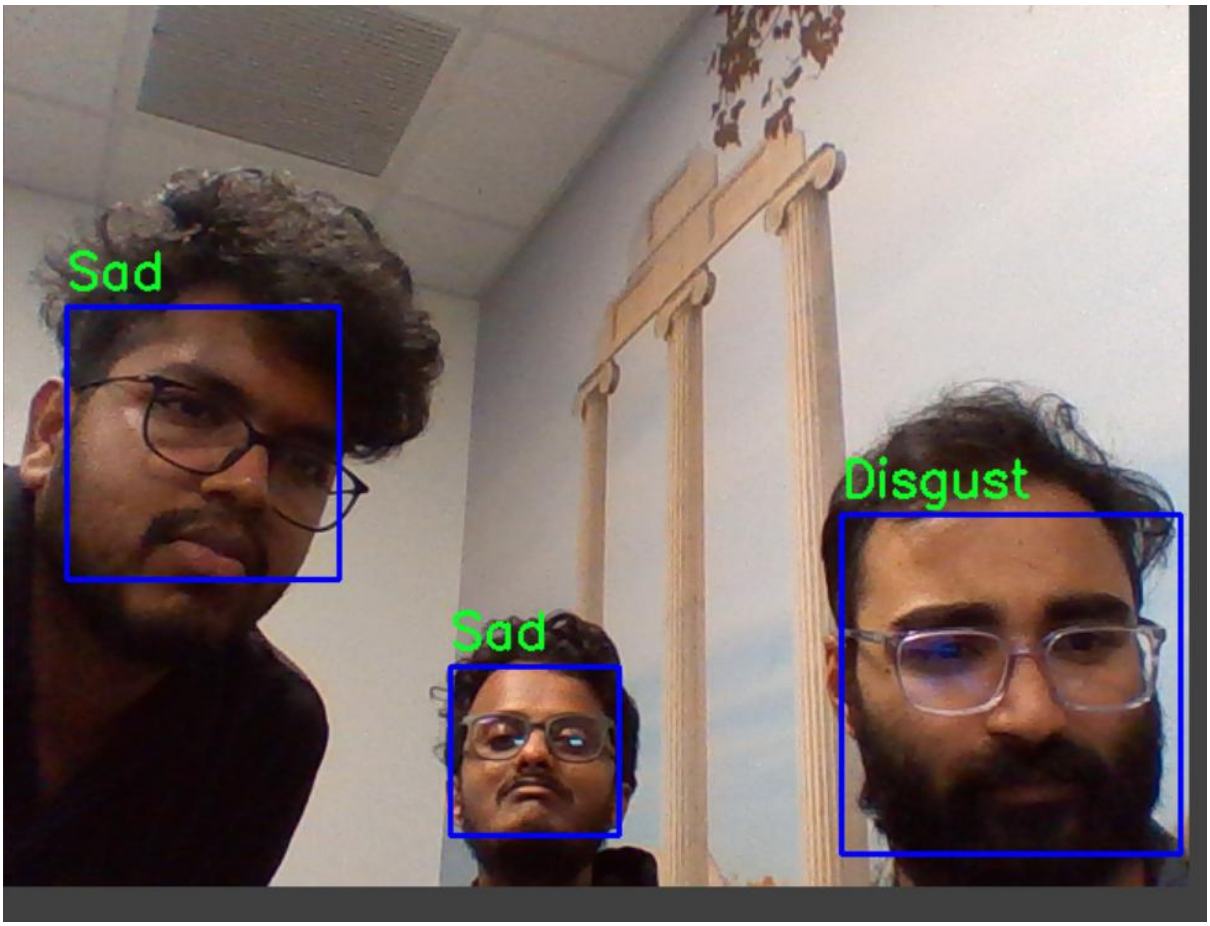


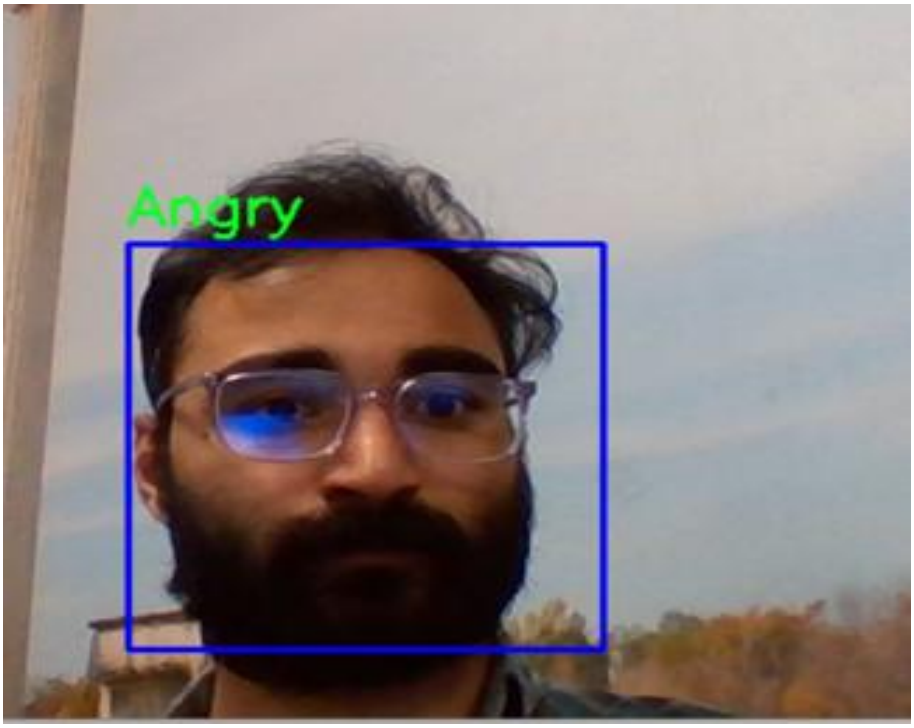
Confusion Matrix:

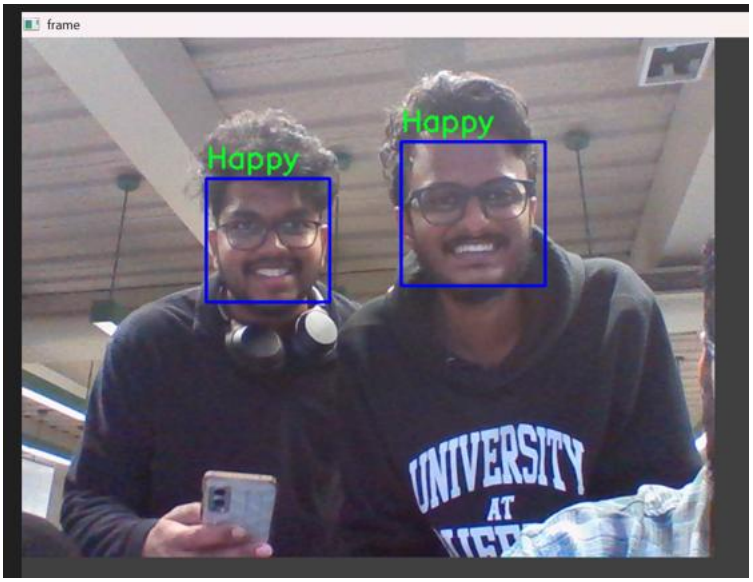


#BONUS

We have integrated our model with CV2 and webcam that detects the emotions which captured from the webcam and shows the output that which emotion we are we have attached few output Screenshots below:




Angry


frame
Happy
Happy
UNIVERSITY
AT

Real World Deep Learning Applications:
We have deployed our model it can help in Real World applications
Example:

- **Health Care**
- **Automobile Industry**

Health Care: In this for example if a patient in a situation unable to talk then based on emotions we can detect what exactly the patient feeling.

Automobile Industry: We can Consider a driver in car, Truck if the person is not feeling well or sleepy or Fainted situation then Based on emotion required alerts will be sent this is also much helpful.

Novelty:

We have come up with Resnet-50 this models use more Conv Layers different from pre models so that we train the model in a better way and we get Better Results

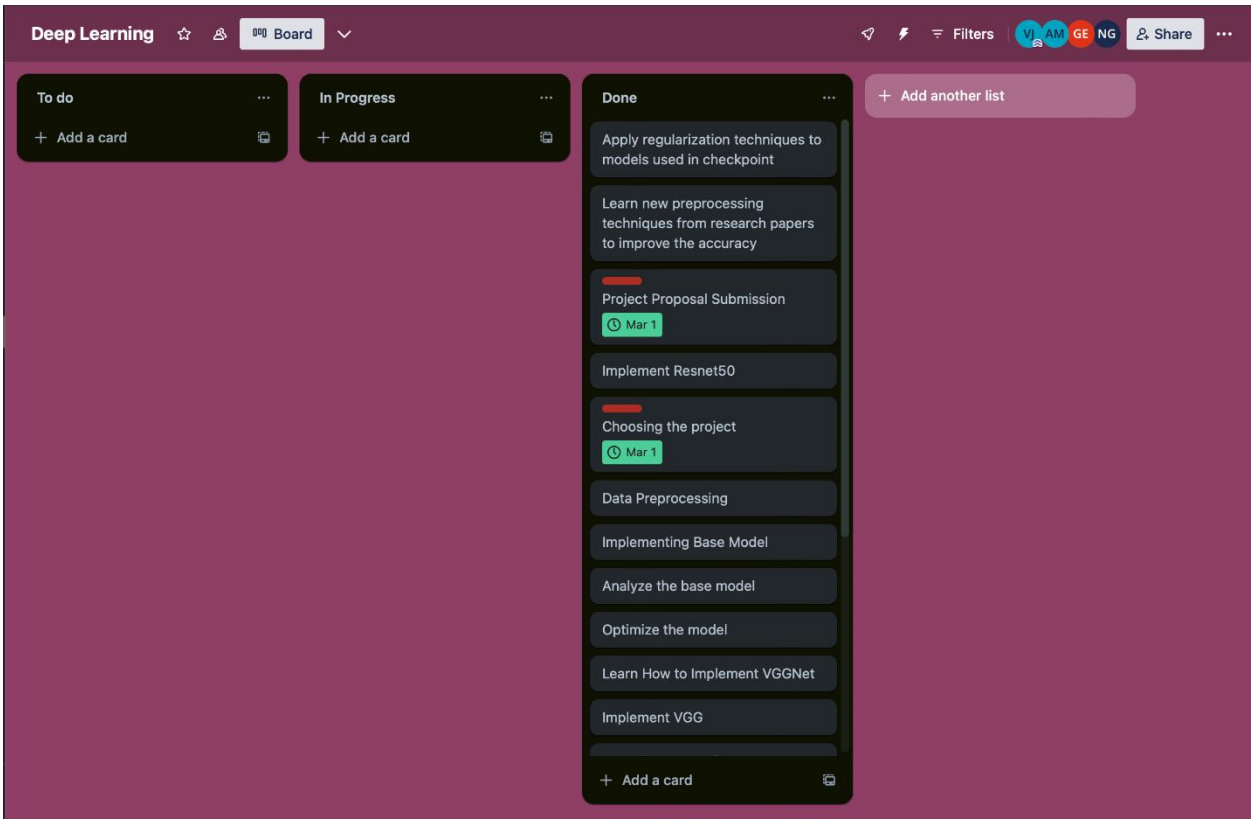
Deploy the Model:
We have integrated our Model with the webcam so that it Detects the Human Emotions and tell that what our emotion exactly is.

#References

https://medium.com/analytics-vidhya/building-a-real-time-emotion-detector-towards-machine-with-e-q-c20b17f89220

https://arxiv.org/abs/2206.09509

Trello Checkpoints:



Improvements for the project after the checkpoint submission:

We tried various preprocessing techniques like dividing the pixel values by 255.0 and also other techniques like data augmentation, histogram equalization techniques. But our current preprocessing techniques seem good for now. In future we are planning to dig deep and learn new techniques by referring to more research papers on this Facial Expression Recognition problem. Obtaining a good accuracy on this dataset is quite challenging, but we are learning new things to improve the accuracy.

One thing we understood is that human facial expressions can be subtle and complex. Emotions like surprise and fear or disgust and anger can share similar facial features, which makes it challenging for the model to distinguish between them accurately. And also within the same class, expressions can vary greatly in intensity. Some expressions are underrepresented in the dataset so this can lead to a bias towards the more frequent classes. So, our plan after this checkpoint is to apply some regularization techniques to the above models and then implement some other models to improve the accuracy and finally detect the emotions realtime.

Now we are almost satisfied above conflicts and model that is detecting the emotions and also the accuracy also increased.