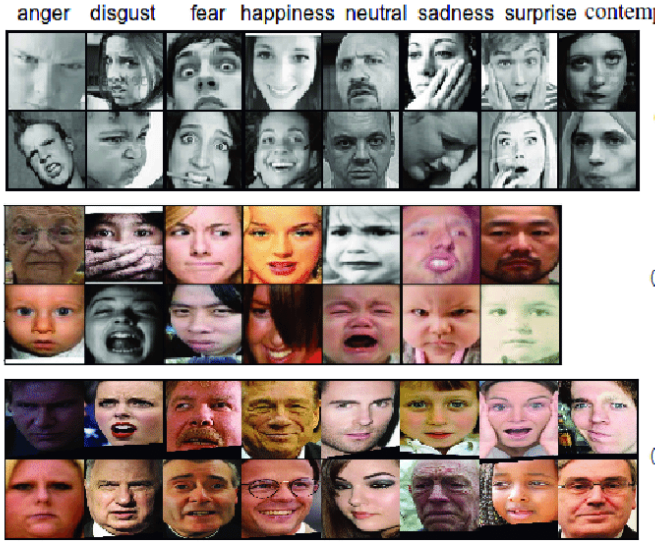
OVERVIEW OF EXISTING DATASETS USED

RAF-DB DATASET:

Label	Number of Images for 'Training'	Number of Images for 'Testing'
Surprise	1290	329
Fear	281	74
Disgust	717	160
Happiness	4772	1185
Sadness	1982	478
Anger	705	162
Neutral	2524	680



SOURCE:https://www.researchgate.net/figure/Sample-images-from-the-a-FER-b-RAF-DB-and-c-AffectNet-datasets_fig3_343035056

FED 2023 (FACIAL EXPRESSION DETECTION):

Label	Number of Images for 'Training'	Number of Images for 'Testing'
Amenability	239	189
Confusion	296	214
Aversion	394	238
Concentration	272	160

OAHEGA DATASET:

Label	Number of Images
Anger	1313
Happiness	3740
Neutral	4027
Sadness	3934
Surprise	1234

TOTAL:

Dataset Name(s)	Labeled Emotion	Number of Images for 'Training'	Number of Images for 'Testing'	Number of Images per Label
RAF-DB	Fear	281	74	**355**
	Disgust	717	160	**877**
RAF-DB & FED	Surprise	1260+1234	478	**2972**
	Happiness	4772+3740	1185	**9697**
	Sadness	1982+3934	478	**6394**
	Anger	705+1313	162	**2180**
	Neutral	2524+4027	680	**7231**
FER	Amenability	239	189	**428**
	Confusion	296	214	**510**
	Aversion	394	238	**632**
	Concentration	272	160	**432**
TOTAL				**31, 708**

JUSTIFICATION

Relevance: The labels Happy, Angry and Neutral from the RAF-DB and the FED datasets, our team was able to successfully gather a sufficient amount of data for training and testing for the labels Happiness, Anger and Neutral. These choices, as is likely with most participants of this project, were easy enough to both find and categorize. The FER dataset provides a few labeled sets of data that were concluded to suffice as samples of the Engaged class, which proved to be significantly more difficult when searching for datasets. The justification behind the choice of the 'Agree' and 'Think' labels (subsequently renamed 'Amenability' and 'Concentration' to maintain the consistent use of nouns within the labeling of emotions/expressions) have to do both with the body language and facial features of one who is active in thought. The 'Think' label indicated a clear depiction of active thinking , similarly, the 'Agree' images reflect not that of naivete but enthusiastic engagement.

IMAGE SOURCE INFORMATION

Dataset Name	Author(s)	Organization(s)	License	URL
RAF-DB	Deng, Weihong; Du, JunPing; Li, Shan.	Pattern Recognition and Intelligent System Laboratory, Beijing University of Posts and Telecommunications.	N/A	http://www.whdeng.cn/raf/model1.html
FED	Patel, Prajval	Kaggle	N/A	https://www.kaggle.com/datasets/prajvalpatel/facial-expression-detection-2023
FER	Kovenko, Volodymyr; Shevchuk, Vitalii.	Mendeley Data	CC BY-NC-SA 4.0	https://www.kaggle.com/datasets/sujaykapadnis/emotion-recognition-datase

DATA CLEANING

Techniques and methods:

Using both tools within skimage and numpy, our team was able to accomplish both the resizing, color scaling and re-organization of the datasets outlined in the previous section of this report. The following information will outline the tools and techniques used in the processing and transformation of the datasets and the subsequent resulting sets of images to be used in further sections of this project.

- Grayscale: using skimage's imsave function
- Resizing: using skimage.transform's resize function

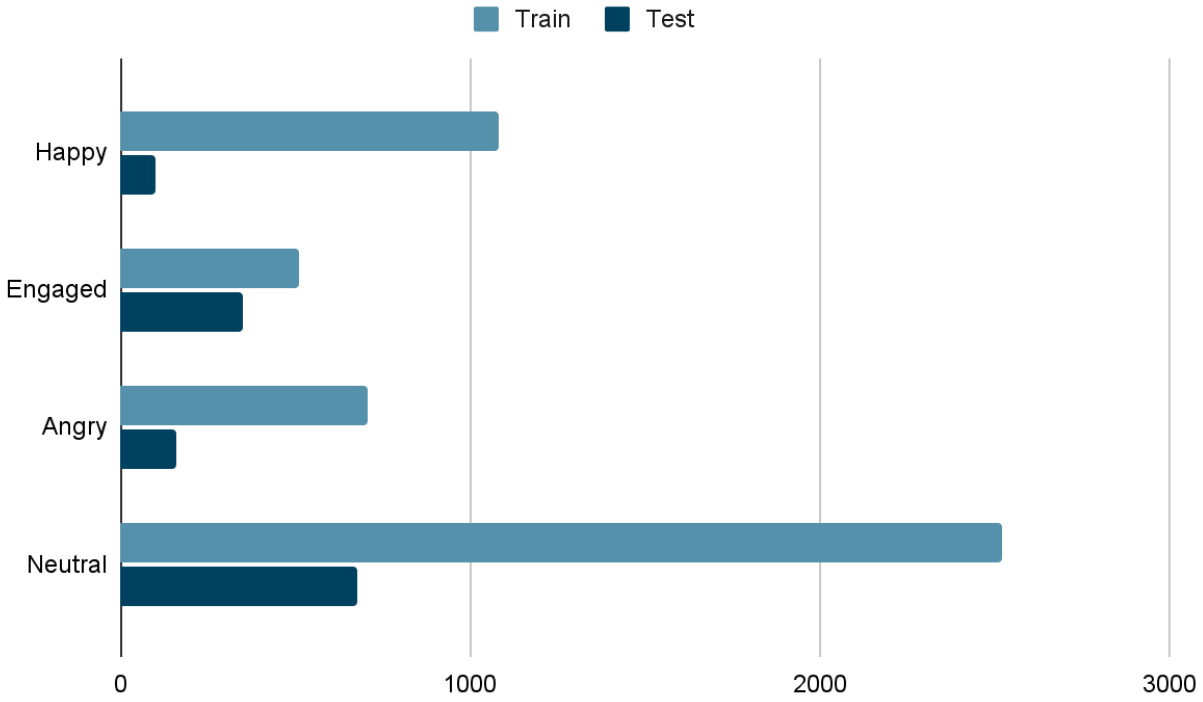
Challenges

One of the most obvious challenges encountered during cleaning was the merging of datasets, the relabeling of the final dataset and handling various file types (e.g. png, jpg). The

merging of datasets was solved by simply redirecting large samples of data deemed to be of the same expression/emotion into the necessary categorized files. Labeling is handled by creating a csv file with the modified image file names now under the correct labels. Finally, with careful attention paid to modifying images of various types, our team was able to manually input extensions as necessary.

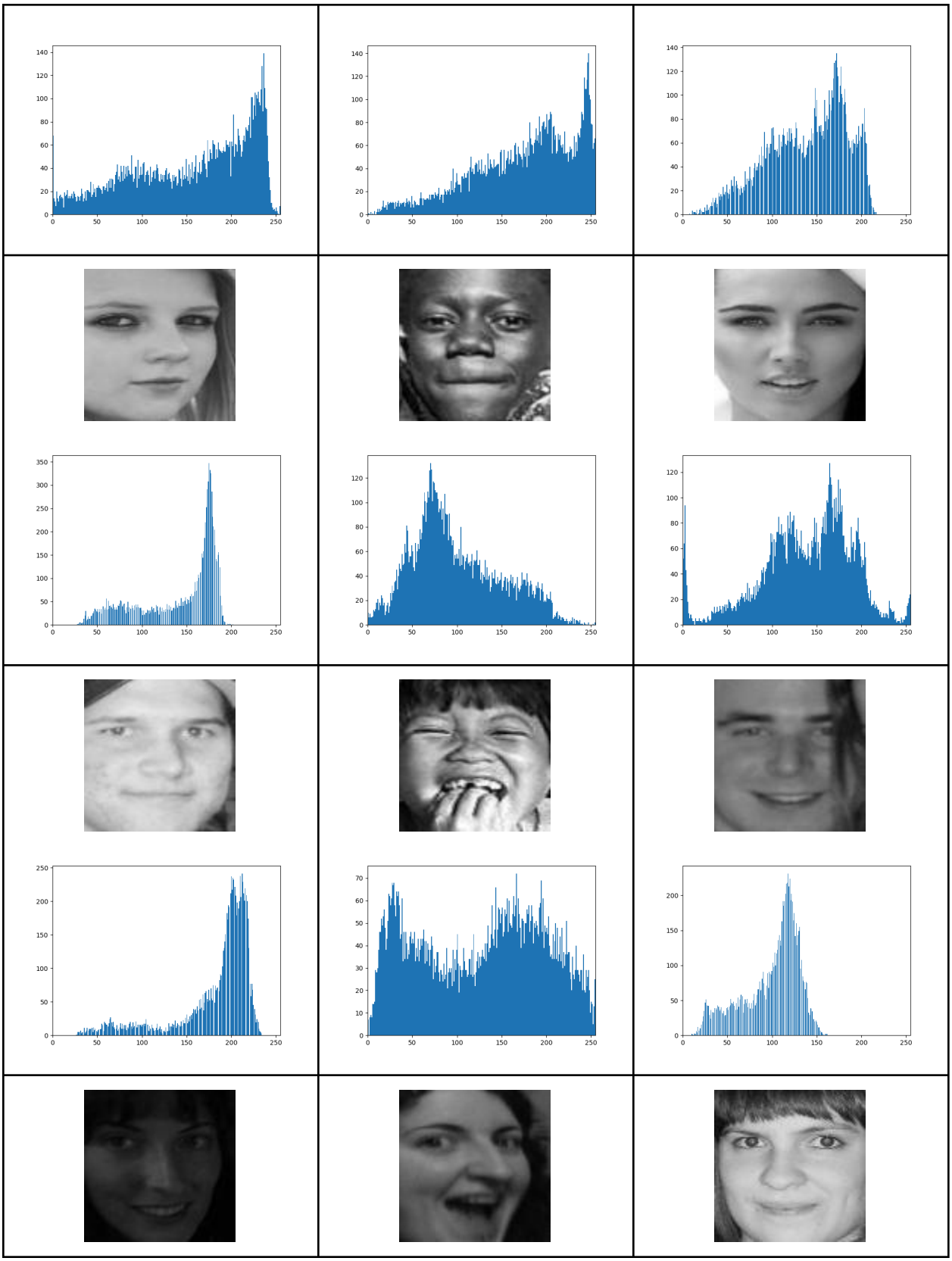
DATASET VISUALIZATION

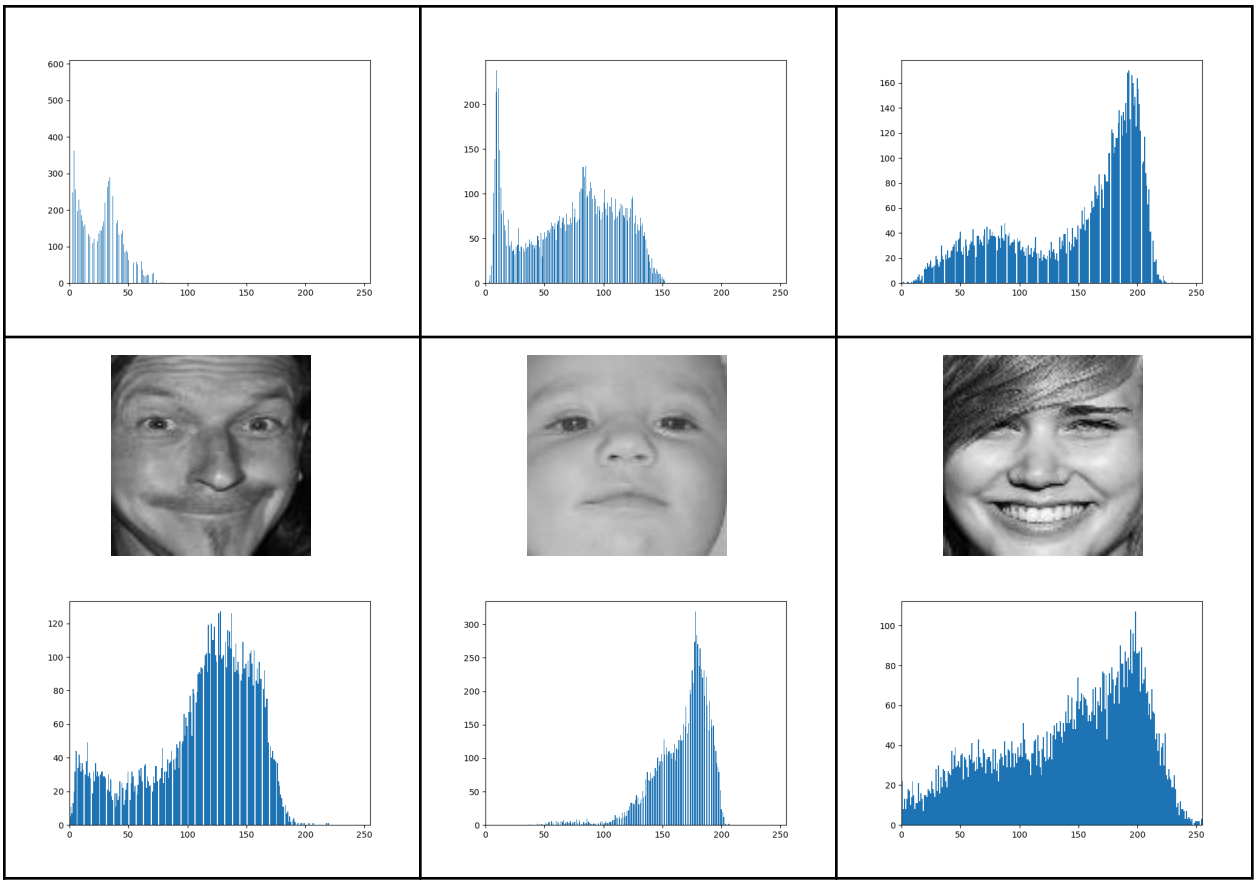
CLASS DISTRIBUTION:



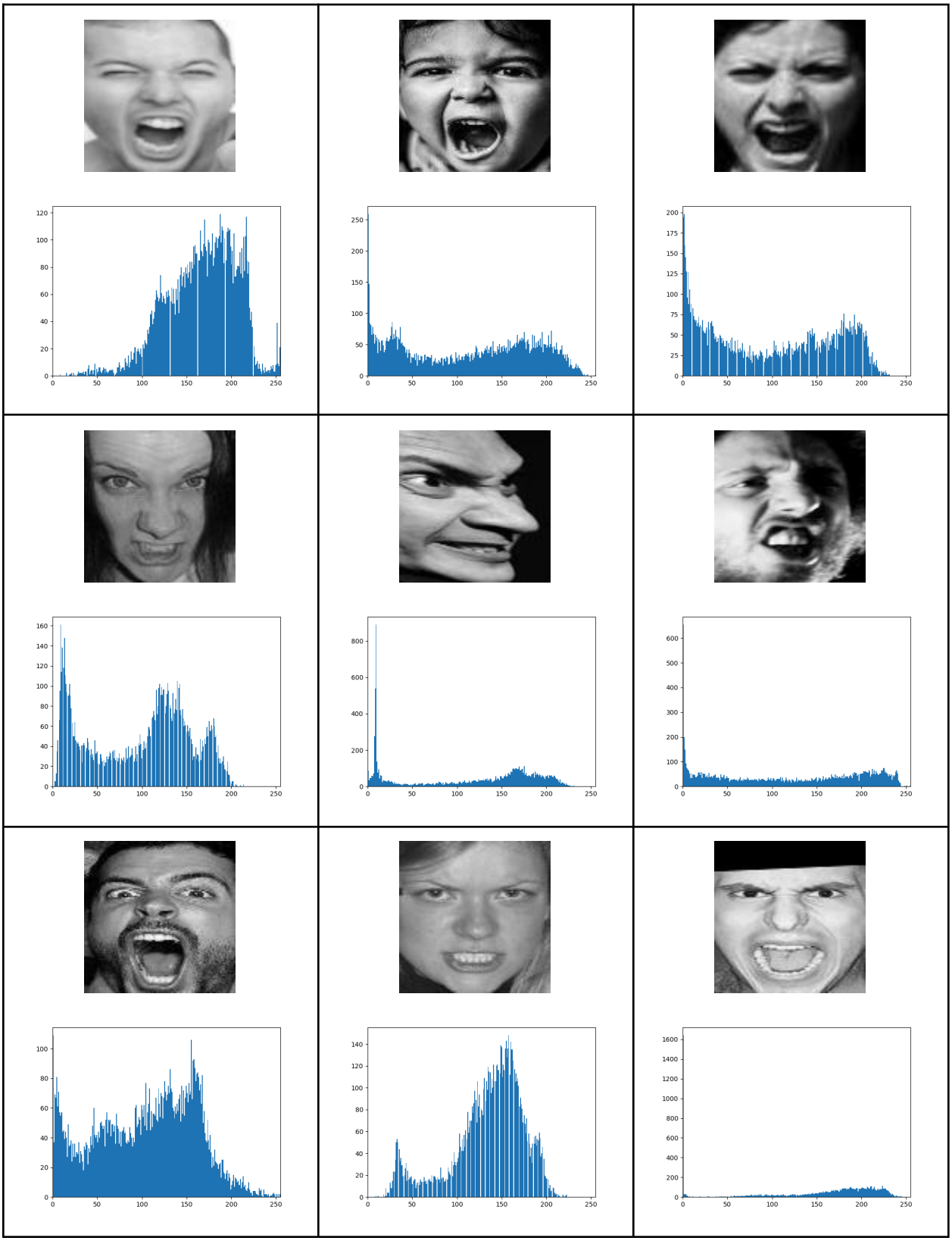
PIXEL INTENSITY DISTRIBUTION & SAMPLE IMAGES

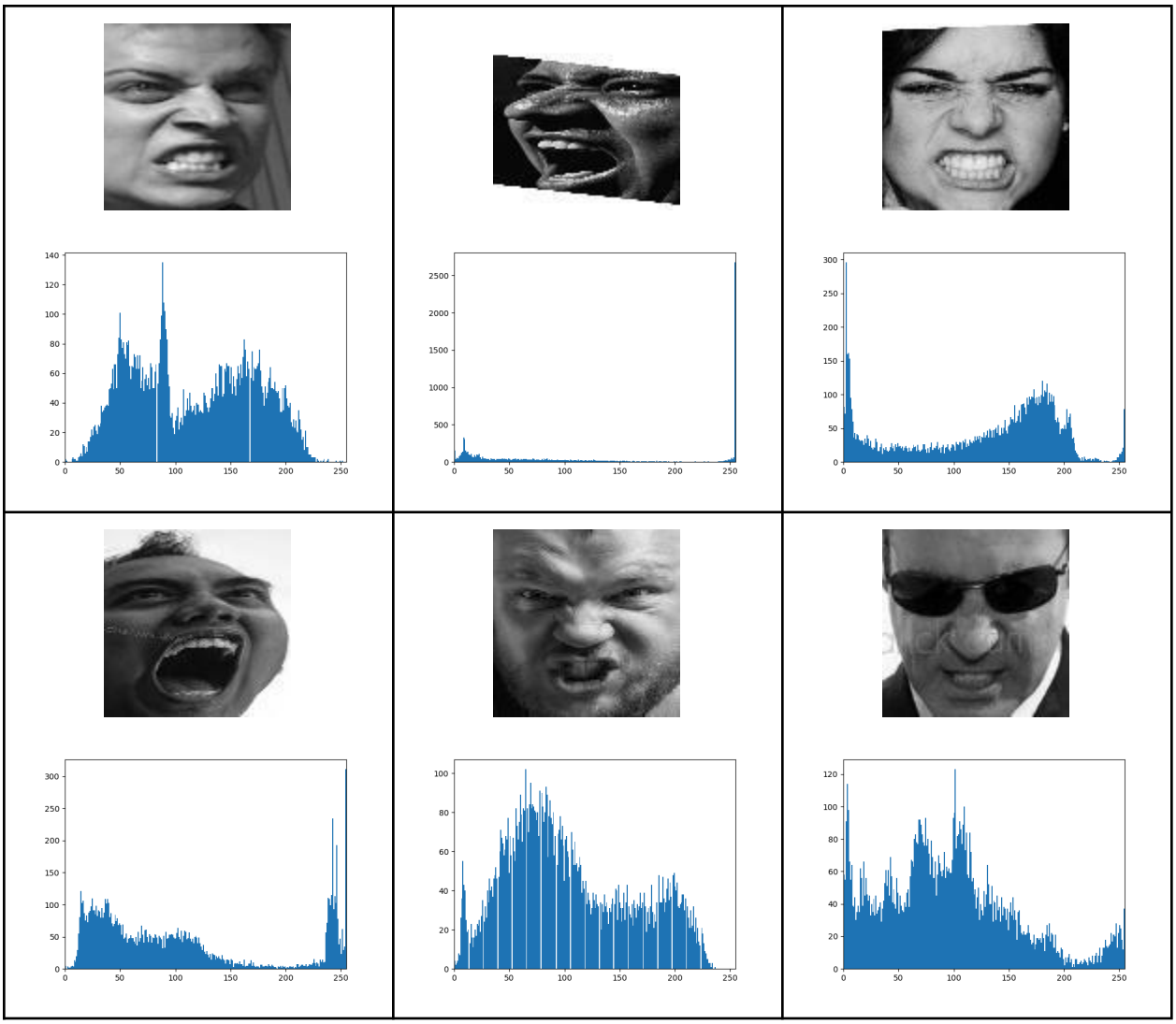
CLASS: HAPPY



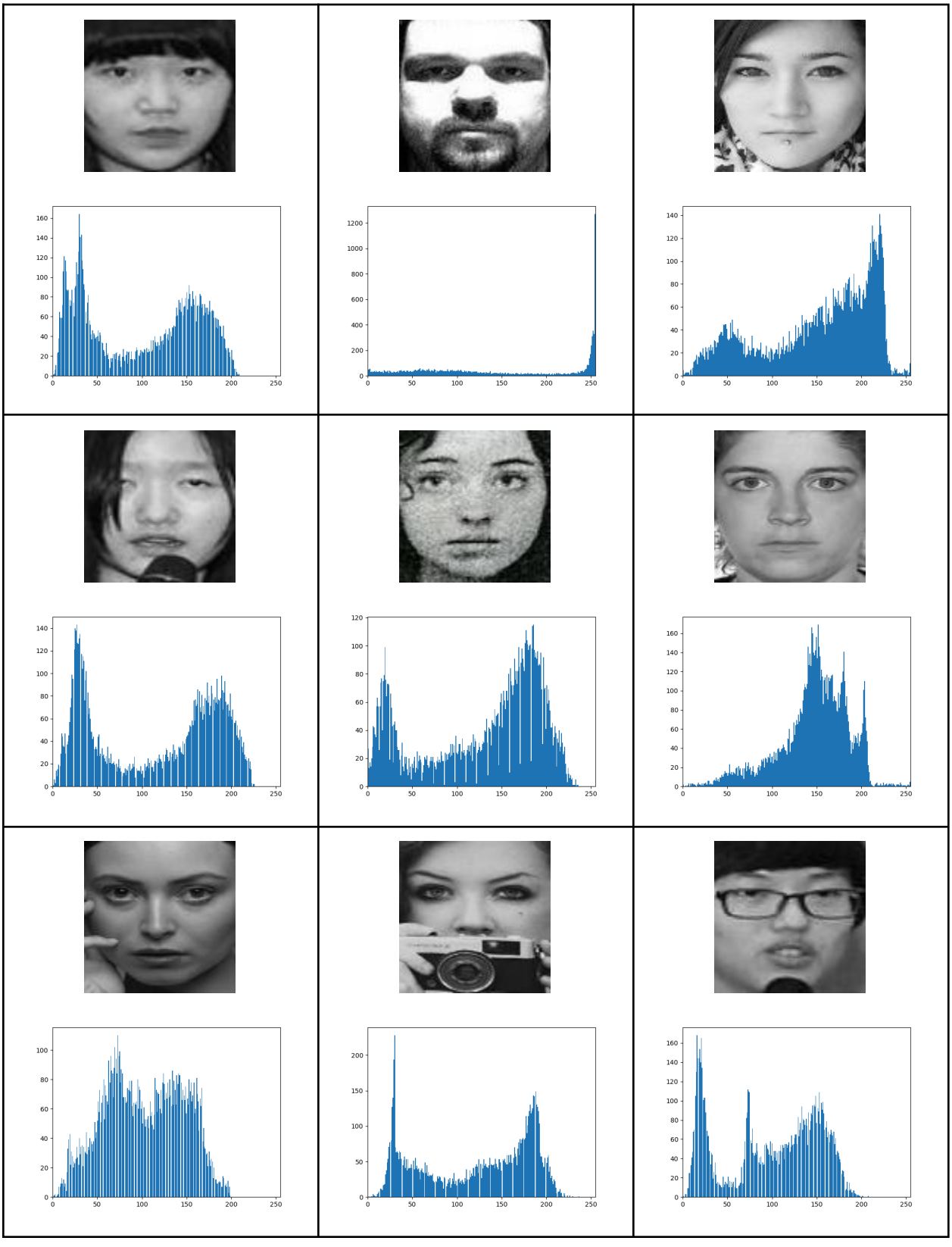

0 100 200 300 400 500 600
0 50 100 150 200 250
0 50 100 150 200
0 50 100 150 200 250
0 20 40 60 80 100 120 140 160
0 50 100 150 200 250
0 20 40 60 80 100 120
0 50 100 150 200 250
0 50 100 150 200 250
0 50 100 150 200 250
0 20 40 60 80 100
0 50 100 150 200 250

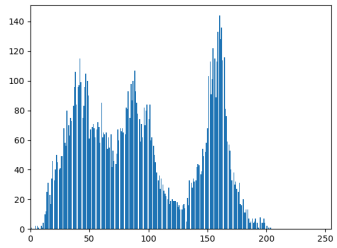
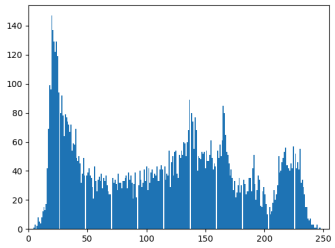
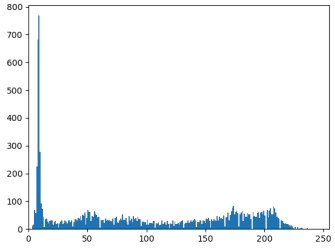
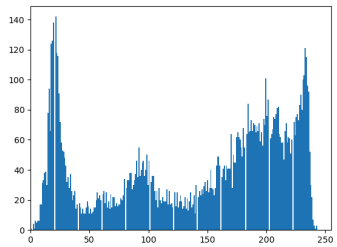
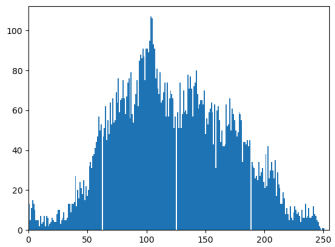
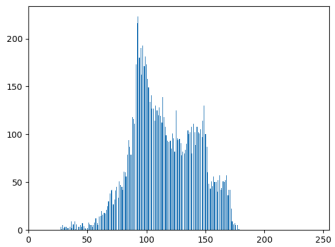
CLASS: ANGRY




0 20 40 60 80 100 120 140
0 50 100 150 200 250
0 500 1000 1500 2000 2500
0 50 100 150 200 250
0 50 100 150 200 250 300
0 50 100 150 200 250
0 50 100 150 200 250 300
0 50 100 150 200 250
0 20 40 60 80 100
0 50 100 150 200 250
0 20 40 60 80 100 120
0 50 100 150 200 250

CLASS: NEUTRAL



0 20 40 60 80 100 120 140
0 50 100 150 200 250

0 20 40 60 80 100 120 140
0 50 100 150 200 250

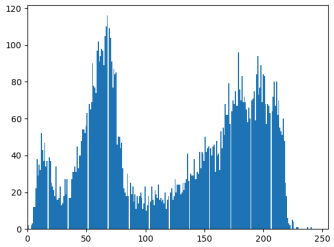
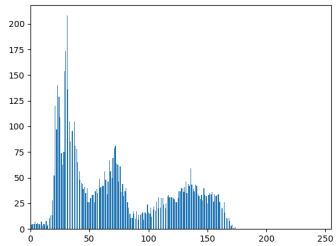
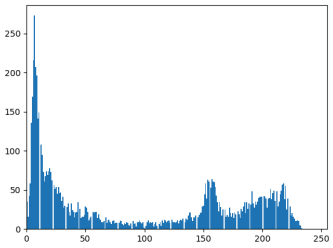
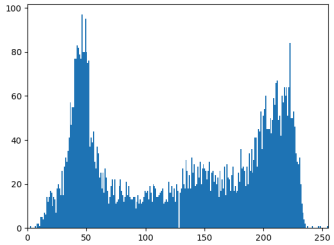
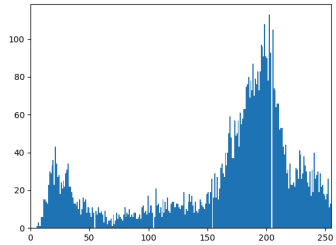
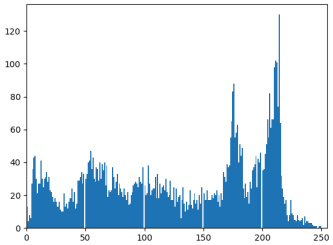
0 100 200 300 400 500 600 700 800
0 50 100 150 200 250

0 20 40 60 80 100 120 140
0 50 100 150 200 250

0 20 40 60 80 100
0 50 100 150 200 250

0 50 100 150 200
0 50 100 150 200 250

CLASS: ENGAGED

CNN MODEL ARCHITECTURE AND OVERVIEW

Overview

Convolution Layers: Three convolution layers were established in the CNN definition for the main model of this project. One of the variants was decided to increase this layer size to 4. The included PyTorch functions are listed below along with a brief description of their purpose:

Conv2d - The objective is to perform a 2D convolution operation on an input signal that consists of many input planes, such as a batch of images. In order to extract features from the input image, the Conv2d layer is necessary. It helps in capturing spatial hierarchy in images by applying multiple filters (kernels) to the input image, which results in various elements of the image, like edges, textures and patterns.

BatchNorm2d - Applied on a 4D input (a mini batch of 2D inputs with extra channel dimensions) with the intention of applying batch normalisation. By dividing the result by the batch standard deviation and removing the batch mean, BatchNorrm2d normalizes the output of the convolutional layers. This decreases internal covariate shift, which fastens and stabilizes the training process. Also it helps in regularizing the model and lessen overfitting.

LeakyReLU - Uses the activation function of the Leaky Rectified Linear Unit. The model gains non-linearity from the LeakyReLU activation function, which enables it to learn intricate patterns. It enables a small gradient, while the unit is not active, which helps alleviate the “dying ReLU” problem, which can occasionally occur during training and lead neurons to exclusively output zeros. This is in contrast to conventional ReLU.

MaxPool2d - Applying a 2D max pooling procedure across an input signal made up of many input planes. By decreasing the input volume’s spatial dimensions (height and width), MaxPool2d helps lower the network’s parameter count and

computational load. Additionally, it aids in making feature recognition resistant to input translations. This layer takes the greatest value over a certain window size and effectively down samples the input.

Fully Connected Layers: Two FC layers were also added in the CNN definition. These provide a way for our model to apply the linear transformations to the input vectors. The forward() function of the CNN model simply calls the layers in succession*. The included PyTorch functions are listed below along with a brief description of their purpose:

Dropout - During training, ranodmly zeros a portion of the input tenso'r elements with probability p. A regularisation method called dropout is employed to stop overfitting. During training, a portion of the input units are randomly set to zero, which strengthens the model and keeps it from becoming to dependent on particular neurons. The network is encouraged to learn more widely distributed and broadly defined representations as a result.

Linear - Adds a linear transformation to the data that is received. The retrieved features from the convolutional layers are mapped to the final output classes using the linear layer (also called Fully Connected (FC) layer). By learning decision boundaries in the feature space, these layers handle the classification task and apply linear modifications to the input data.

LeakyReLU- The model gains non-linearity from the LeakyReLU activation function, which enables it to learn intricate patterns. It enables a small gradient, while the unit is not active, which helps alleviate the "dying ReLU" problem, which can occasionally occur during training and lead neurons to exclusively output zeros.

*image is flattened into a 2D array in between the convolution layers and the FC layers

Activation functions: The activation function, as described above, are used in both the convolution and FC layers of the CNN model class. Both layer types include:

LeakyReLU- The activation function allows the model to gain non-linearity from the LeakyReLU activation function, which enables it to learn intricate patterns. It

enables a small gradient, while the unit is not active, which helps alleviate the "dying ReLU" problem, which can occasionally occur during training and lead neurons to exclusively output zeros.

Regularization and Overfitting Solutions

Our team utilized imagenet's pretrained weights to help reduce overfitting. Additionally, we included two dropout layers in the fc layer of the model. As described in the previous section, these layers provide a means to remove unnecessary data from a previous layer.

OVERVIEW OF VARIANTS COMPARED TO MAIN CNN MODEL

Deviations

VARIANT 1: A new kernel size was implemented in each convolution layer. The size was increased from 3 to 4.

VARIANT 2: A fourth convolution layer was added to this variation. This increased the running time.

TRAINING PROCESS METHODOLOGY

Epochs

Attempting 10 proved to take a lot of time. 3 epochs was determined to be sufficient as the model achieved accuracies over 80% after a single epoch in many instances during the testing and training processes. It was additionally concluded by our team that, due to the time constraints, testing every model over ten epochs would take a significant amount of time. Our result ultimately concluded to be sufficient in their variation and accuracy.

Learning Rate

The learning rate for our model is set at 0.001. This rate was chosen in relation to our gradient loss calculations which can subsequently be used during backpropagation.

Loss Function

The loss function used was applied through PyTorch's neural network modules class. The CrossEntropyLoss function allows for users to calculate the cross entropy loss of input data and the expected outputs.

OPTIMIZATION TECHNIQUES

Adam Optimizer

Using the PyTorch library optim, the program has access to the Adam function. This function takes the model's parameters and the predetermined learning rate as arguments. This essentially ensures that our data is efficiently processed by minimizing the loss in our model through the manipulation of our model's parameters. The Adam algorithm aims to overall enhance performance by successfully applying new rules to learning rates and gradients.

EVALUATION: PERFORMANCE METRICS

Table 1: Precision and Recall Percentages of each Label in All Models

LABEL	MAIN MODEL		VARIANT1 MODEL		VARIANT2 MODEL	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Angry	77.05%	82.46%	70.39%	86.99%	85.00%	88.70%
Engaged	95.37%	93.64%	92.12%	89.74%	95.79%	81.25%
Happy	79.89%	84.48%	84.21%	73.68%	98.11%	59.77%
Neutral	75.58%	63.73%	68.32%	63.89%	54.75%	98.99%

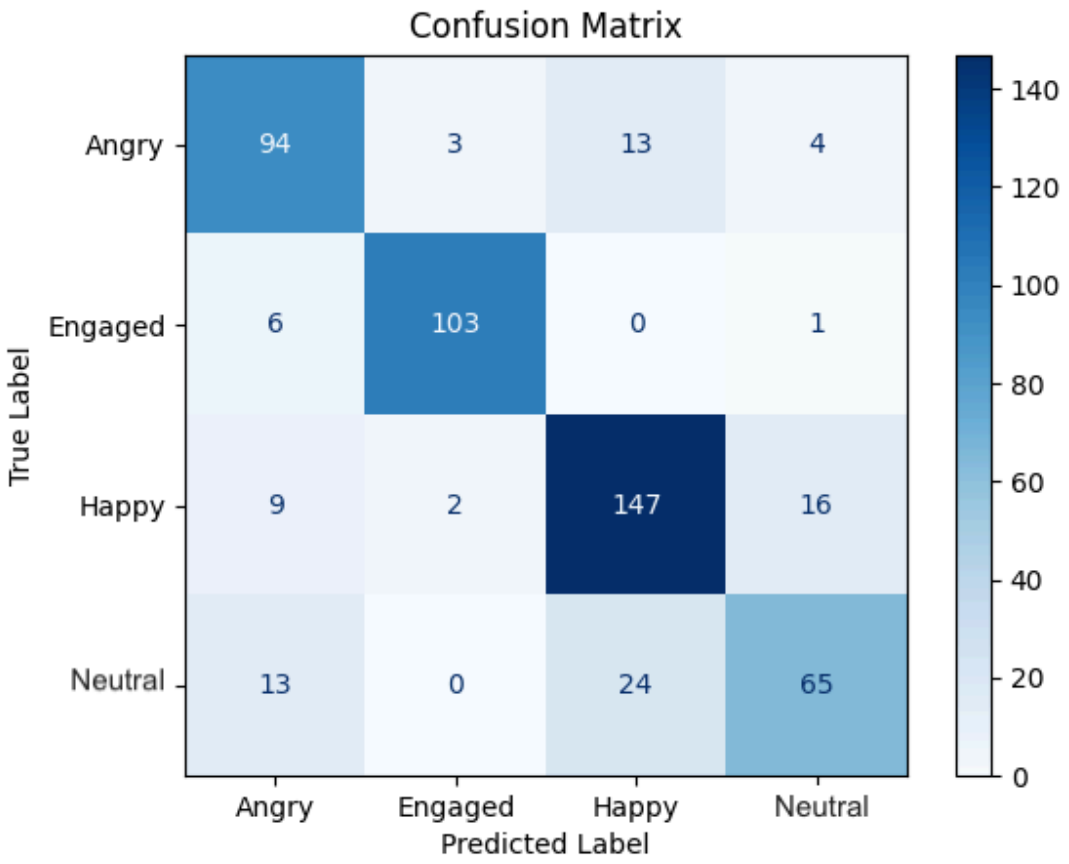
Table 2: Micro and Macro Precision, Recall, F1-Scores for All Models. All Models's Accuracy Score also included.

MODEL	MACRO			MICRO			ACCURACY
	P	**R**	**F1**	**P**	**R**	**F1**	
Main	81.97%	81.08%	81.52%	81.80%	81.80%	81.80%	81.80%
Variant 1	78.76%	78.57%	78.66%	78.60%	78.60%	78.60%	78.60%

Variant 2	83.41%	82.18%	82.79%		79.00%	79.00%	79.00%	79.00%

EVALUATION: CONFUSION MATRIX ANALYSIS

Main Model:

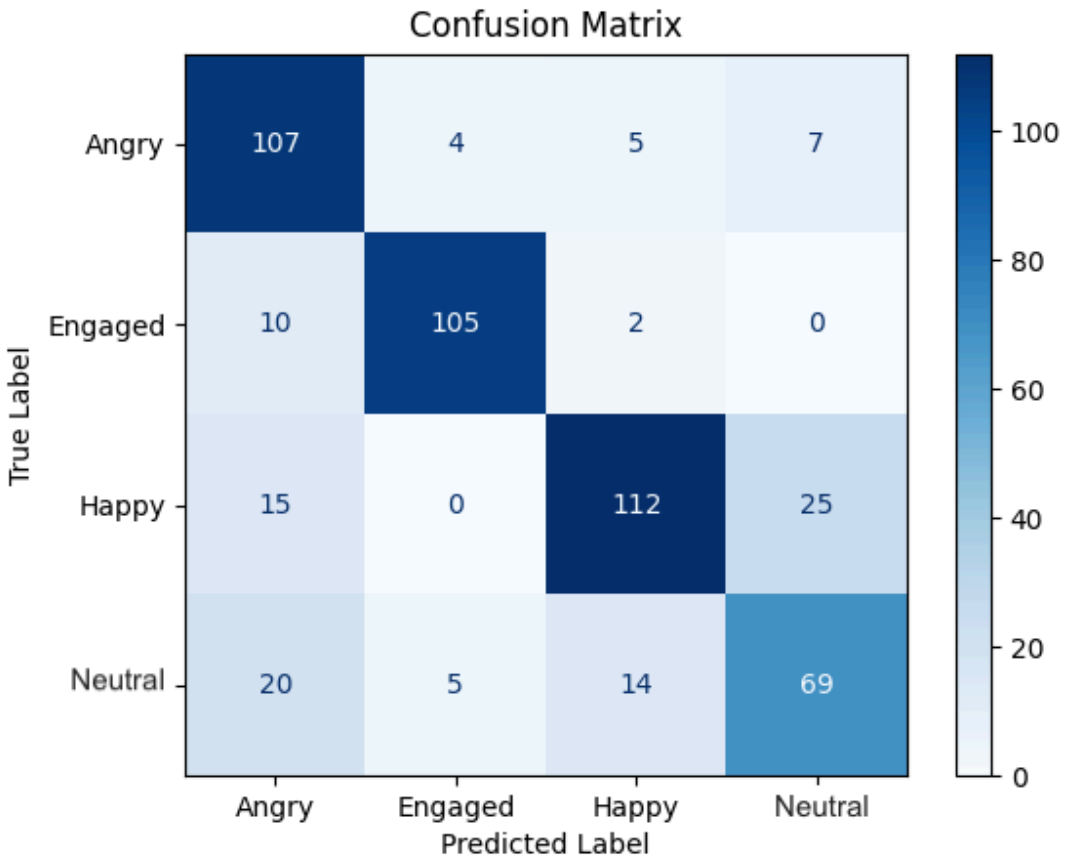


Terminal Output:

```
Saving Checkpoint
Epoch [1/3], Validation Loss: 0.7054, Validation Accuracy:
70.74%
...
Saving Checkpoint
Epoch [2/3], Validation Loss: 0.5398, Validation Accuracy:
78.56%
...
Saving Checkpoint
Epoch [3/3], Validation Loss: 0.3950, Validation Accuracy:
85.17%
```

```
...
Test Accuracy of the model on the test images: 81.8 %
```

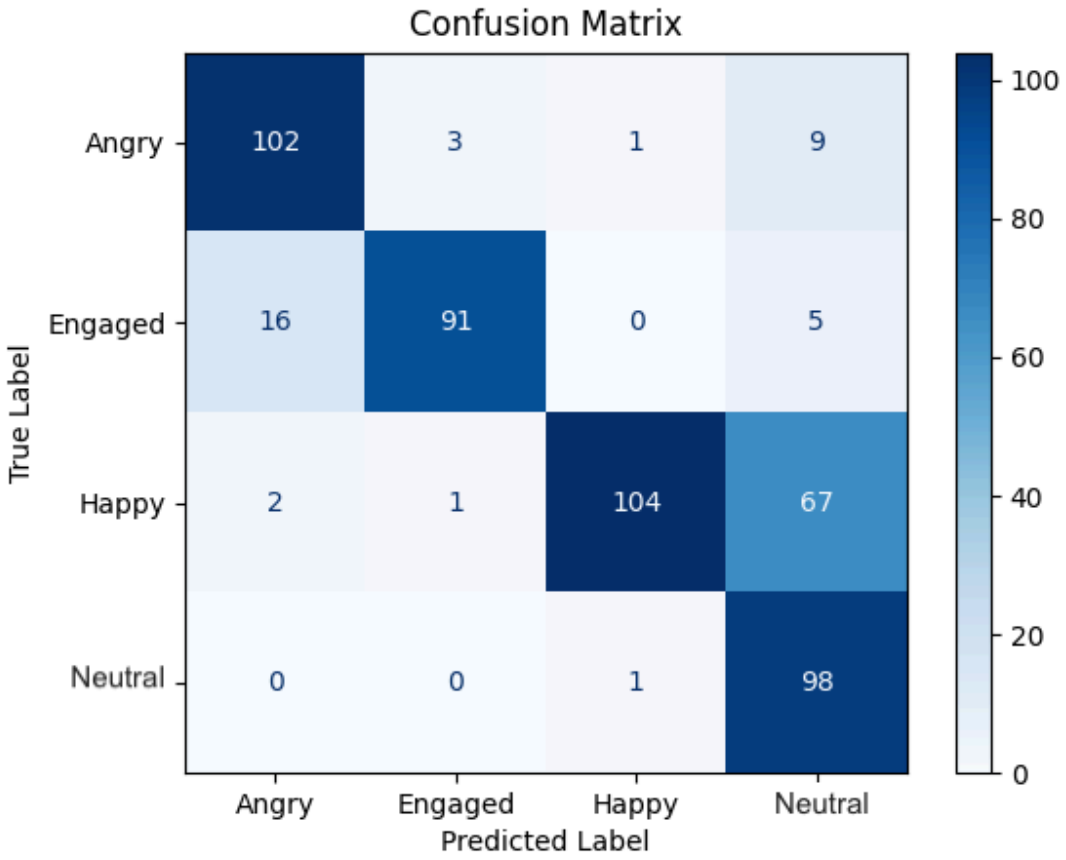
Variant 1 Model:



Terminal Output:

```
Saving Checkpoint
Epoch [1/3], Validation Loss: 1.2319, Validation Accuracy:
47.29%
...
Saving Checkpoint
Epoch [2/3], Validation Loss: 0.6682, Validation Accuracy:
77.56%
...
Saving Checkpoint
Epoch [3/3], Validation Loss: 0.5879, Validation Accuracy:
78.16%
...
Test Accuracy of the model on the test images: 78.60 %
```

Variant 2 Model:



Terminal Output:

```
Saving Checkpoint
Epoch [1/3], Validation Loss: 1.4974,Validation
Accuracy: 50.30%
...
Saving Checkpoint
Epoch [2/3], Validation Loss: 0.8376, Validation Accuracy:
75.35%
...
Saving Checkpoint
Epoch [3/3], Validation Loss: 0.6959, Validation Accuracy:
76.55%
...
Test Accuracy of the model on the test images: 79.0 %
```

Misclassification

Many of the images labeled ‘Happy’ and, on occasion, ‘Neutral’ are very similar to that of ‘Engaged’. This may be due to the datasets used. ‘Happy’ , ‘Angry’ and ‘Neutral’ were taken from the same dataset and ‘Engaged’ taken from a second that did not include any of the same labels as the first set. A brief look at the ‘Engaged’ dataset, we can see that there are many images that, even to a human, may be labeled under ‘Happy’ or ‘Neutral’ based on whether or not the person is smiling, thinking or curious.

Successes

Due in part to both the dataset and the model, the success of all four classes exists in more than one model tested and analyzed for this project.

VARIATIONS AND CONCLUSIONS

Depth

In the main CNN script (CNN_Main.py), there are a total of 4 convolutional layers utilized to ensure the proper analysis of the dataset to achieve a sufficient and satisfiable accuracy of ~70-85%. Increasing the convolutional layers would help the model capture more complex features, ergo improving performance on complex datasets and allowing the network to build upon the features detected by previous layers. On the other hand, the latter would also increase the risk of overfitting, most probably in the case of a small or undiverse dataset.

In the Variant 1 case (CNN_Variant1.py), the number of convolutional layers were increased to 5, theoretically leading to higher validation accuracy and potentially lower losses, whereas experimentally it actually led to slightly decreased validation accuracies of about 5-8% and a higher loss value, which would conclude that the resulting increase in the convolutional layers led to overfitting for the selected dataset. It has been therefore proven that the number of layers should be kept at 4 for the best results.

Kernel

In the main CNN script (CNN_Main.py), the kernel size is set to 3, resulting in a 3x3 analysis of the region surrounding every pixel in the data files. The aforementioned setting produced satisfiable accuracies of ~70-85% with minor loss values of 0.4-0.7. Increasing the kernel size would potentially increase the detection of small and intricate features such as edges, corners, textures, and minor variations in facial features. However, increasing the latter to a point beyond a 3x3 kernel format would also hinder the detection of subtle differences in facial expressions and minor facial features, in the case of facial recognition convolutional neural networks.

In the Variant 2 case (CNN_Variant1.py), the kernel size is set to 4, resulting in a 4x4 analysis of the region surrounding every pixel in the data files. Theoretically, the latter should increase the validation accuracy and decrease the loss value, where in fact, the opposite happened much like in Variant 1 with validation accuracies being lowered by ~5-10% and loss values increased by ~0.3 for every epoch analysis.

REFERENCES

Deng, W., Du, J., Li, S. Pattern Recognition and Intelligent System Laboratory, *RAF-DB*. (2017). Beijing University of Posts and Telecommunications. Accessed: May, 2024. [Online]. Available: https://www.kaggle.com/datasets/shuvoalok/raf-db-dataset

Deng, W. Pattern Recognition and Intelligent System Laboratory, *Real-world Affective Faces Database.* (2017). Beijing University of Posts and Telecommunications. Accessed: May, 2024. [Online]. Available: http://www.whdeng.cn/raf/model1.html

Kovenko, Volodymyr; Shevchuk, Vitalii. OAHEGA : EMOTION RECOGNITION DATASET. (2021). Mendeley Data, V2. Accessed: May 2024. [Online]. Available: https://www.kaggle.com/datasets/sujaykapadnis/emotion-recognition-dataset. DOI: 10.17632/5ck5zz6f2c.2

Patel, P. Kaggle.com, *Facial Expression Detection 2023*. (2023). Prajval Patel (datasets, Kaggle.com. Accessed: May 2024. [Online]. Available: https://www.kaggle.com/datasets/prajvalpatel/facial-expression-detection-2023

R. vaishnav, *Visualizing Feature Maps using PyTorch*, Medium, Jun. 28, 2021. https://ravivaishnav20.medium.com/visualizing-feature-maps-using-pytorch-12a48cd1e573

Pytorch, Conv2d — *PyTorch 1.10.1 documentation*, pytorch.org. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

PyTorch, BatchNorm2d — *PyTorch 1.9.0 documentation*, pytorch.org. https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

F. Franco, "*Batch Normalization with PyTorch*," Medium, Jan. 13, 2024. https://medium.com/@francescofranco_39234/batch-normalization-with-pytorch-20cdc205377f

PyTorch, LeakyReLU — *PyTorch 1.11.0 documentation*, pytorch.org. https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html

Pytorch, MaxPool2d — *PyTorch 1.8.1 documentation*, pytorch.org. https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html

GeeksforGeeks, *Apply a 2D Max Pooling in PyTorch*, GeeksforGeeks, Mar. 20, 2023. https://www.geeksforgeeks.org/apply-a-2d-max-pooling-in-pytorch/

PyTOrch, Dropout — *PyTorch 1.8.1 documentation*, pytorch.org. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

L. Shukla, *Implementing Dropout Regularization in PyTorch*, W&B, Jul. 02, 2020. https://wandb.ai/authors/ayusht/reports/Implementing-Dropout-Regularization-in-PyTorch--VmlldzoxNTgwOTE

PyTorch, *Linear- PyTorch 1.8.0 documentation*, pytorch.org. https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

Kanaries, *nn.Linear in PyTorch: Clearly Explained* docs.kanaries.net, Jun. 20, 2023. https://docs.kanaries.net/topics/Python/nn-linear