Summary (TL;DR)

- This project leverages Deep Learning to recognize and detect human emotions in images
- Human emotions are fundamental to our experience on our planet, as emotions affect everyone around us [1]
- One of the key inspirations for this project came from interacting with people; I realized that my interaction depends heavily on the emotions of the person I am interacting with
- A Convolutional Neural Network (CNN) was trained with multiple parameters to identify emotions in images of humans
- The following parametric tuning was then applied to improve accuracy which resulted in overall accuracy of 63%:
    - Increasing the number of Epochs
    - Decreasing batch sizes
    - Increasing the learning rate
- Future considerations for improving this Neural Network include:
    - Testing Gradient Descent with momentum as the optimizer
    - Testing different Loss functions
    - Updating the convolutional layers with varied parameters
- I am passionate about intelligent systems that can improve the life quality of people
- This project serves as a foundation for an ongoing project to build various computer vision and natural language processing functions that can then be used to create a personal assistant device such as a desktop personal assistant or personal robot
- This desktop assistant/personal robot is meant to be a consumer device to help improve the life quality of its customer

TABLE OF CONTENTS

Business Problem

- A key problem in the domain of personal assistant/personal robot is to build a deeper connect with it's customer in a way that it enriches the life of the customer
- Current technology solutions are limited to offering helpful services to their customers but no commercial grade device has emotional support features that are persistent
- There are however some social companion robots, however, they are focussed only on elder care [1]
- Emotion detection serves as a key component of the intelligence that needs to be persistent within the AI based system to create an emotional connection
- Humans innately use facial emotional expressions to infer emotions of others and this guides them to better interact with others
- Extensive research is being actively performed on the various areas of AI application towards emotion recognition [2]
- This project is a first step in the direction of defining algorithm that can focus on facial expressions to identify emotions

[1]: Carley Thornell. 6 AI innovations for those aging in place (Jul 2022). USA Today (link)
[2]: Anvita Saxena, Ashish Khanna, Deepak Gupta (2020). Emotion Recognition and Detection Methods: A Comprehensive Survey. Journal of Artificial Intelligence and Systems, 2, 53–79. (link)

Scope, Approach, Process, and Libraries

Scope:

- Define vision and approach
- Coding of the CNN
- Parametric optimization
- Report and presentation

Approach:

- Understanding existing research on image based emotion recognition and

detection
- Determining the appropriate libraries
- Coding framework
- Optimization approach including variable identification and prioritization

Process:
- Setting up the Linux Ubuntu system (GPU drivers), CUDA, and CUDNN
- Installing the key libraries (See below)
- Coding and related troubleshooting
- Identifying baseline performance
- Leveraging prioritized parameters to tune performance

Libraries
- Tensorflow: Tensorflow is an end-to-end platform to develop and deploy machine learning algorithms
- Keras: an open source library for training and deploying machine learning algorithms
- Numpy: library for arrays and mathematical functions
- Pandas: library for reading, cleansing and analyzing data
- Sklearn: library for data processing and visualization

Neural Network & Algorithm Overview

- Convolutional Neural Network (CNN) was chosen for this project as it is the preferred neural networks for small scale image classification [1]
- CNN progressively extracts higher level representations of image contents
- CNN do not require pre-processing and instead uses raw pixels to derive features and infer the object
- Rectified Linear Units (ReLU) was the activation function used to introduce non-linearity into the model
- Adam Optimizer was used for the CNN as it has proven to be better in terms of general performance and parameters requires less tuning
- Using the Convolutional Neural Network (CNN), facial expressions were predicted with an accuracy of 61.7 percent

[1]: Mingxing Tan and Zihang Dai. Toward Fast and Accurate Neural Networks for Image Recognition (Sept ‘21). Google Research Blog (link)
[2] Akash Ajagekar, Wiki Entry. (Fall ‘21). Cornell University Computational Optimization Open Textbook (link)

Dataset Overview

- The data used is from the Kaggle FER2013 collection which has approx 35,9K 48x48 grayscale photos

- The faces in the photos are centered and roughly occupy the same amount of space in each image
- Each of the photos from the dataset represents one of the following emotions :
    - Anger
    - Disgust
    - Fear
    - Happy
    - Neutral
    - Sadness
    - Surprise

CNN & Algorithm Performance Summary

- The baseline CNN using Adam optimizer performance is 56.9%
- The CNN using SGD optimizer performance is 33.5%
- Due to decreased SGD performance, the Adam Optimizer was selected and then further optimized using parametric tuning

- The Adam Optimizer CNN with highest achieved accuracy is 61,7% which is a 8.4% improvement compared to baseline of 56.9%
- The highest performing CNN has the following key parameters:
    - Batch Size = 16
    - Epoch = 60
    - Learning Rate = 0.002
- **Optimization process key insights**
    - Increasing Epochs from 30 to 75 resulted in accuracy improvements, however, 60 is the max Epoch at which loss rates do not improve
    - Decreasing batch sizes resulted in accuracy improvements; we began with 64 and determined 16 is optimal
    - Adam Learning Rate accuracy is optimal when increased from 0.0001 to 0.0002. However, increasing further results in degradation

Base Adam Model Performance Epochs:30, Batches :64, LR: 0.0001. Accuracy: 56.0%

	precision	recall	f1-score	support
0	0.469	0.463	0.466	501
1	0.000	0.000	0.000	45
2	0.336	0.075	0.123	477
3	0.754	0.872	0.809	921
4	0.405	0.582	0.477	617
5	0.791	0.586	0.673	406
6	0.510	0.601	0.552	622
accuracy			0.569	3589
macro avg	0.466	0.454	0.443	3589
weighted avg	0.551	0.569	0.543	3589

Optimization: Epochs Increased From 30-75 Accuracy: 60.7%

	precision	recall	f1-score	support
0	0.524	0.499	0.511	501
1	0.000	0.000	0.000	45
2	0.462	0.191	0.270	477
3	0.818	0.878	0.847	921
4	0.460	0.478	0.469	617
5	0.762	0.677	0.717	406
6	0.496	0.736	0.593	622
accuracy			0.607	3589
macro avg	0.503	0.494	0.487	3589
weighted avg	0.596	0.607	0.589	3589

Optimization: Batches Decreased From 64 To 16. Accuracy: 61.1%

	precision	recall	f1-score	support
0	0.543	0.491	0.516	501
1	0.667	0.044	0.083	45
2	0.471	0.241	0.319	477
3	0.817	0.848	0.832	921
4	0.501	0.464	0.481	617
5	0.634	0.798	0.707	406
6	0.517	0.707	0.597	622
accuracy			0.611	3589
macro avg	0.593	0.513	0.505	3589
weighted avg	0.604	0.611	0.595	3589

Optimization: LR Increased From 0.0001 to 0.0002. Accuracy: 61.7%

	precision	recall	f1-score	support
0	0.542	0.507	0.524	501
1	0.500	0.022	0.043	45
2	0.472	0.210	0.290	477
3	0.842	0.860	0.851	921
4	0.465	0.499	0.482	617
5	0.719	0.744	0.731	406
6	0.520	0.738	0.610	622
accuracy			0.617	3589
macro avg	0.580	0.511	0.504	3589
weighted avg	0.612	0.617	0.602	3589

Code Overview & Libraries

- Code has 859 lines in a single Python File and also an associated Jupyter notebook
- Code has five main sections:

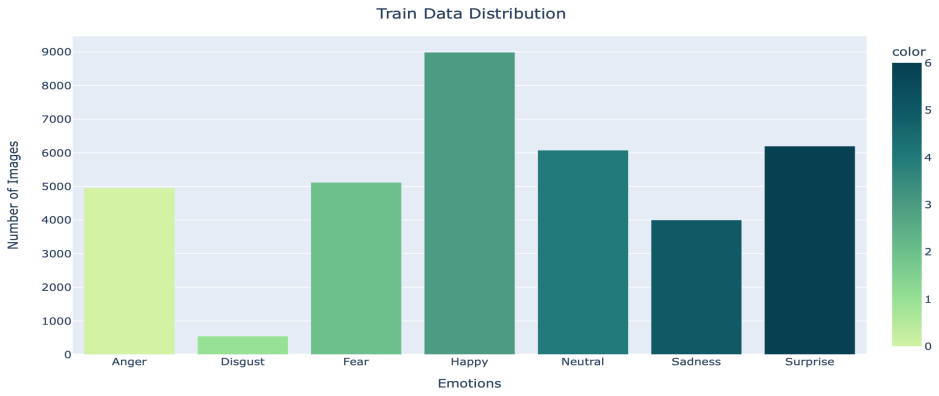
1. **Libraries**
2. **Data Processing & Setup**
3. **CNN Model Setup**
4. **Model Performance, Benchmarking, and Optimization**

Code Overview: 1. Libraries

- **Tensorflow:** Tensorflow is an end-to-end platform to develop and deploy machine learning algorithms
- **Keras:** an open source library for training and deploying machine learning algorithms
- **Numpy:** library for arrays and mathematical functions
- **Pandas:** library for reading, cleansing and analyzing data
- **Sklearn:** library for data processing and visualization

Code Overview: 2. Data Processing & Setup

- The data used is from the [Kaggle FER2013 collection] which has 35887 48x48 grayscale labeled emotion photos
- Below is the chart showing the data distribution by label:



- The data was split into approximately following subsets:
  - Training: ~80%
  - Test: ~10%
  - Validation: ~10%

```
(29068, 48, 48, 1)
(3589, 48, 48, 1)
(3230, 48, 48, 1)
```

Code Overview: 3. CNN Model Setup

- The Emotion Recognition and Detection CNN is composed of the following types of layers including their counts:

Layer Type	Overview	Count
Convolutional 2D layer	Performs convolution through scalar multiplication of inputs and weights	6
Batch Normalization	Layer to normalize data between layers using mini batches to speed up training	7
Max Pooling Layer	Takes the max values from batch input and summarizes into a feature map	5
Dropout	Masking layer that nullifies the contribution of some neurons towards the next layer to prevent overfitting	7

Dense	It is the network layer that takes input from all neurons in preceding layer and passing it as input to the next layer	3
Flatten	This layer takes the multiple pooled feature maps and and flattens it into a long vector	1

- Below is the parameter summary for the baseline model Convolutional Neural Network (CNN)

```
=================================
Total params: 5,810,183
Trainable params: 5,805,191
Non-trainable params: 4,992
_________________________________
```

Code Overview: 4. Model Performance, Benchmarking, and Optimization

- Code runs the CNN Adam model and computes the training and validation accuracy, loss, and confusion matrix
- We then build a Stochastic Gradient Descent Optimization model leveraging the same parameters as the CNN
- Code runs the SGD Model and computes the training and validation accuracy and loss
- We then apply parametric based changes to further optimize the Adam based CNN. Below is the list:
    - Increasing Epochs from 30 to 50
    - Increasing Epochs from 50 to 75
    - Decreasing Batch Size from 64 to 32
    - Decreasing Batch Size from 32 to 16
    - Decreasing Batch Size from 16 to 8
    - Increasing Adam Learning Rate from 0.0001 to 0.0002
    - Increasing Adam Learning Rate from 0.0001 to 0.0009

    - Increasing Adam Learning Rate from 0.0001 to 0.001
    - Increasing Adam Learning Rate from 0.0001 to 0.002

Key Challenges & Learnings For Future Exhibits:

Key Challenges:

- GPU based acceleration ran into several issues:
    - GPU identification by Ubuntu ran into issues due to driver incompatibilities and lack of clear documentation
    - Tensorflow GPU installation using both Nvidia and Tensorflow installations had gaps and resulted in failure to correctly install
    - Even after resolution of Tensorflow-GPU issues using Conda install, Tensorflow caused severe issues resulting in extremely low accuracy of the neural network (~20%)

- The workaround for Tensorflow-GPU issues was to choose the Non-GPU version of Tensorflow, however, due to significant other issues around Ubuntu/Tensorflow, the recommended approach was to use a new system
- Macbook M1 (ARM) was then used to replace the Ubuntu Desktop
- Despite the relatively new nature of Tensroflow on M1 (ARM), it performed well on the Mac but the CNN training time significantly increased

Key Learnings For Future :

- It's critical to not continue investing time into a systems problem that continues to worsen over time (cut losses)
- Always be willing to change an OS/system in sticky situations including using online compute (Colab et al)
- The Ubuntu system is being replaced as it has caused many issues and there is a high probability that these would continue despite clean installs

Future Enhancements & Optimizations

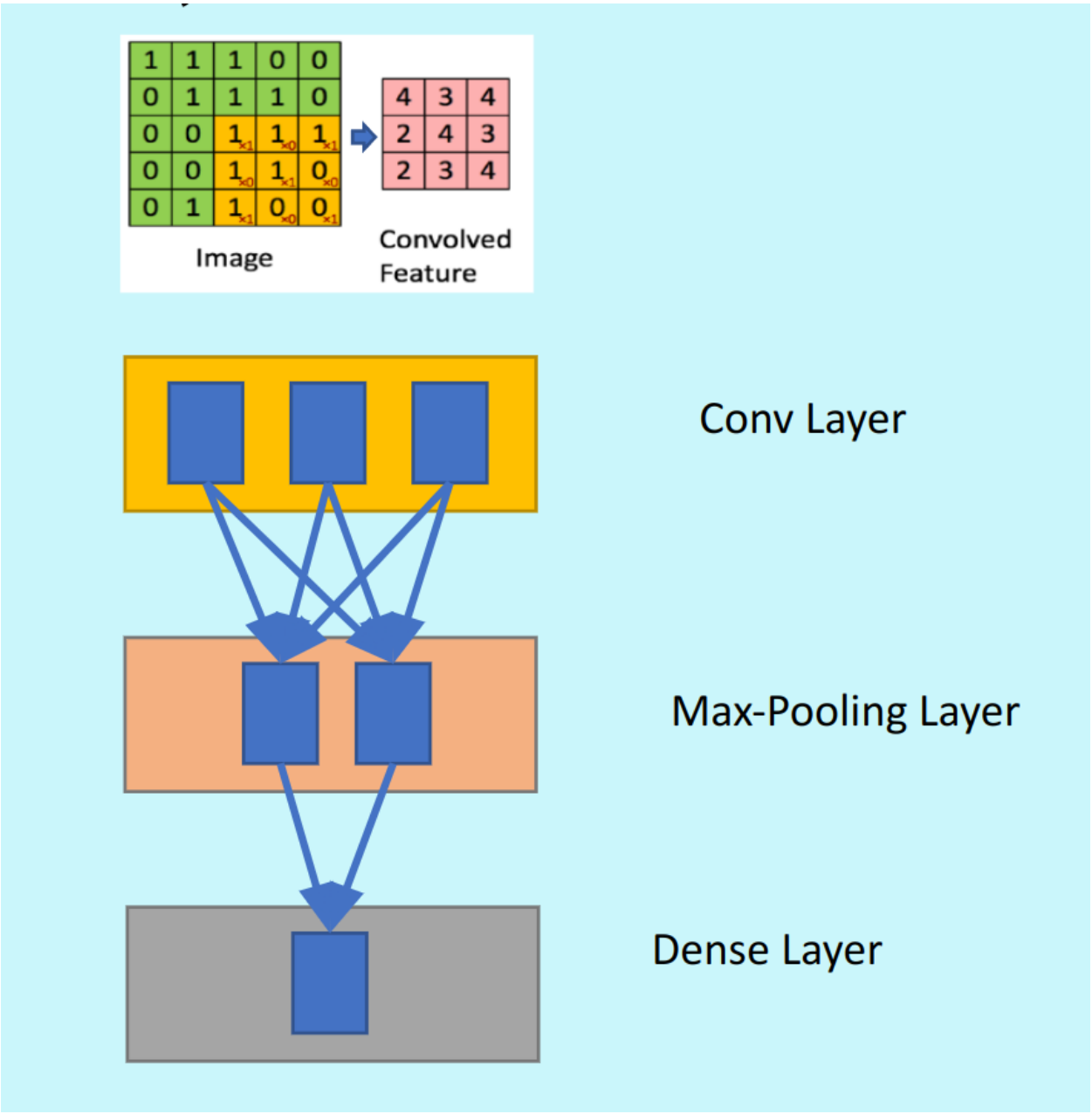
- Implement video camera frame capture to test and validate emotions
- Leverage Gradient Descent with Momentum for Optimization
- Experiment with other Loss Functions especially Dice Coefficient
- Update parameters for the Convolution layers
- Leverage other larger datasets for emotions

Project Resource Links

- Github repository homepage (Link)
- Jupyter Notebook (Link)
- Python code file (Link)
- Project Presentation (Link)

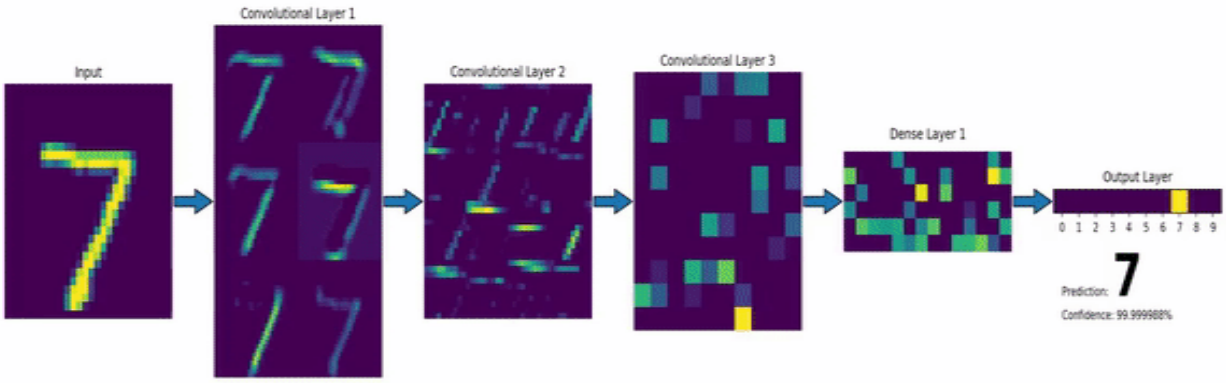
APPENDIX

CNN Visualized (1 of 3)

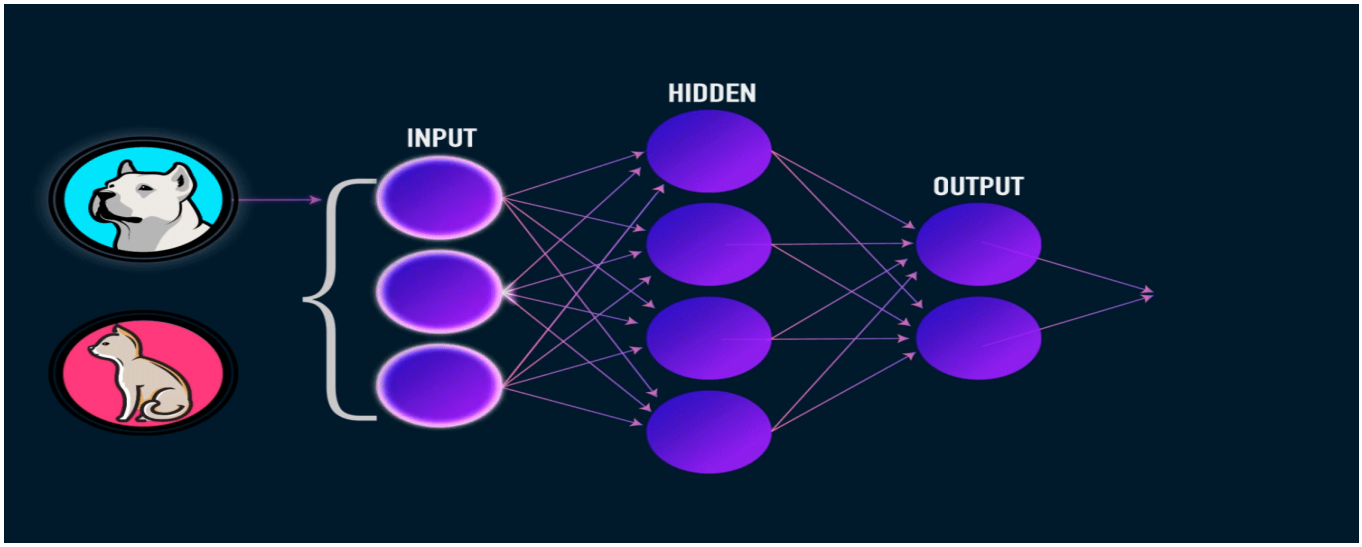


1]: Mingxing Tan and Zihang Dai. Toward Fast and Accurate Neural Networks for Image Recognition (Sept 2021). Google Research Blog (link)

CNN Visualized (2 of 3)



CNN Visualized (3 of 3)



EXHIBITS

Data Shape

Jupyter Notebook Step 2

```
(35887, 3)
```

Data Check

Jupyter Notebook Step 3

```
emotion    0
pixels     0
Usage      0
dtype: int64
```

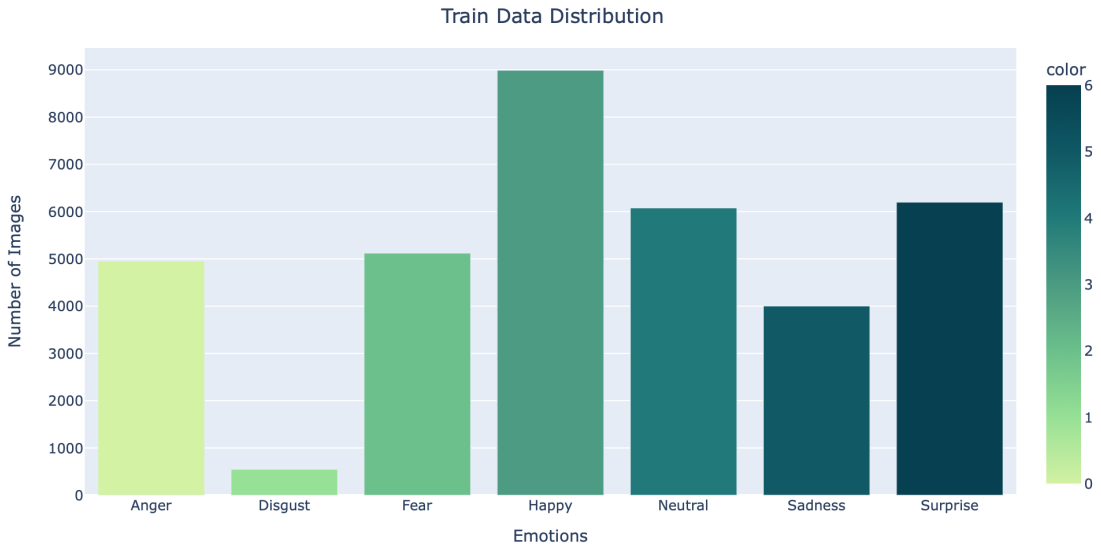
Data Head Validation

Jupyter Notebook Step 4

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

Data Distribution

Jupyter Notebook Step 5



Data Population Split

Jupyter Notebook Step 10

```
(29068, 48, 48, 1)
(3589, 48, 48, 1)
(3230, 48, 48, 1)
```

Data Sample Validation

Jupyter Notebook Step 11

Data Model Summary

Jupyter Notebook Step 13 (Part 1)

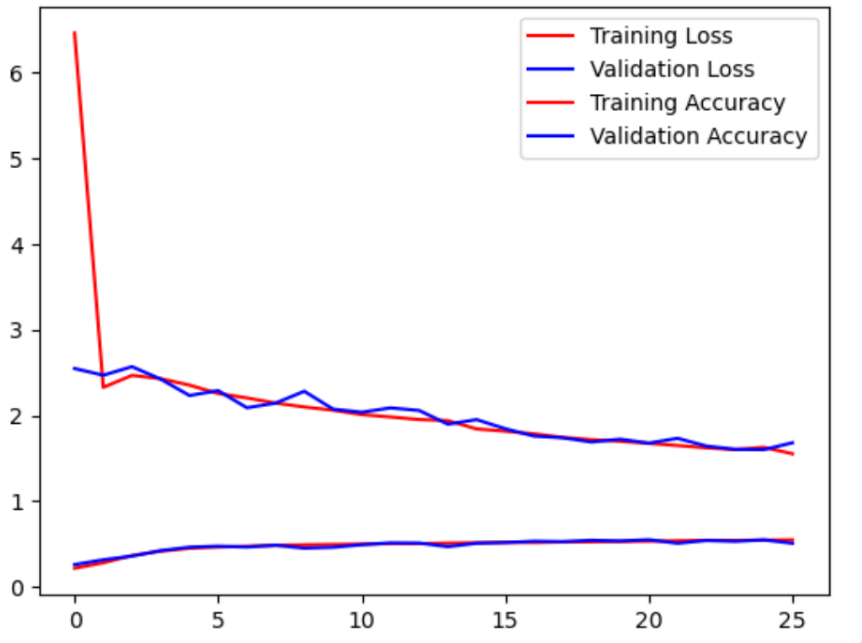
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	320
conv2d_1 (Conv2D)	(None, 48, 48, 64)	18496
batch_normalization (BatchNormalization)	(None, 48, 48, 64)	256
max_pooling2d (MaxPooling2D)	(None, 24, 24, 64)	0
dropout (Dropout)	(None, 24, 24, 64)	0
conv2d_2 (Conv2D)	(None, 24, 24, 128)	204928
batch_normalization_1 (BatchNormalization)	(None, 24, 24, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 128)	0
dropout_1 (Dropout)	(None, 12, 12, 128)	0
conv2d_3 (Conv2D)	(None, 12, 12, 512)	590336
batch_normalization_2 (BatchNormalization)	(None, 12, 12, 512)	2048
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 512)	0
dropout_2 (Dropout)	(None, 6, 6, 512)	0
conv2d_4 (Conv2D)	(None, 6, 6, 512)	2359808
batch_normalization_3 (BatchNormalization)	(None, 6, 6, 512)	2048
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 512)	0
dropout_3 (Dropout)	(None, 3, 3, 512)	0
conv2d_5 (Conv2D)	(None, 3, 3, 512)	2359808
batch_normalization_4 (BatchNormalization)	(None, 3, 3, 512)	2048
max_pooling2d_4 (MaxPooling2D)	(None, 1, 1, 512)	0
dropout_4 (Dropout)	(None, 1, 1, 512)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 256)	131328
batch_normalization_5 (BatchNormalization)	(None, 256)	1024
dropout_5 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 512)	131584
batch_normalization_6 (BatchNormalization)	(None, 512)	2048
dropout_6 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 7)	3591

Jupyter Notebook Step 13 (Part 2)

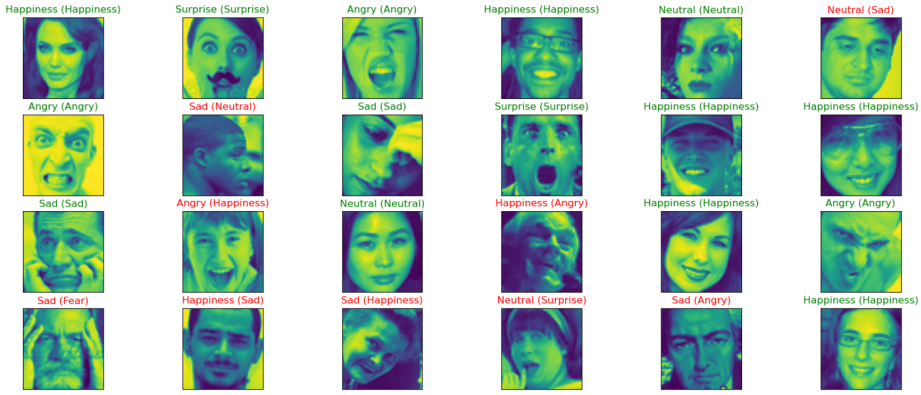
```
=================================
Total params: 5,810,183
Trainable params: 5,805,191
Non-trainable params: 4,992
_________________________________
```

Visualizing Model Results

Jupyter Notebook Step 15 (Part 1)

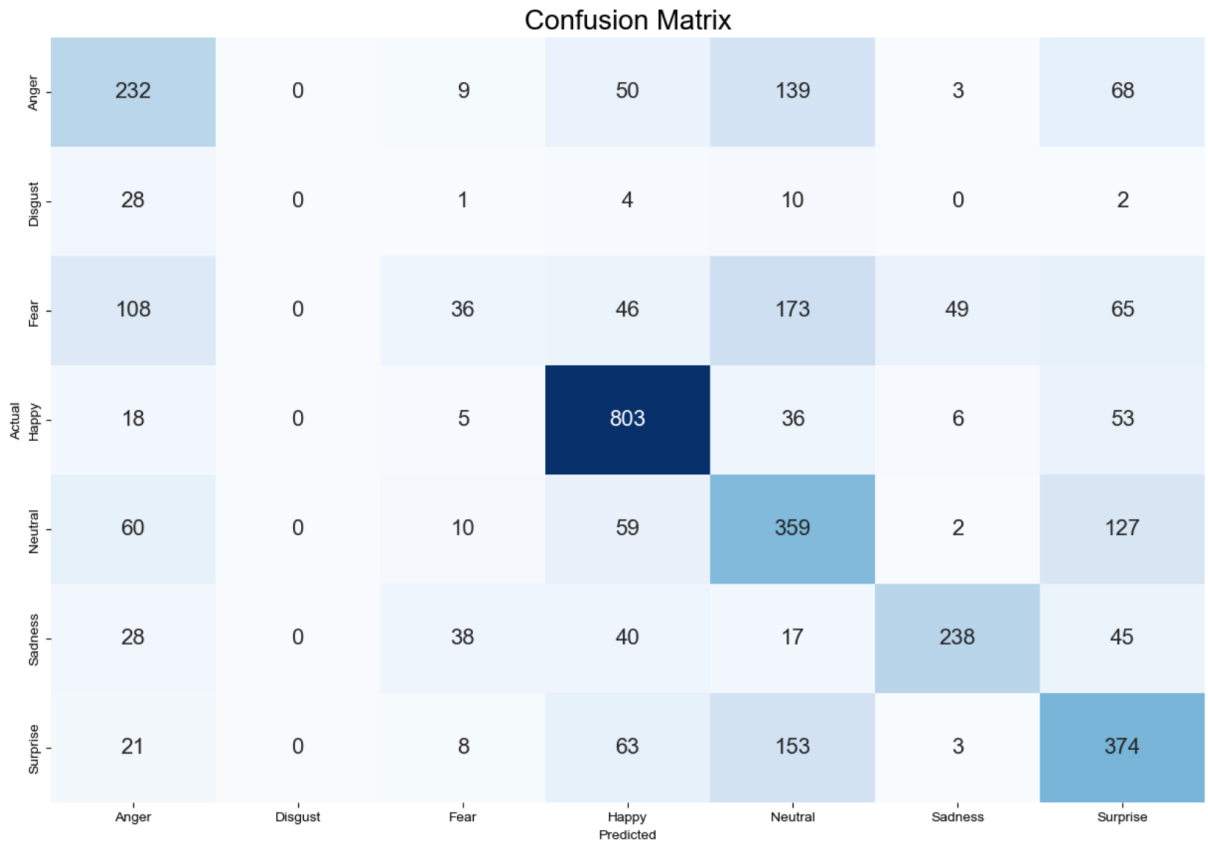


Jupyter Notebook Step 15 (Part 2)



Base Model (Adam): Confusion Matrix

Jupyter Notebook Step 16



Base Model (Adam): Performance Metrics

Jupyter Notebook Step 17

	precision	recall	f1-score	support
0	0.469	0.463	0.466	501
1	0.000	0.000	0.000	45
2	0.336	0.075	0.123	477
3	0.754	0.872	0.809	921
4	0.405	0.582	0.477	617
5	0.791	0.586	0.673	406
6	0.510	0.601	0.552	622
accuracy			0.569	3589
macro avg	0.466	0.454	0.443	3589
weighted avg	0.551	0.569	0.543	3589

Stochastic Gradient Descent (Sgd): Performance Comparison

Jupyter Notebook Step 18

```
Epoch 30: early stopping
113/113 [==============================] - 3s 23ms/step - loss: 9.4962 - accuracy: 0.3349
Test Acc: 0.3349122405052185
```

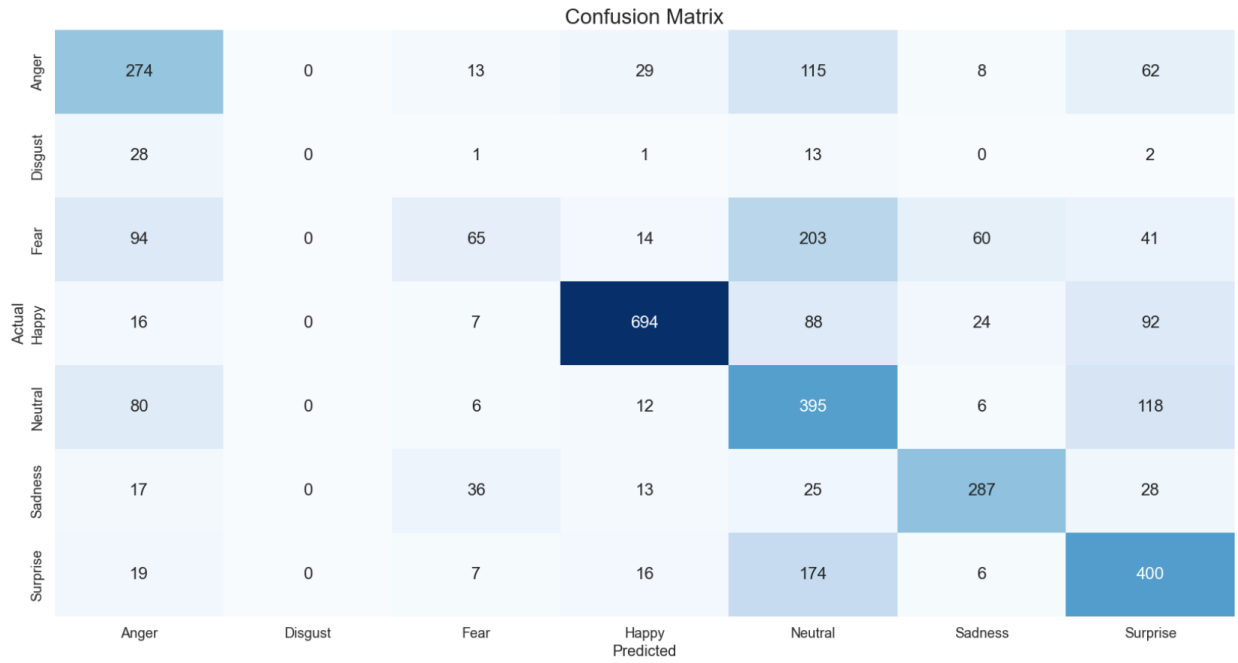
Optimization - Parametric Tuning Using Adam & Increasing Epochs From 30 To 50

Jupyter Notebook Step 21

Performance Metrics

	precision	recall	f1-score	support
0	0.519	0.547	0.533	501
1	0.000	0.000	0.000	45
2	0.481	0.136	0.212	477
3	0.891	0.754	0.816	921
4	0.390	0.640	0.485	617
5	0.734	0.707	0.720	406
6	0.538	0.643	0.586	622
accuracy			0.589	3589
macro avg	0.508	0.490	0.479	3589
weighted avg	0.608	0.589	0.578	3589

Confusion Matrix



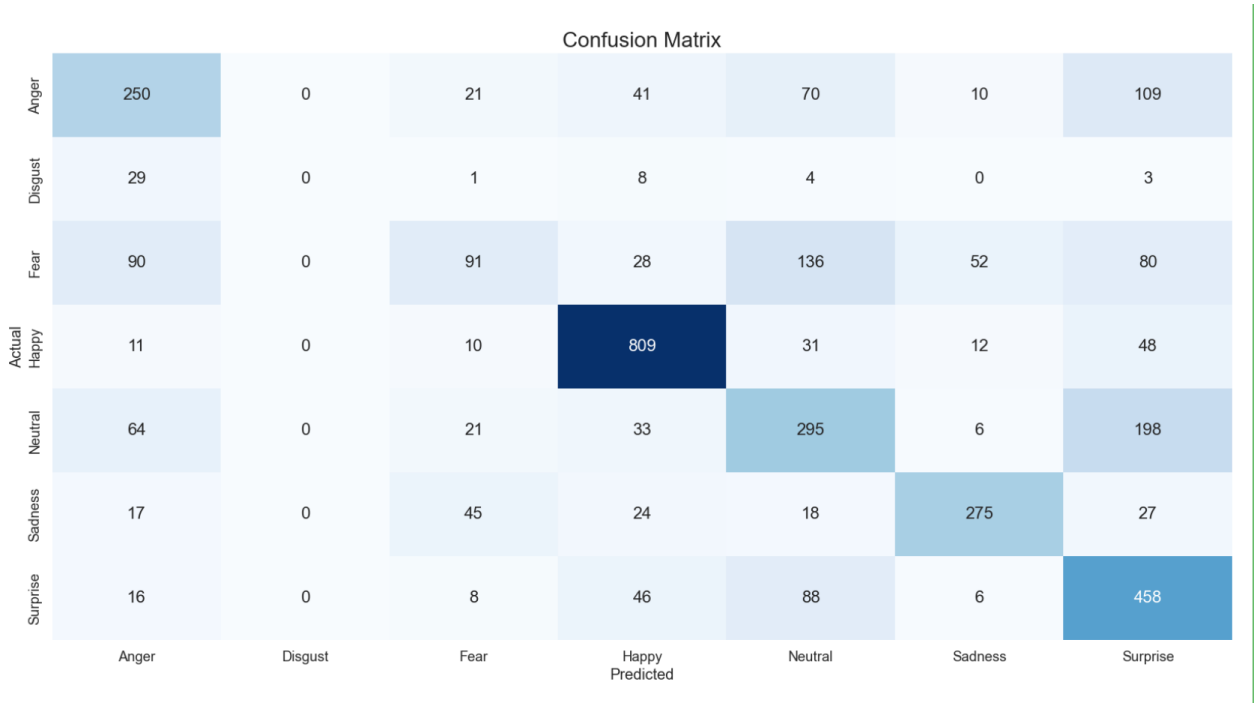
Optimization - Parametric Tuning Using Adam & Increasing Epochs From 30 To 75

Jupyter Notebook Step 23

Performance Metrics

	precision	recall	f1-score	support
0	0.524	0.499	0.511	501
1	0.000	0.000	0.000	45
2	0.462	0.191	0.270	477
3	0.818	0.878	0.847	921
4	0.460	0.478	0.469	617
5	0.762	0.677	0.717	406
6	0.496	0.736	0.593	622
accuracy			0.607	3589
macro avg	0.503	0.494	0.487	3589
weighted avg	0.596	0.607	0.589	3589

Confusion Matrix



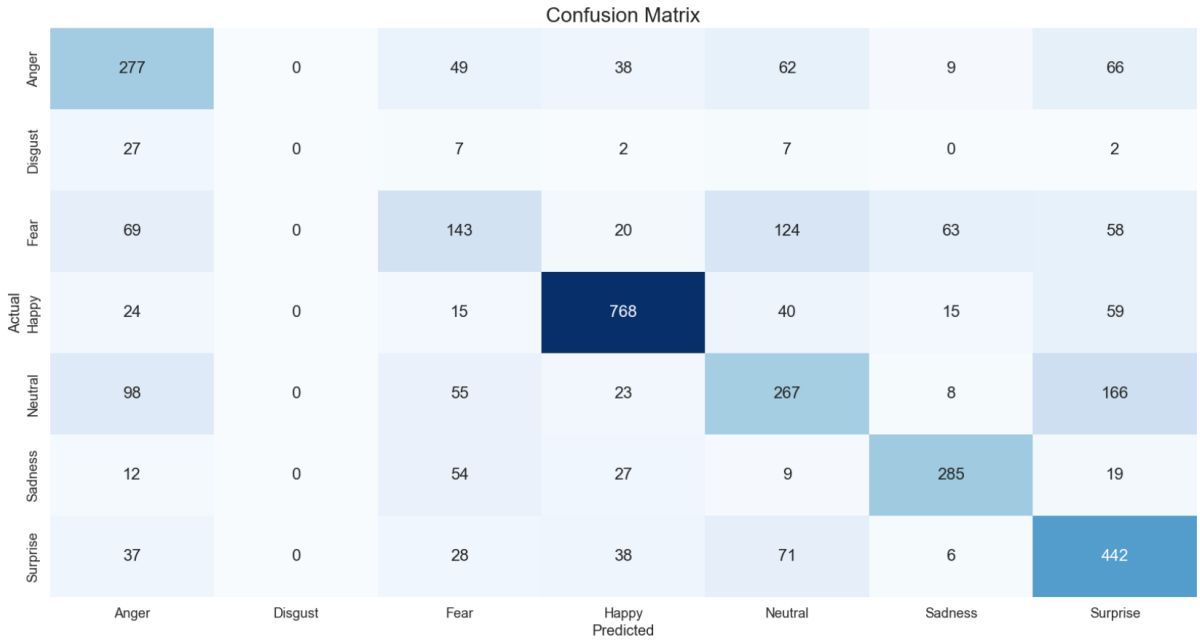
Optimization - Parametric Tuning Using Adam & Decreasing Batch Size from 64 to 32

Jupyter Notebook Step 25

Performance Metrics

	precision	recall	f1-score	support
0	0.509	0.553	0.530	501
1	0.000	0.000	0.000	45
2	0.407	0.300	0.345	477
3	0.838	0.834	0.836	921
4	0.460	0.433	0.446	617
5	0.738	0.702	0.720	406
6	0.544	0.711	0.616	622
accuracy			0.608	3589
macro avg	0.500	0.505	0.499	3589
weighted avg	0.597	0.608	0.599	3589

Confusion Matrix



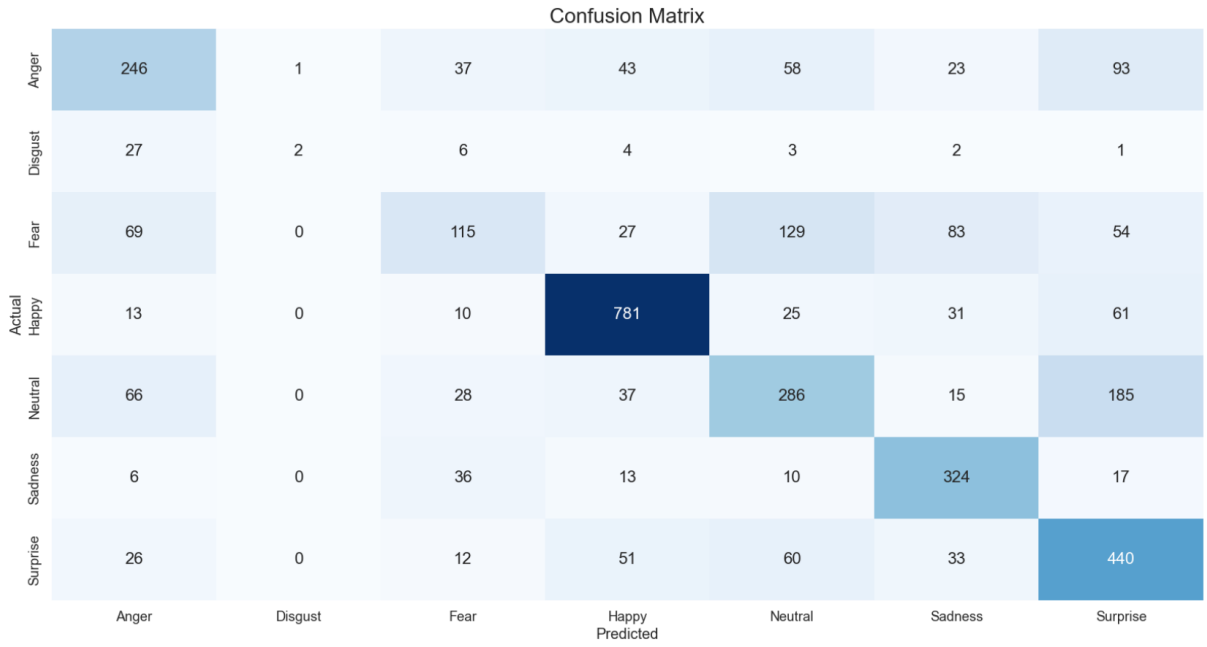
Optimization - Parametric Tuning Using Adam & Decreasing Batch Size from 32 to 16

Jupyter Notebook Step 27

Performance Metrics

	precision	recall	f1-score	support
0	0.543	0.491	0.516	501
1	0.667	0.044	0.083	45
2	0.471	0.241	0.319	477
3	0.817	0.848	0.832	921
4	0.501	0.464	0.481	617
5	0.634	0.798	0.707	406
6	0.517	0.707	0.597	622
accuracy			0.611	3589
macro avg	0.593	0.513	0.505	3589
weighted avg	0.604	0.611	0.595	3589

Confusion Matrix



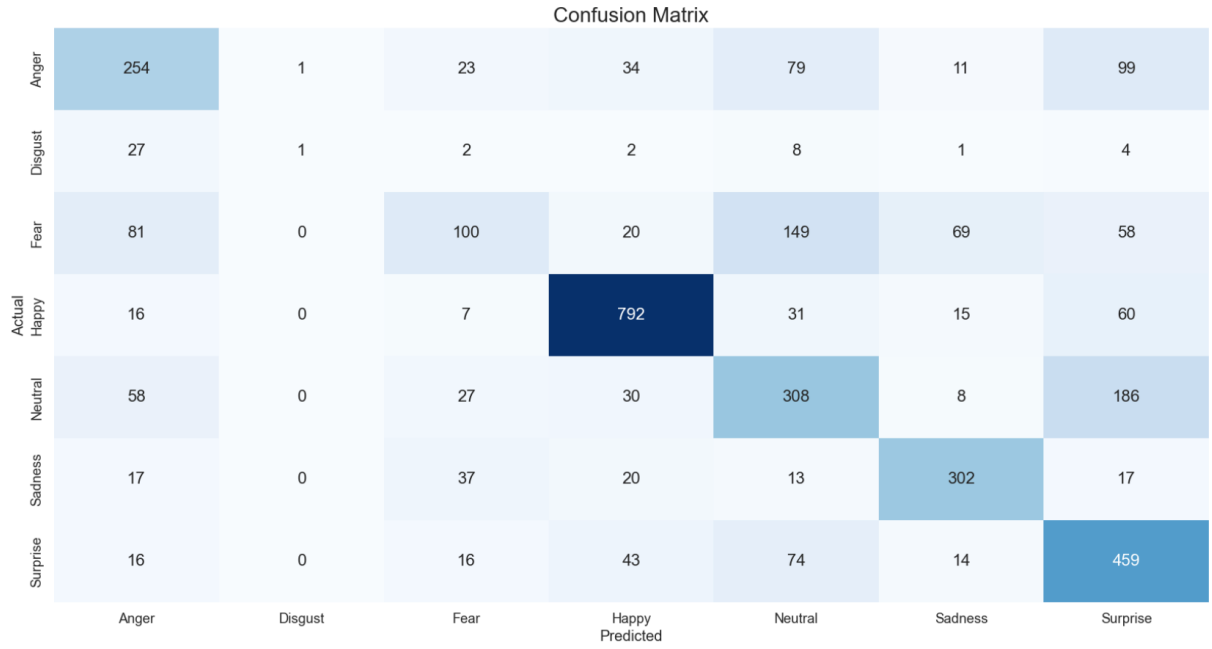
Optimization - Parametric Tuning Using Adam & Decreasing Batch Size from 16 to 8

Jupyter Notebook Step 29

Performance Metrics

	precision	recall	f1-score	support
0	0.542	0.507	0.524	501
1	0.500	0.022	0.043	45
2	0.472	0.210	0.290	477
3	0.842	0.860	0.851	921
4	0.465	0.499	0.482	617
5	0.719	0.744	0.731	406
6	0.520	0.738	0.610	622
accuracy			0.617	3589
macro avg	0.580	0.511	0.504	3589
weighted avg	0.612	0.617	0.602	3589

Confusion Matrix



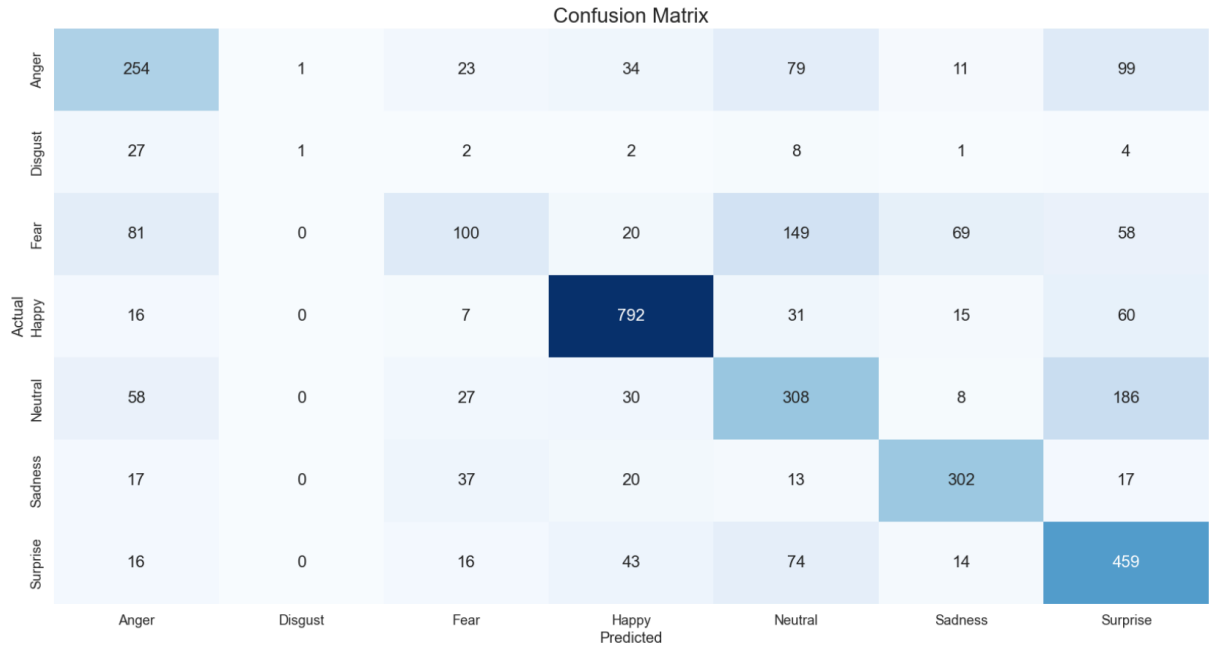
Optimization - Parametric Tuning Using Adam & Increasing Learning Rate (LR) from 0.0001 to 0.0002

Jupyter Notebook Step 30

Performance Metrics

	precision	recall	f1-score	support
0	0.542	0.507	0.524	501
1	0.500	0.022	0.043	45
2	0.472	0.210	0.290	477
3	0.842	0.860	0.851	921
4	0.465	0.499	0.482	617
5	0.719	0.744	0.731	406
6	0.520	0.738	0.610	622
accuracy			0.617	3589
macro avg	0.580	0.511	0.504	3589
weighted avg	0.612	0.617	0.602	3589

Confusion Matrix



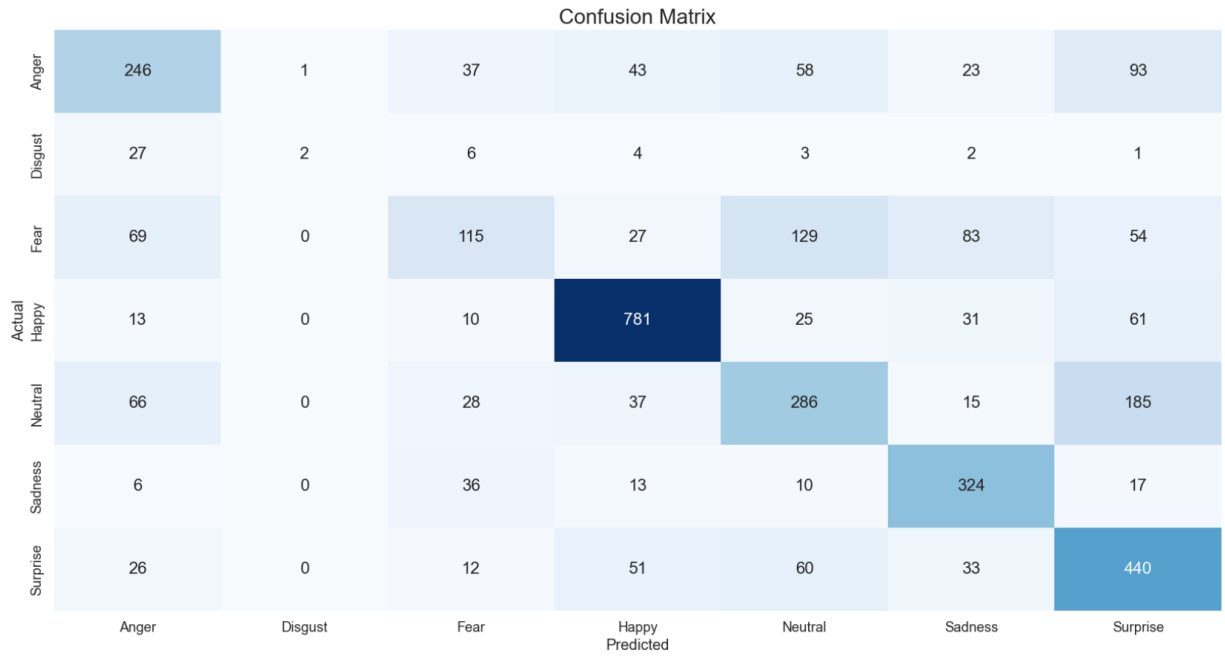
Optimization - Parametric Tuning Using Adam & Increasing Learning Rate (LR) from 0.0001 to 0.0009

Jupyter Notebook Step 33

Performance Metrics

	precision	recall	f1-score	support
0	0.543	0.491	0.516	501
1	0.667	0.044	0.083	45
2	0.471	0.241	0.319	477
3	0.817	0.848	0.832	921
4	0.501	0.464	0.481	617
5	0.634	0.798	0.707	406
6	0.517	0.707	0.597	622
accuracy			0.611	3589
macro avg	0.593	0.513	0.505	3589
weighted avg	0.604	0.611	0.595	3589

Confusion Matrix



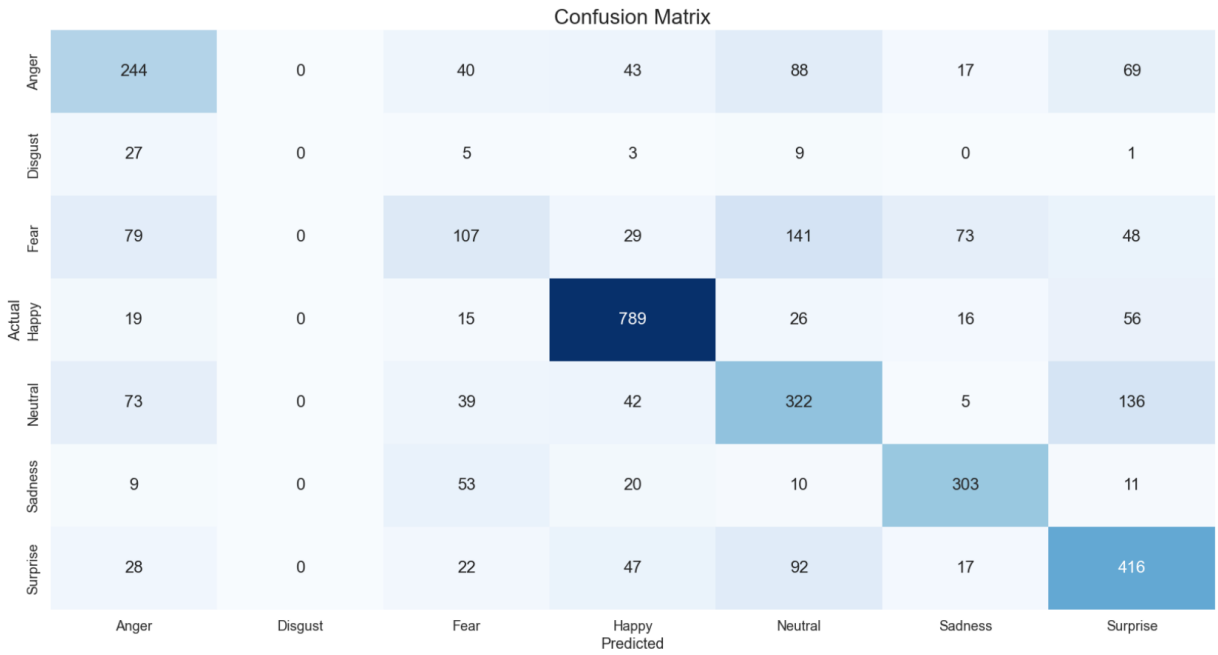
Optimization - Parametric Tuning Using Adam & Increasing Learning Rate (LR) from 0.0001 to 0.001

Jupyter Notebook Step 36

Performance Metrics

	precision	recall	f1-score	support
0	0.509	0.487	0.498	501
1	0.000	0.000	0.000	45
2	0.381	0.224	0.282	477
3	0.811	0.857	0.833	921
4	0.468	0.522	0.493	617
5	0.703	0.746	0.724	406
6	0.564	0.669	0.612	622
accuracy			0.608	3589
macro avg	0.491	0.501	0.492	3589
weighted avg	0.588	0.608	0.594	3589

Confusion Matrix



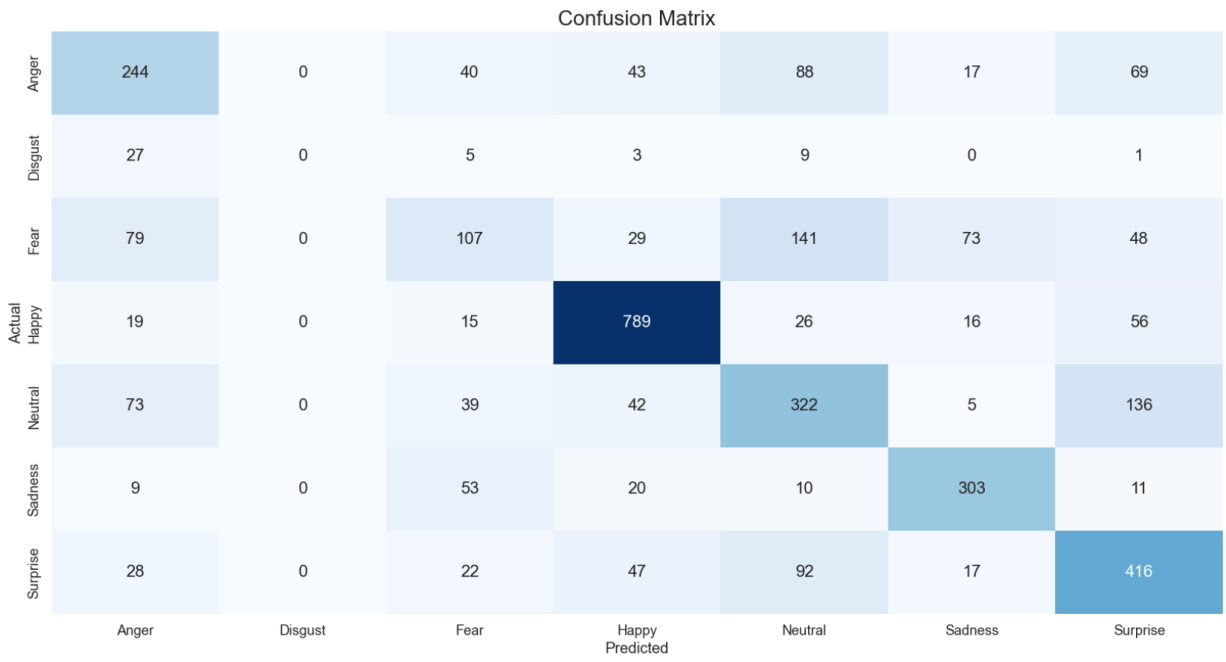
Optimization - Parametric Tuning Using Adam & Increasing Learning Rate (LR) from 0.0001 to 0.002

Jupyter Notebook Step 38

Performance Metrics

	precision	recall	f1-score	support
0	0.509	0.487	0.498	501
1	0.000	0.000	0.000	45
2	0.381	0.224	0.282	477
3	0.811	0.857	0.833	921
4	0.468	0.522	0.493	617
5	0.703	0.746	0.724	406
6	0.564	0.669	0.612	622
accuracy			0.608	3589
macro avg	0.491	0.501	0.492	3589
weighted avg	0.588	0.608	0.594	3589

Confusion Matrix



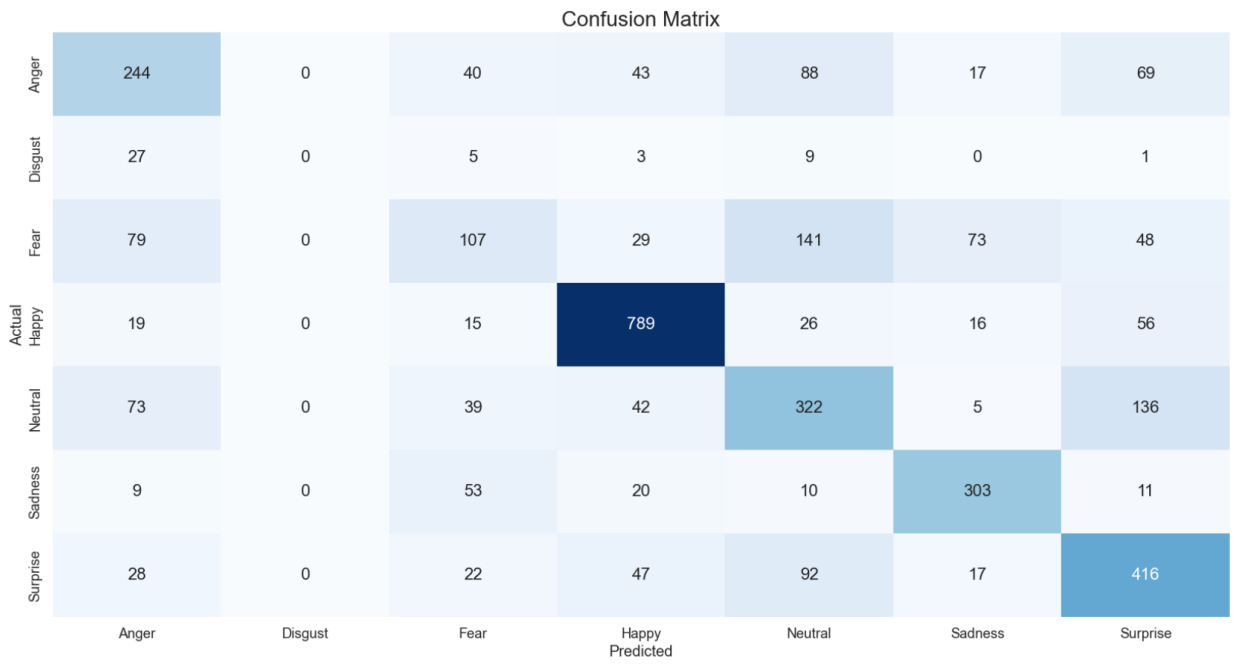
Optimization - Parametric Tuning Using Adam & Increasing Learning Rate (LR) from 0.0001 to 0.0025

Jupyter Notebook Step 40

Performance Metrics

	precision	recall	f1-score	support
0	0.509	0.487	0.498	501
1	0.000	0.000	0.000	45
2	0.381	0.224	0.282	477
3	0.811	0.857	0.833	921
4	0.468	0.522	0.493	617
5	0.703	0.746	0.724	406
6	0.564	0.669	0.612	622
accuracy			0.608	3589
macro avg	0.491	0.501	0.492	3589
weighted avg	0.588	0.608	0.594	3589

Confusion Matrix



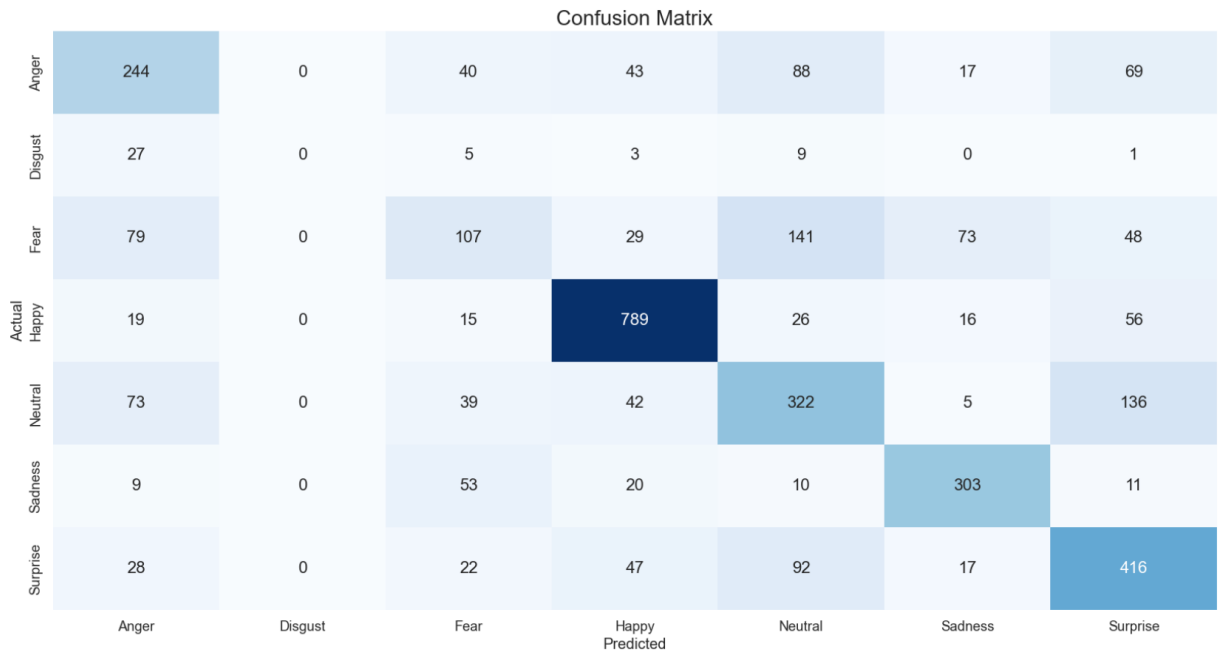
Optimization - Parametric Tuning Using Adam & Increasing Learning Rate (LR) from 0.0001 to 0.0021

Jupyter Notebook Step

Performance Metrics

	precision	recall	f1-score	support
0	0.509	0.487	0.498	501
1	0.000	0.000	0.000	45
2	0.381	0.224	0.282	477
3	0.811	0.857	0.833	921
4	0.468	0.522	0.493	617
5	0.703	0.746	0.724	406
6	0.564	0.669	0.612	622
accuracy			0.608	3589
macro avg	0.491	0.501	0.492	3589
weighted avg	0.588	0.608	0.594	3589

Confusion Matrix


Confusion Matrix
Actual
Anger
Disgust
Fear
Happy
Neutral
Sadness
Surprise
244 0 40 43 88 17 69
27 0 5 3 9 0 1
79 0 107 29 141 73 48
19 0 15 789 26 16 56
73 0 39 42 322 5 136
9 0 53 20 10 303 11
28 0 22 47 92 17 416
Anger
Disgust
Fear
Happy
Neutral
Sadness
Surprise
Predicted