COMP472: Artificial Intelligence
Concordia University, Montreal, Quebec

# Project “Education MindA.I.lytics”

Github project repository link:
https://github.com/loaidieu/CNN-Emotion-Dectector

April 21th, 2024

We certify that this submission is the original work of members of the group and the Faculty’s Expectations of Originality.

☑ Anh Quan Truong **(Data, Training Specialist)** - 40186528
☑ Kevin Theam **(Evaluation Specialist)** - 40192205
☑ Tat Loai Dieu Lieu **(Data Specialist)** - 40225452

## Abstract

The dataset used in this project encompasses a diverse range of facial images depicting various emotions such as happiness, neutrality, surprise, and engagement. With a total of 35,887 images across different categories, including training and testing sets, the dataset offers broad demographic representation and artistic variations. However, challenges arise from side-view complexities, watermarked images, and artistic renderings, potentially influencing the accuracy of emotion recognition models. Data selection involves handpicking categories and creating a focused subset to ensure balanced representation of emotions. Subsequently, each emotion folder (i.e. “focused”, “happy”, “neutral”, “surprised”) is populated with 400 images, selected from the original data. Image conversion into NumPy arrays facilitates machine learning tasks, while data augmentation techniques like rotation enhance dataset diversity. Along with considerations for image resizing and pixel intensity distribution analysis, normalization is applied for efficient model training. Ambiguities in labeling and data preprocessing underscore the need for meticulous data handling and validation. Overall, the dataset provides valuable insights into emotion recognition model development but requires careful consideration of various challenges and preprocessing steps to ensure model robustness and accuracy.

# 1. Dataset

## 1.1. Overview

The dataset “Emotion Detection”[1] encompasses a variety of images reflecting diverse emotions, including happiness and neutrality. Primarily, the images showcase frontal shots of faces set against varied backgrounds. An interesting aspect of this dataset is its broad demographic representation, spanning a wide range of ages and encompassing distinct races and genders. Additionally, the dataset exhibits specific characteristics, such as side-view images (where the complete facial structure may not be visible), artistic renderings (e.g., drawings, sketches), or instances where individuals in the images wear sunglasses.

| Type of dataset | Emotion Categories | Number of images | Total images/dataset | Total images |
|---|---|---|---|---|
| Training | angry | 3995 | 28709 | 35887 |
| | disgusted | 436 | | |
| | fearful | 4097 | | |
| | happy | 7215 | | |
| | neutral | 4965 | | |
| | sad | 4830 | | |
| | surprised | 3171 | | |
| Testing | angry | 958 | | |
| | disgusted | 111 | | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | fearful | 1024 | 7178 |  |
|  | happy | 1774 |  |  |
|  | neutral | 1233 |  |  |
|  | sad | 1247 |  |  |
|  | surprised | 831 |  |  |

*Figure 1.1. Overview of the dataset*

**1.2. Dataset Justification**

The motivation for selecting these datasets for the project lies in their unique characteristics and direct relevance to the objectives of facial emotion recognition. The datasets were specifically chosen for the following reasons:

- *Emotional Diversity*: The datasets comprehensively cover a spectrum of emotions, prominently featuring happiness, neutrality, surprise, sadness and even anxiety. This aligns seamlessly with the project's overarching goal of crafting a robust emotion recognition system.
- *Demographic Representation*: The inclusion of diverse age groups, races, and genders within the datasets contributes to the creation of a comprehensive dataset. This diversity enhances the model's capacity to generalize effectively across a wide range of demographic categories.

However, these datasets present specific disadvantages:

- *Side-View Complexity*: The inclusion of side-view images, where facial structures may be partially obscured, introduces complexity to the recognition task. This challenge requires the model to navigate instances where the facial expression might be misinterpreted as anger, despite the person actually expressing a neutral emotion.
- *Watermarked Images*: A further challenge is presented by certain images within the dataset bearing watermarks.
- *Artistic Variations*: The dataset's incorporation of artistic renderings and instances featuring individuals wearing sunglasses adds variations that necessitate the model's adept handling for accurate recognition.

In addition, there are several challenges for the AI model to learn:

- *Subtle Facial Expressions*: Teaching the model to accurately recognize subtle changes in facial expressions, especially those indicative of nuanced emotions like mild surprise or slight discomfort.
- *Diverse Demographics*: Ensuring the model can generalize well across diverse demographic groups, including different age ranges, ethnicities, and genders, to avoid bias and improve inclusivity.
- *Dynamic Environments*: Adapting to variations in lighting conditions, background settings, and dynamic environments to maintain accurate recognition in real-world scenarios.
- *Subject-Specific Challenges*: Handling cases where individuals have unique facial features or characteristics, such as beards, glasses, or distinctive hairstyles, which might impact facial expression analysis.
- *Ambiguous Expressions*: Addressing situations where facial expressions are ambiguous or mixed, making it challenging to categorize them into a single emotion category accurately.

These challenges contribute to the model's overall learning experience, fostering adaptability and accuracy in recognizing facial emotions across a wide array of situations and individuals.

### 1.3. Provenance Information:

- *Source of Images:* https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer?rvi=1
- *Collection Methodologies:* Download
- *License:* CC0: Public Domain (CC0 1.0 DEED – CC0 1.0 Universal)

## 2. Data Cleaning

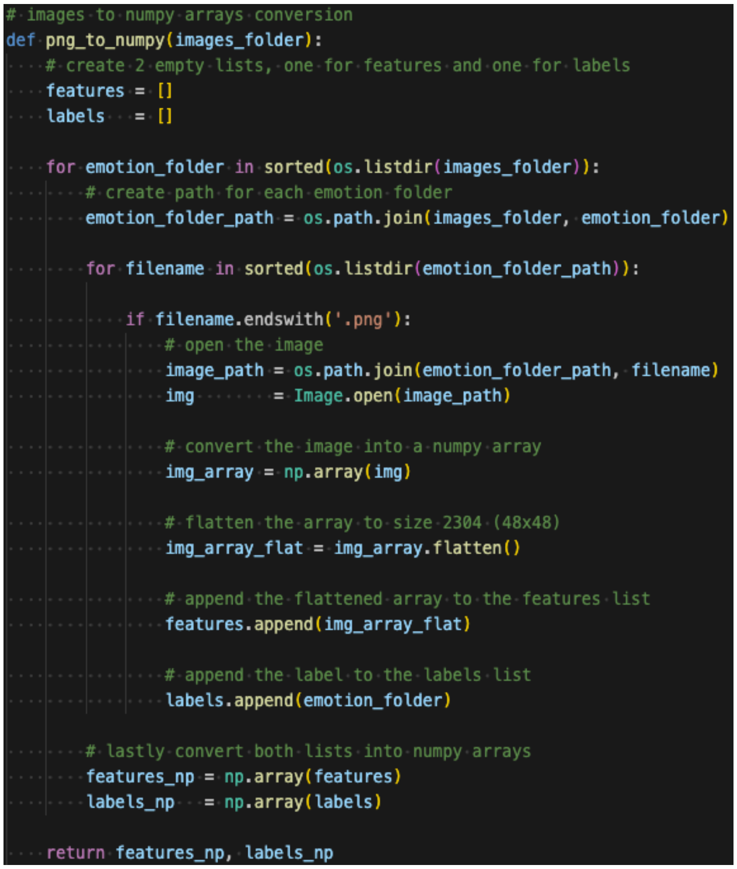
### 2.1. Data Selection

Three folders of images are hand-selected from the original dataset, which are “happy”, “neutral” and “surprised”. Subsequently, from the “neutral” folder, images that appear to depict individuals who seem focused or intent on something, are extracted and labeled “focused”, making the 4th emotion that is used in this project. Overall, each folder has 400 images. A limit is imposed on the number of images per folder to ensure that all four emotions are equally represented.

It is believed that having “focused/engaged” is essential for creating a more representative dataset since being focused is one of the more common emotions. These focused expressions capture moments of concentration, determination, and engagement that are somewhat distinct from neutral expressions. Including such images enriches the dataset with emotional variety, enhancing [1] its relevance and applicability across various domains. Moreover, training machine learning models on a dataset with diverse emotional expressions, including focused ones, improves their robustness and performance in real-world scenarios. With that said, differentiating between focused and neutral expressions can be challenging due to several factors. First, both emotions may exhibit similar facial features making it difficult to discern subtle differences. Additionally, individual variations in facial expressions in emotional states can further complicate this fact. Furthermore, contextual factors, such as the absence of accompanying body language or environmental cues, may hinder accurate interpretation of the emotion conveyed. Finally, subjective interpretation and inherent biases of human annotators can also influence the labeling process, leading to inconsistencies in emotion classification. As a result, contrary to popular belief, having both "focused/engaged" and "neutral" images in a dataset might potentially impact ML models trained for emotion recognition tasks.

### 2.1. Image Conversion

Converting images into NumPy arrays is essential for machine learning (ML), which often require data to be in a numerical format for analysis and processing. NumPy arrays provide an efficient and versatile way to represent image data in a format that can be easily manipulated and fed into ML models. By converting images into NumPy arrays, we can perform various preprocessing steps such as normalization, resizing, and feature extraction, which are crucial for training accurate ML models. Additionally, NumPy arrays allow for seamless integration with PyTorch [2].

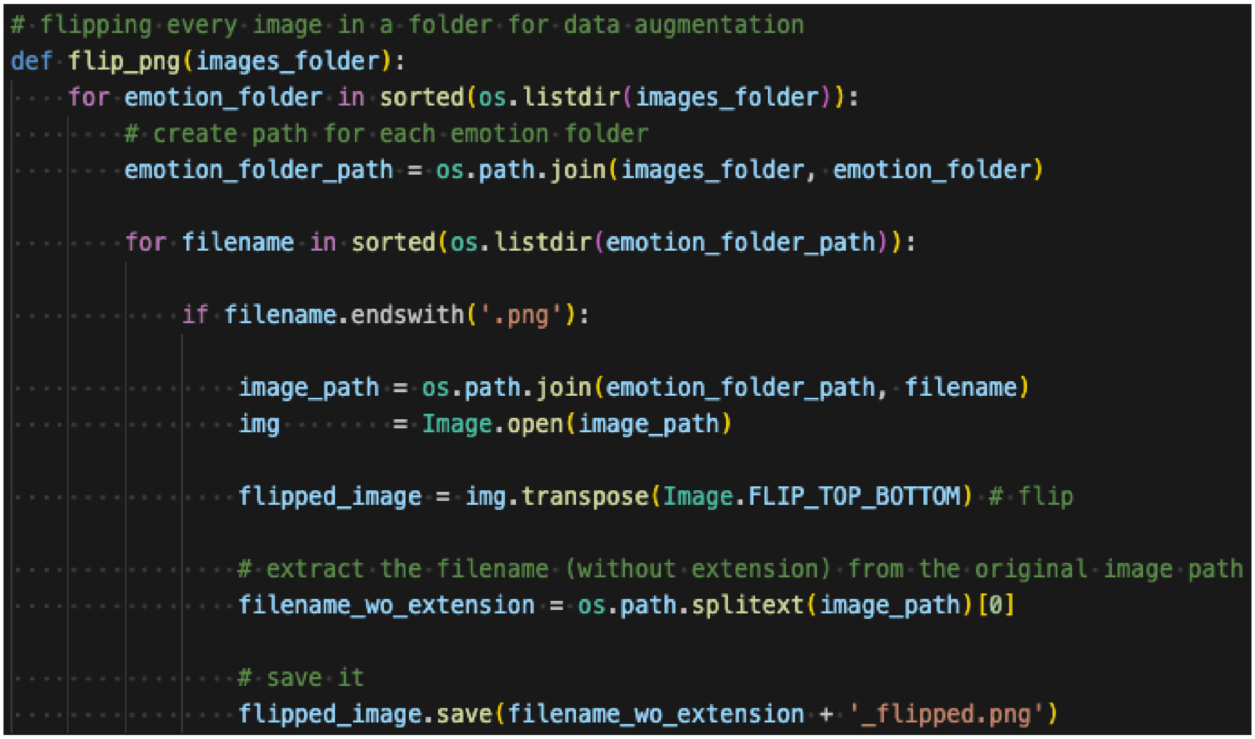


```python
# images to numpy arrays conversion
def png_to_numpy(images_folder):
    # create 2 empty lists, one for features and one for labels
    features = []
    labels   = []

    for emotion_folder in sorted(os.listdir(images_folder)):
        # create path for each emotion folder
        emotion_folder_path = os.path.join(images_folder, emotion_folder)

        for filename in sorted(os.listdir(emotion_folder_path)):

            if filename.endswith('.png'):
                # open the image
                image_path = os.path.join(emotion_folder_path, filename)
                img        = Image.open(image_path)

                # convert the image into a numpy array
                img_array = np.array(img)

                # flatten the array to size 2304 (48x48)
                img_array_flat = img_array.flatten()

                # append the flattened array to the features list
                features.append(img_array_flat)

                # append the label to the labels list
                labels.append(emotion_folder)

        # lastly convert both lists into numpy arrays
        features_np = np.array(features)
        labels_np   = np.array(labels)

    return features_np, labels_np
```

*Figure 2.1. Python code to convert images into NumPy arrays*

### 2.2. Data Augmentation

Image augmentation, specifically rotation, is a fundamental technique in image processing for increasing the diversity and robustness of training data. By rotating images, we can create variations of the original images that simulate real-world scenarios and can potentially improve the model's ability to generalize. Rotation augmentation also helps prevent overfitting by exposing the model to a wider range of perspectives and orientations, thereby enhancing its ability to accurately classify or detect objects in images [3]. Moreover, image rotation augmentation can be easily implemented.



```python
# flipping every image in a folder for data augmentation
def flip_png(images_folder):
    for emotion_folder in sorted(os.listdir(images_folder)):
        # create path for each emotion folder
        emotion_folder_path = os.path.join(images_folder, emotion_folder)

        for filename in sorted(os.listdir(emotion_folder_path)):

            if filename.endswith('.png'):

                image_path = os.path.join(emotion_folder_path, filename)
                img        = Image.open(image_path)

                flipped_image = img.transpose(Image.FLIP_TOP_BOTTOM) # flip

                # extract the filename (without extension) from the original image path
                filename_wo_extension = os.path.splitext(image_path)[0]

                # save it
                flipped_image.save(filename_wo_extension + '_flipped.png')
```

*Figure 2.2. Python code to rotate an image and store the resulting image in the same folder as its original counterpart*

This code goes through every folder and flips any images that it finds vertically. It then saves all the flipped images in the same folder as their original counterparts. This essentially doubles the amount of train and test images.

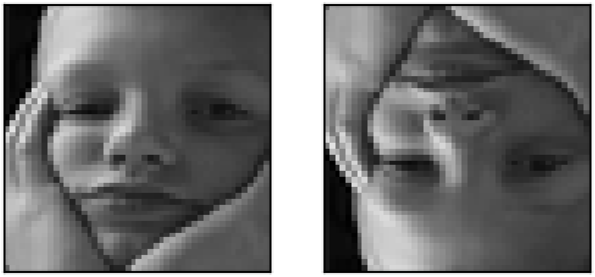


*Figure 2.3. Right (original) vs left (flipped)*

### 2.3. Image Resizing

Resizing images, whether upscaling or downscaling, involves altering their dimensions from the original size to a different size. While resizing can be useful for various purposes, such as standardizing image dimensions or reducing computational complexity, it comes with trade-offs. One significant consideration is the potential loss of information that occurs during resizing. Upscaling images may introduce artifacts and interpolation that degrade image quality, while downscaling can lead to loss of details and fine features. Additionally, resizing images can negatively affect the performance of ML models [4]. As a result, all images obtained from the original dataset are kept at their original dimensions (48x48).

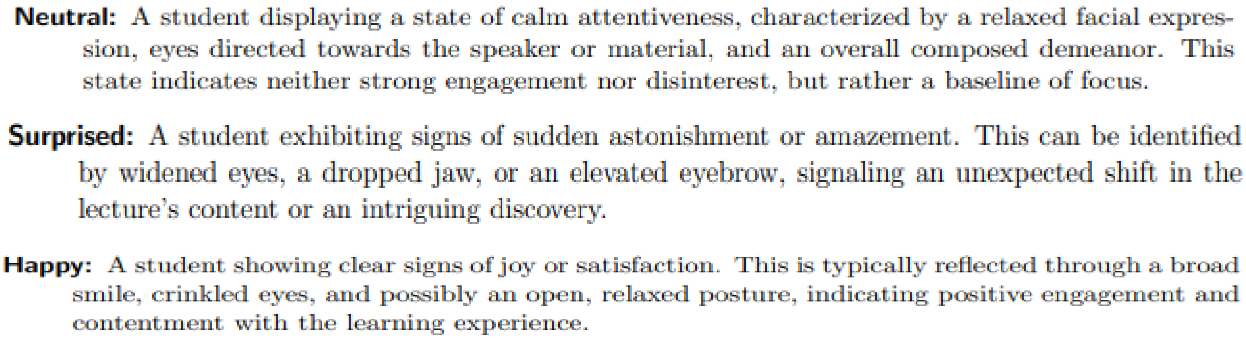
## 3. Labeling

### 3.1. Overview

Our project used the ‘Emotion Detection’ dataset from Kaggle. The dataset provided facial expressions categorized into different emotions such as anger, disgust, fear, happy, neutral, sad and surprise. The fact that the dataset was already pre-labelled helped us a lot in our project. We decided to choose a subset of the dataset which includes Neutral, Surprised and Happy.

### 3.2. Methods

In the initial phase, we spent some time verifying the accuracy of the pre-labeled dataset obtained from Kaggle. We went through each image in its respective folder to ensure that the labeling of the data was accurate. Emotional expression can be subjective from one person to another. To remain consistent, we followed the following guidelines for each category:



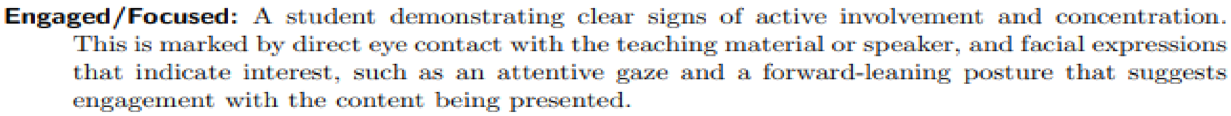
**Neutral:** A student displaying a state of calm attentiveness, characterized by a relaxed facial expression, eyes directed towards the speaker or material, and an overall composed demeanor. This state indicates neither strong engagement nor disinterest, but rather a baseline of focus.

**Surprised:** A student exhibiting signs of sudden astonishment or amazement. This can be identified by widened eyes, a dropped jaw, or an elevated eyebrow, signaling an unexpected shift in the lecture's content or an intriguing discovery.

**Happy:** A student showing clear signs of joy or satisfaction. This is typically reflected through a broad smile, crinkled eyes, and possibly an open, relaxed posture, indicating positive engagement and contentment with the learning experience.

*Figure 3.1. Guidelines/Criterias for our Neutral, Surprised and Happy classes*

A challenge we had was finding an Engaged/Focused dataset. After extensively searching for a dataset that met our requirements and not finding anything that suited our needs for our emotion detection project. We decided to create a subset of “Engaged” from the “Neutral” category. Our criteria for “Engaged” was based on this:



**Engaged/Focused:** A student demonstrating clear signs of active involvement and concentration. This is marked by direct eye contact with the teaching material or speaker, and facial expressions that indicate interest, such as an attentive gaze and a forward-leaning posture that suggests engagement with the content being presented.

*Figure 3.2. Guidelines/Criterias for our Engaged class*

We reviewed each image labeled as “Neutral” to find any indications of engagement. We successfully managed to manually identify 500 images that fit our “Engaged” requirements from the “Neutral” dataset.

### 3.3 Ambiguities

The main ambiguity we had was identifying an image between neutral and engaged since they had a lot of similarities. To address this, we first identified potential candidates for the “Engaged” category. Then that folder was reviewed by our team. By collaboratively reviewing and doing consensus discussions we made sure that the images we chose met our requirements.

Here are examples of our “Engaged” dataset:

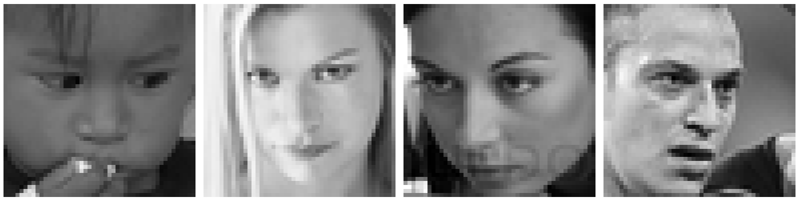


*Figure 3.3. Sample images from our Engaged data*

### 3.4 Bias Attributes

To make sure that our AI system is not only technically proficient but also ethically sound. We analyze our model for two categories which are “Age” and “Gender”. By picking these two attributes, we will be able to evaluate biases in our project when it comes to recognizing emotions.

To label our existing data for those bias attributes, we used VGG Image Annotator which is a manual annotation software. This tool allows us to label our images directly in the browser. For the Gender category we separate our data between “Male” and “Female”. For the Age category, we labeled them as “Young” or “Old”.

Here are examples:

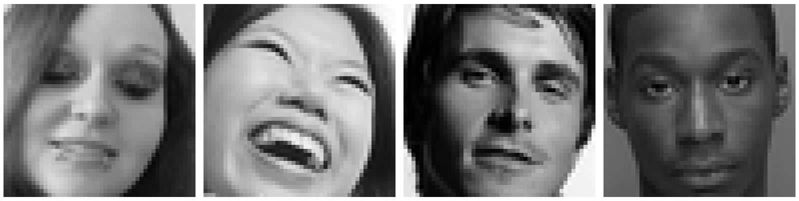


*Figure 3.4. Sample images Females vs Males*

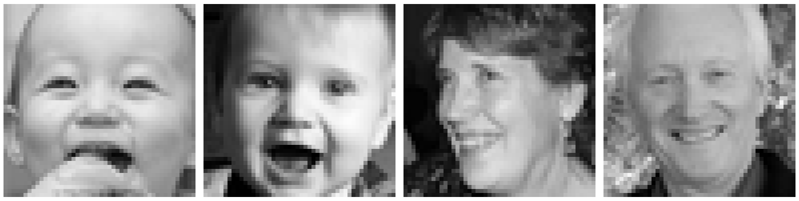


*Figure 3.5. Sample images Young vs Old*

To verify our new labeled data, we went through each image to ensure that the labeling of the data was accurate. For ambiguous cases during the labeling process of the bias attributes, we held review sessions and team discussions. By doing a collaborative approach we maintained high standards in data labeling for our project.

# 4. Data Visualization

## 4.1. Class Distribution

Since all classes have the same amount of images, i.e. 800 (before data augmentation), the class distribution is expected to be even, as evidenced in *Figure 4.1.*

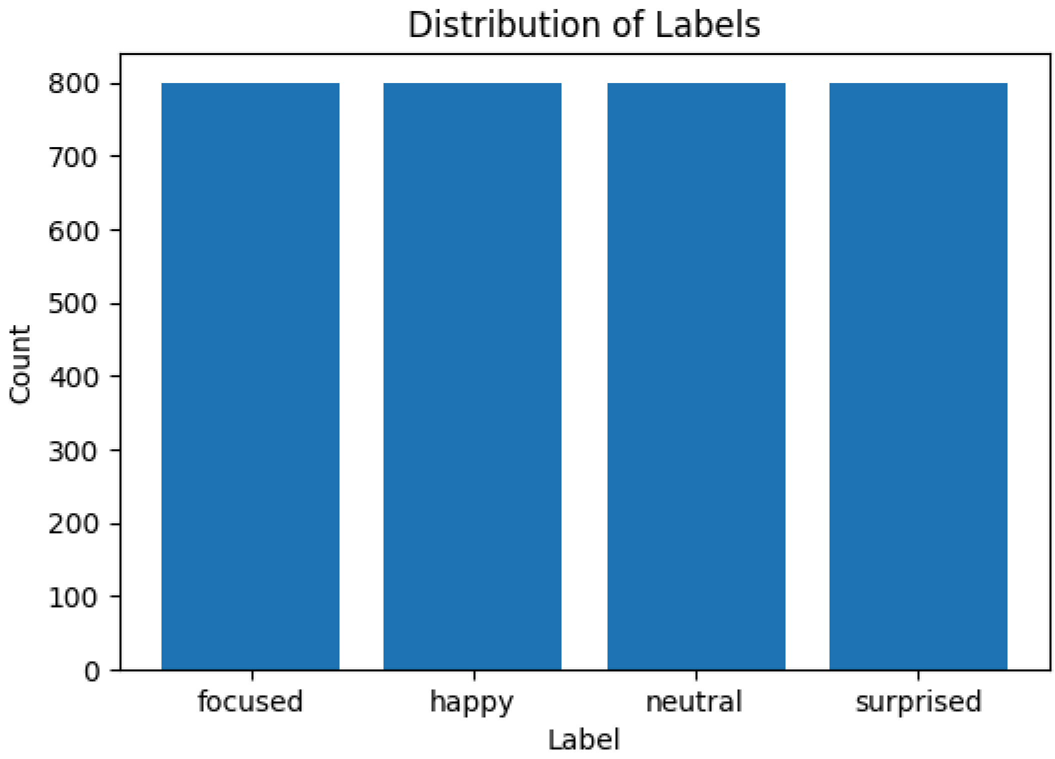
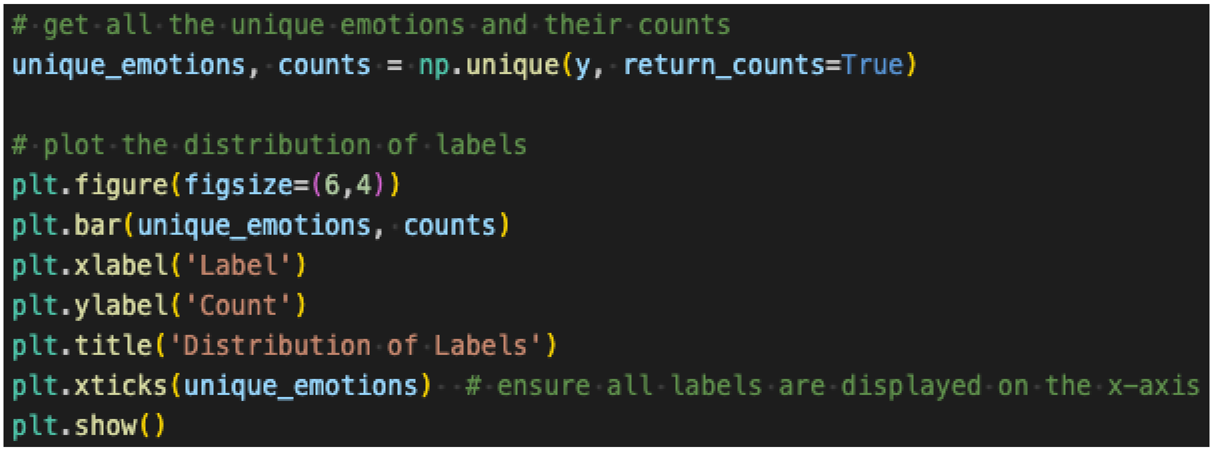


*Figure 4.1. Distribution of images per emotion*



```
# get all the unique emotions and their counts
unique_emotions, counts = np.unique(y, return_counts=True)

# plot the distribution of labels
plt.figure(figsize=(6,4))
plt.bar(unique_emotions, counts)
plt.xlabel('Label')
plt.ylabel('Count')
plt.title('Distribution of Labels')
plt.xticks(unique_emotions)  # ensure all labels are displayed on the x-axis
plt.show()
```

*Figure 4.2. Python code used to plot the distribution above*

### 4.2. Sample Images

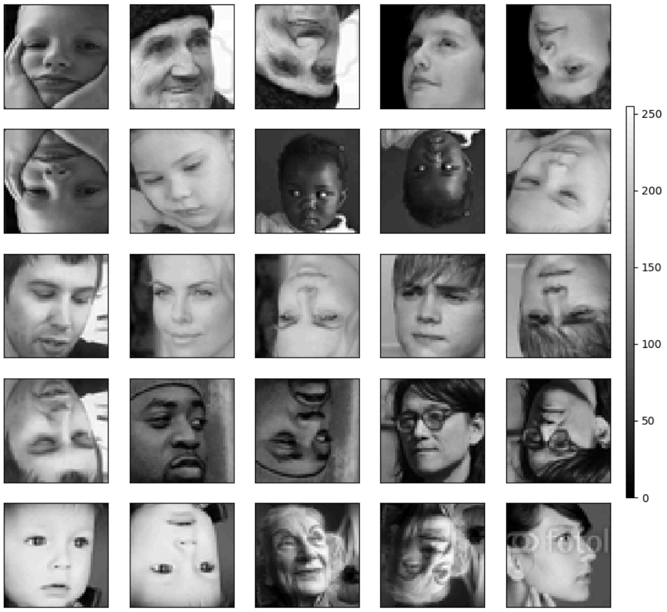


*Figure 4.3. 25 “focused” images*

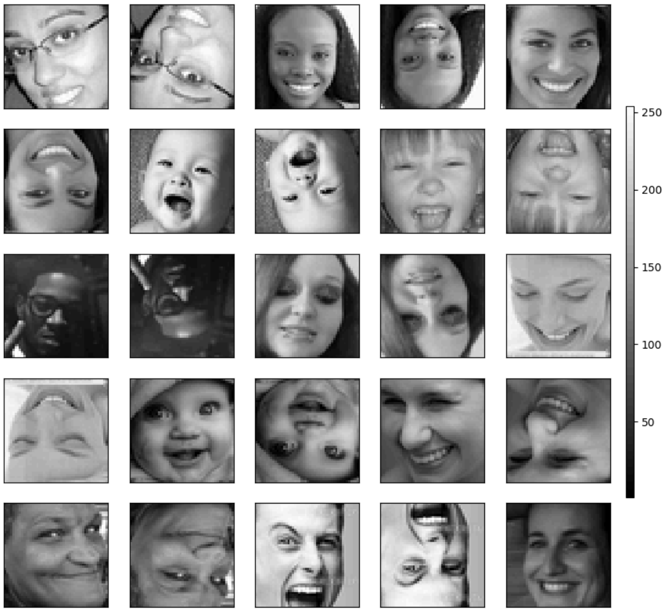


*Figure 4.4. 25 “happy” images*

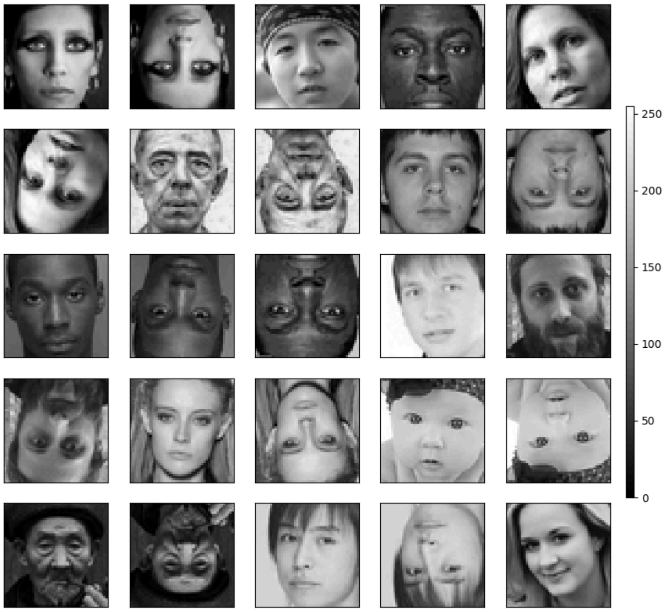


*Figure 4.5. 25 "neutral" images*

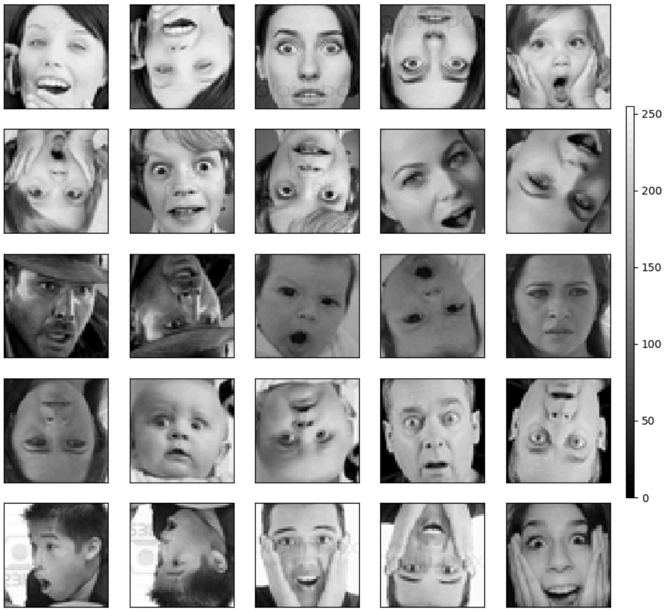
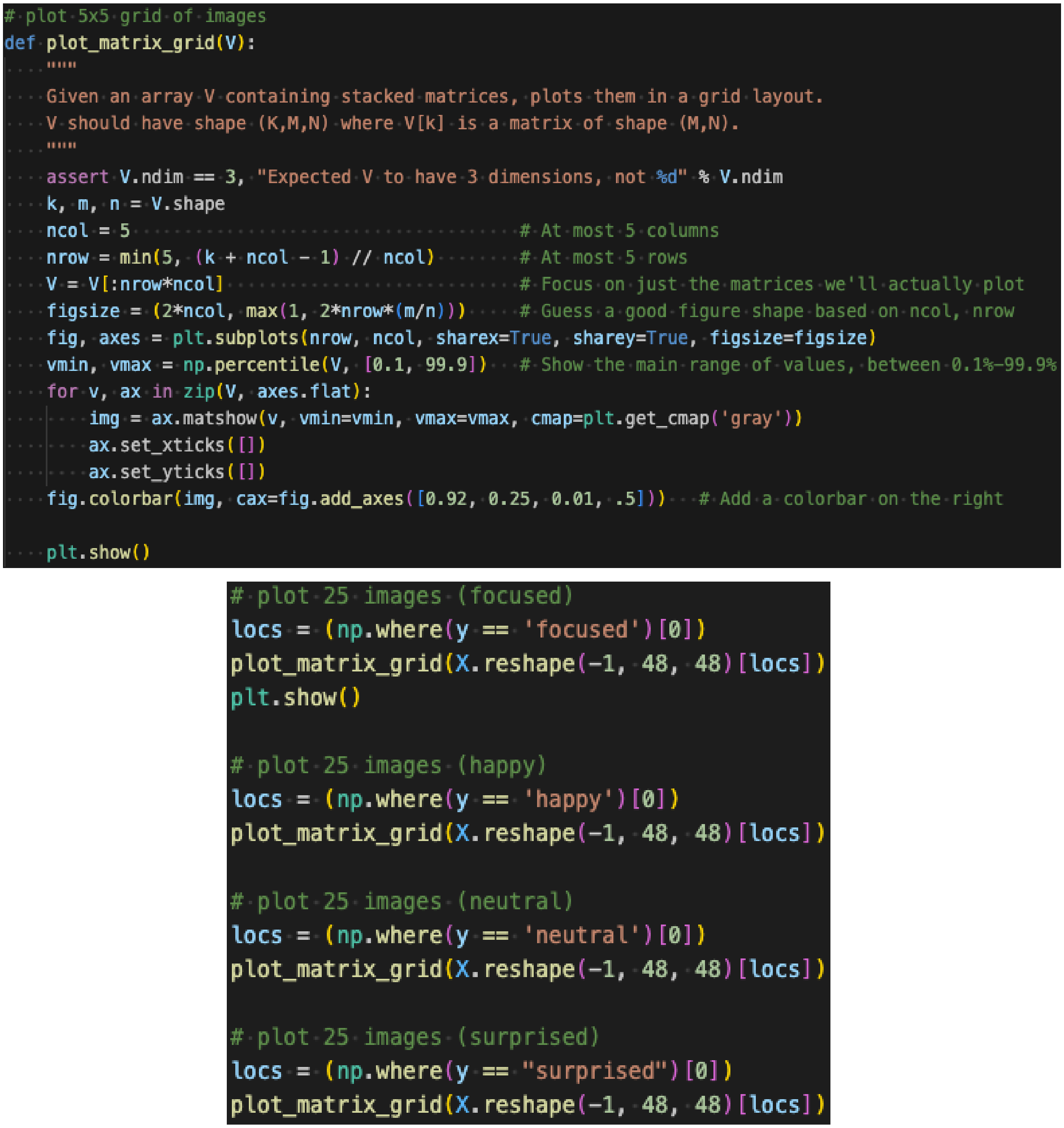


*Figure 4.6. 25 "surprised" images*



```python
# plot 5x5 grid of images
def plot_matrix_grid(V):
    """
    Given an array V containing stacked matrices, plots them in a grid layout.
    V should have shape (K,M,N) where V[k] is a matrix of shape (M,N).
    """
    assert V.ndim == 3, "Expected V to have 3 dimensions, not %d" % V.ndim
    k, m, n = V.shape
    ncol = 5                                      # At most 5 columns
    nrow = min(5, (k + ncol - 1) // ncol)         # At most 5 rows
    V = V[:nrow*ncol]                             # Focus on just the matrices we'll actually plot
    figsize = (2*ncol, max(1, 2*nrow*(m/n)))      # Guess a good figure shape based on ncol, nrow
    fig, axes = plt.subplots(nrow, ncol, sharex=True, sharey=True, figsize=figsize)
    vmin, vmax = np.percentile(V, [0.1, 99.9])    # Show the main range of values, between 0.1%-99.9%
    for v, ax in zip(V, axes.flat):
        img = ax.matshow(v, vmin=vmin, vmax=vmax, cmap=plt.get_cmap('gray'))
        ax.set_xticks([])
        ax.set_yticks([])
    fig.colorbar(img, cax=fig.add_axes([0.92, 0.25, 0.01, .5]))  # Add a colorbar on the right

    plt.show()
```

```python
# plot 25 images (focused)
locs = (np.where(y == 'focused')[0])
plot_matrix_grid(X.reshape(-1, 48, 48)[locs])
plt.show()

# plot 25 images (happy)
locs = (np.where(y == 'happy')[0])
plot_matrix_grid(X.reshape(-1, 48, 48)[locs])

# plot 25 images (neutral)
locs = (np.where(y == 'neutral')[0])
plot_matrix_grid(X.reshape(-1, 48, 48)[locs])

# plot 25 images (surprised)
locs = (np.where(y == "surprised")[0])
plot_matrix_grid(X.reshape(-1, 48, 48)[locs])
```

*Figure 4.7. Python code used to plot the images above (first 25 images of each emotion)*

### 4.3. Pixel Density Distribution

Pixel Intensity Distribution is essential for understanding image characteristics, aiding in feature selection and algorithm design. It provides insights into which features (in this case, pixels) are important in emotion classification [1].

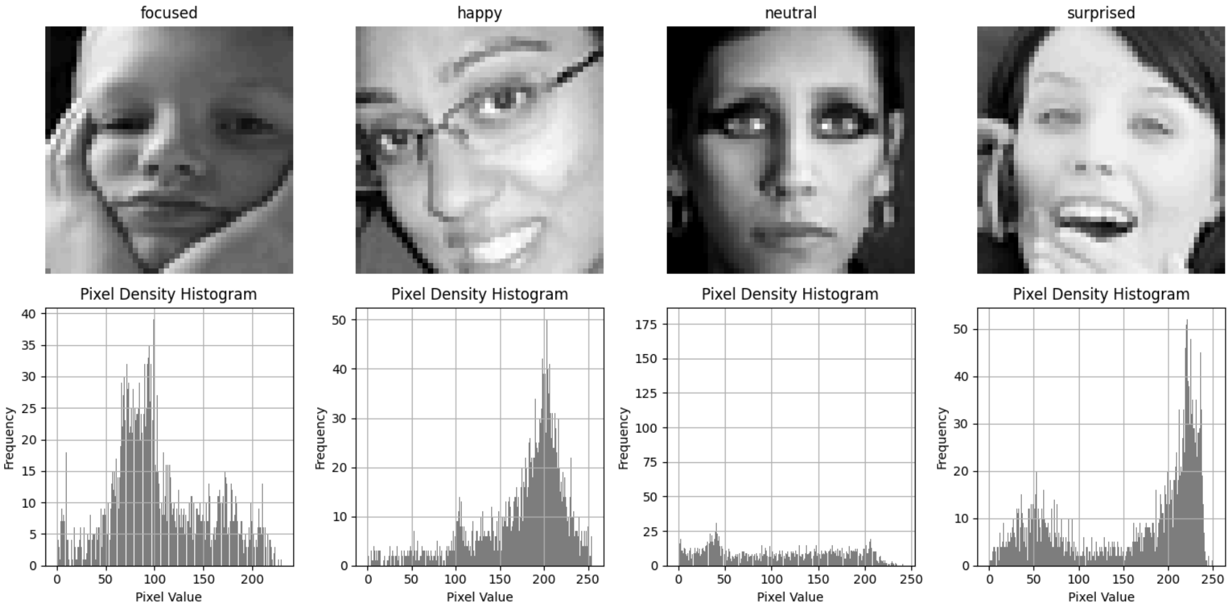
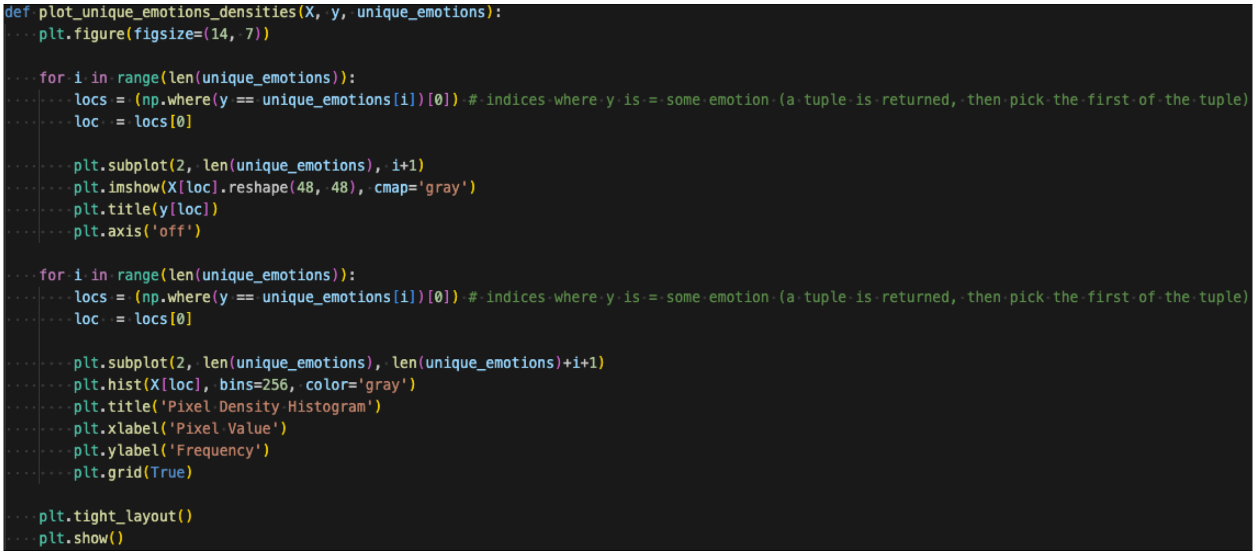


*Figure 4.8. Pixel density histogram for 4 different emotions*



```
def plot_unique_emotions_densities(X, y, unique_emotions):
    plt.figure(figsize=(14, 7))

    for i in range(len(unique_emotions)):
        locs = (np.where(y == unique_emotions[i])[0]) # indices where y is = some emotion (a tuple is returned, then pick the first of the tuple)
        loc  = locs[0]

        plt.subplot(2, len(unique_emotions), i+1)
        plt.imshow(X[loc].reshape(48, 48), cmap='gray')
        plt.title(y[loc])
        plt.axis('off')

    for i in range(len(unique_emotions)):
        locs = (np.where(y == unique_emotions[i])[0]) # indices where y is = some emotion (a tuple is returned, then pick the first of the tuple)
        loc  = locs[0]

        plt.subplot(2, len(unique_emotions), len(unique_emotions)+i+1)
        plt.hist(X[loc], bins=256, color='gray')
        plt.title('Pixel Density Histogram')
        plt.xlabel('Pixel Value')
        plt.ylabel('Frequency')
        plt.grid(True)

    plt.tight_layout()
    plt.show()
```

*Figure 4.9. Python code used to plot the histograms above*

From the histograms, it seems that the "focused" has the most normal distribution out of the 4 emotions. "Happy" and "surprised", on the other hand, seems to skew towards extremities, with "happy" being strongly skewed toward the right side and "surprised", while also skewed toward the right side, having some pixel density activity on the left. Lastly, the pixel distribution for "neutral" seems almost constant. Keep in mind that the person's complexion, the angle of the image, brightness and contrast all impact the output of the pixel density distribution [1]. It is also worth noting that, in this case, it is possible to mistake the "surprised" image for a "happy" one and this is evidenced by the fact that their histograms are all but identical.

## 4.4. Further data processing

As can be seen from the previous section, the images have pixels that range from 0 to 255 (indicative of grayscale images). As an attempt to further process the data, data normalization is applied, a process by which all images end up having a pixel range between -2 and 2. This process also helps in stabilizing and accelerating the training of machine learning models by making the optimization process more efficient and effective. Not only that normalization also aids in reducing the impact of varying pixel value scales across different images, which can lead to more reliable and consistent model performance [2].

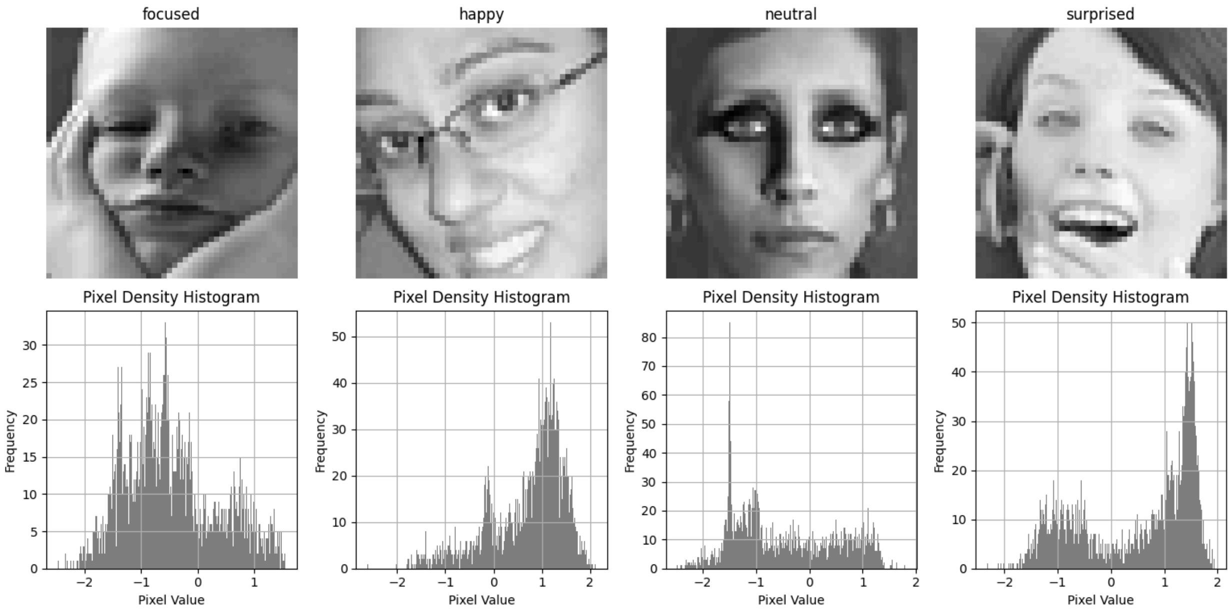
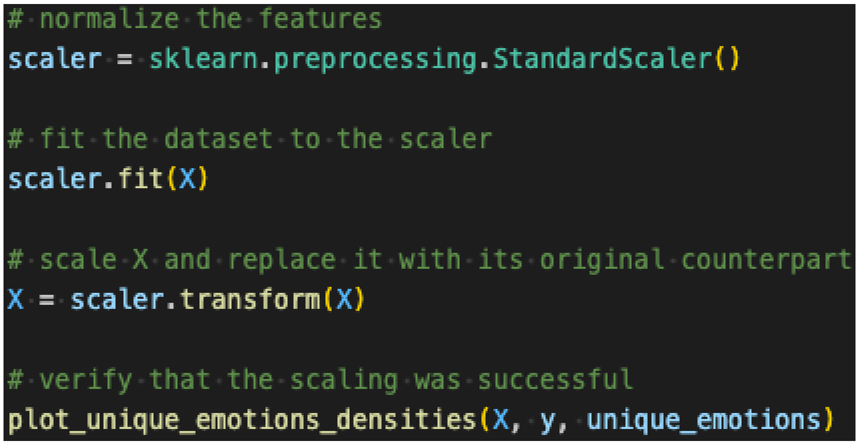


*Figure 4.10. Pixel density histogram for 4 different emotions (after normalization)*



```
# normalize the features
scaler = sklearn.preprocessing.StandardScaler()

# fit the dataset to the scaler
scaler.fit(X)

# scale X and replace it with its original counterpart
X = scaler.transform(X)

# verify that the scaling was successful
plot_unique_emotions_densities(X, y, unique_emotions)
```

*Figure 4.11. Python code (with scikit-learn) used to normalize and plot the histograms above*

With the exception of “neutral”, all other emotions seem to stay the same distribution-wise and shape-wise, albeit the pixel range being a lot smaller than it was before. For “neutral”, the pixel distribution is much clearer and easier to see. However, there’s a spike present in the distribution, which might be harmful to the training of ML models down the line.

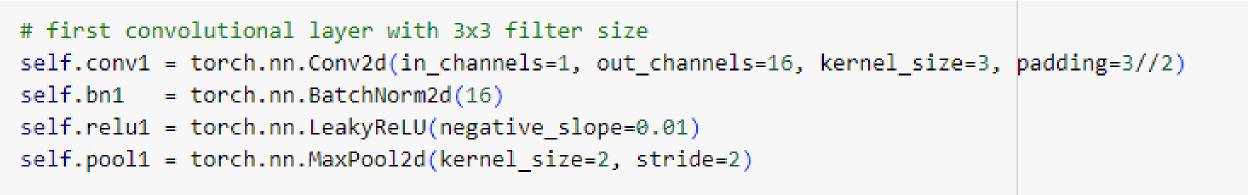
# 5. CNN Architecture

## 5.1. Introduction

Our CNN model is designed for facial image classification. It’s built using PyTorch which is a deep learning framework that helps design complex model architecture. The model processes input images through convolutional, activation and pooling layers for classification

## 5.2. Architecture Details

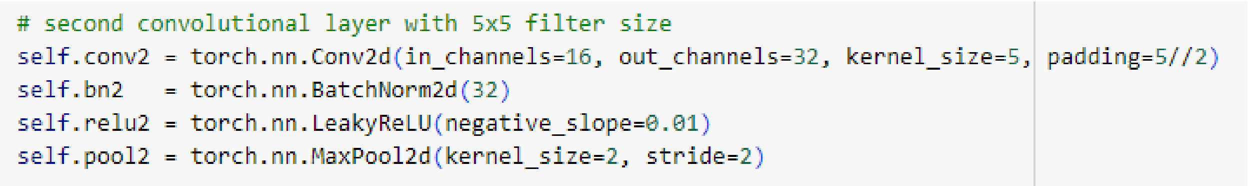
The input layer is our foundational layer. It accepts grayscale facial images of 48x48 pixels. Then, we have 3 convolutional layers which constitute the backbone of our model. The first convolutional layer initiates the feature extraction process. This layer uses 16 filters of size 3x3. After convolution, batch normalization and LeakyReLU with a negative slope of 0.01 are applied in the learning process.



```
# first convolutional layer with 3x3 filter size
self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, padding=3//2)
self.bn1   = torch.nn.BatchNorm2d(16)
self.relu1 = torch.nn.LeakyReLU(negative_slope=0.01)
self.pool1 = torch.nn.MaxPool2d(kernel_size=2, stride=2)
```

*Figure 5.1. Python code of First Convolutional Layer*

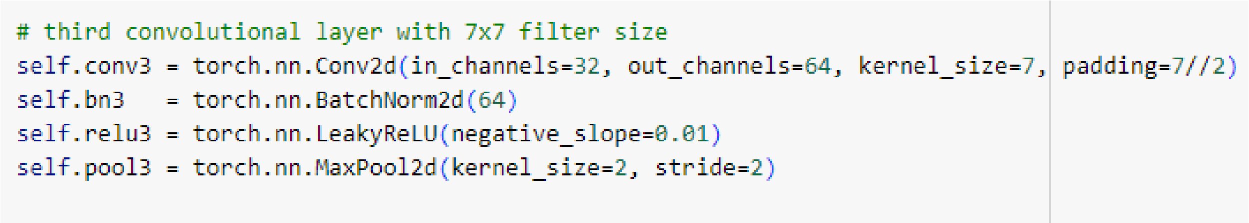
The second convolutional layer increases the model’s complexity. This layer uses 32 filters of size 5x5.



```
# second convolutional layer with 5x5 filter size
self.conv2 = torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, padding=5//2)
self.bn2   = torch.nn.BatchNorm2d(32)
self.relu2 = torch.nn.LeakyReLU(negative_slope=0.01)
self.pool2 = torch.nn.MaxPool2d(kernel_size=2, stride=2)
```

*Figure 5.2. Python code of Second Convolutional Layer*

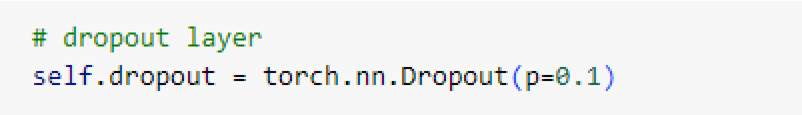
The third and last convolutional layer has 64 filters of size 7x7. Each of those convolutional layers extract and learn features from the input images which is important for our Convolutional Neural Network model.



```
# third convolutional layer with 7x7 filter size
self.conv3 = torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=7, padding=7//2)
self.bn3   = torch.nn.BatchNorm2d(64)
self.relu3 = torch.nn.LeakyReLU(negative_slope=0.01)
self.pool3 = torch.nn.MaxPool2d(kernel_size=2, stride=2)
```

*Figure 5.3. Python code of Third Convolutional Layer*

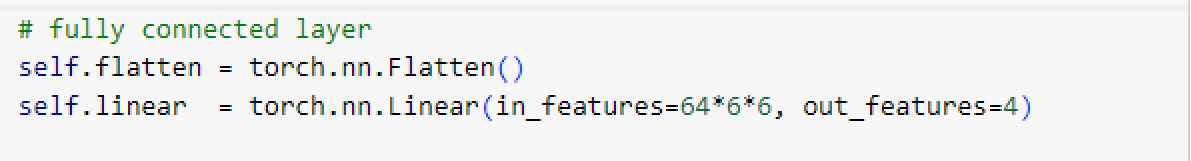
As we can see, our convolutional layers are followed by batch normalization which normalizes the data for the subsequent layer. It also speeds up the training. Afterwards, we applied the activation function LeakyRELU to enhance the model. We choose a negative slope of 0.01 to mitigate the vanishing gradient issue. Then, we got the MaxPooling layers with a 2x2 kernel and a stride of 2 which reduce the spatial dimensions by half after our 3 convolutional stages. Additionally, we have a dropout layer with a probability of “p = 0.1” which helps us reduce overfitting by ignoring a subset of the feature during each training epoch.



```
# dropout layer
self.dropout = torch.nn.Dropout(p=0.1)
```

*Figure 5.4. Python code of Dropout Layer*

Finally, we have our fully connected layer that flattened by converting the 2D feature maps into a 1D vector. Then, it maps the features to our output classes. Our CNN model classifies images into one of four emotions categories.

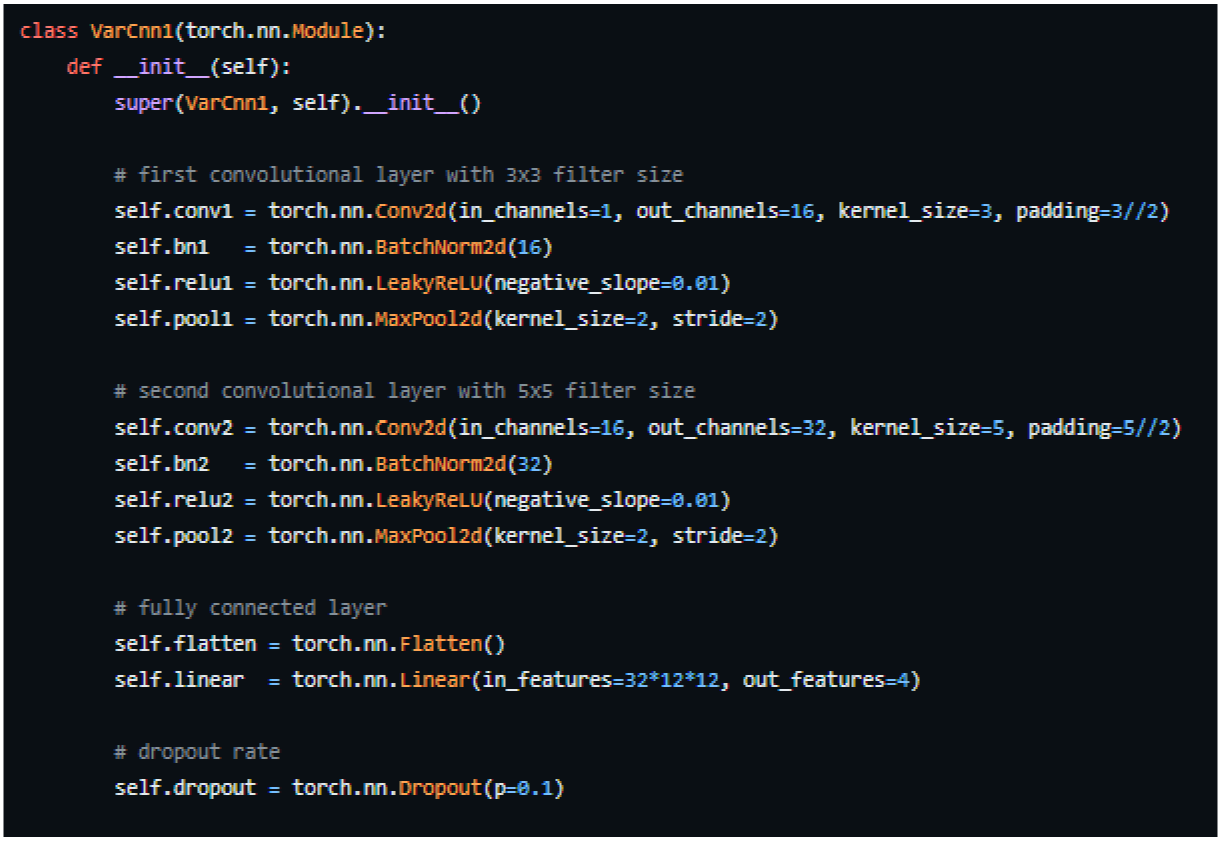


```
# fully connected layer
self.flatten = torch.nn.Flatten()
self.linear  = torch.nn.Linear(in_features=64*6*6, out_features=4)
```

*Figure 5.5. Python code of Fully Connected Layer*

### 5.3. Variant 1 & 2

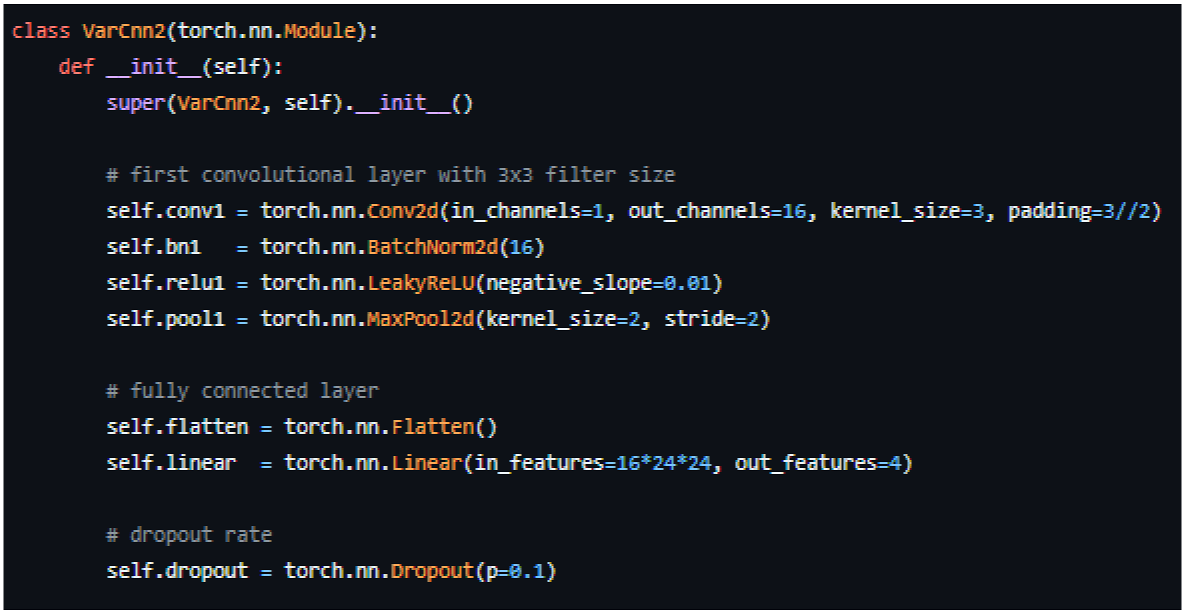
In this project, we experimented with two different versions of our main CNN model to see how simplifying the CNN architecture would impact its performance. For variant 1, it only has two convolutional layers compared to three from the main model. We kept the 3x3 and 5x5 filter sizes but removed the third convolutional with 64 filters of size 7x7.



```
class VarCnn1(torch.nn.Module):
    def __init__(self):
        super(VarCnn1, self).__init__()

        # first convolutional layer with 3x3 filter size
        self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, padding=3//2)
        self.bn1   = torch.nn.BatchNorm2d(16)
        self.relu1 = torch.nn.LeakyReLU(negative_slope=0.01)
        self.pool1 = torch.nn.MaxPool2d(kernel_size=2, stride=2)

        # second convolutional layer with 5x5 filter size
        self.conv2 = torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, padding=5//2)
        self.bn2   = torch.nn.BatchNorm2d(32)
        self.relu2 = torch.nn.LeakyReLU(negative_slope=0.01)
        self.pool2 = torch.nn.MaxPool2d(kernel_size=2, stride=2)

        # fully connected layer
        self.flatten = torch.nn.Flatten()
        self.linear  = torch.nn.Linear(in_features=32*12*12, out_features=4)

        # dropout rate
        self.dropout = torch.nn.Dropout(p=0.1)
```

*Figure 5.5. Variant 1 of our Main CNN Model*

For Variant 2, we further simplified the architecture by removing two convolutional layers present in the main model. We are only using a single convolutional layer with 16 filters of size 3x3 for this variant. In both variants, we made changes to the fully connected layer to adjust with the new architecture. The dropout stays the same across all models. With Variant 1 and 2, we want to observe how simplifying the model will impact its ability to classify images in our project.

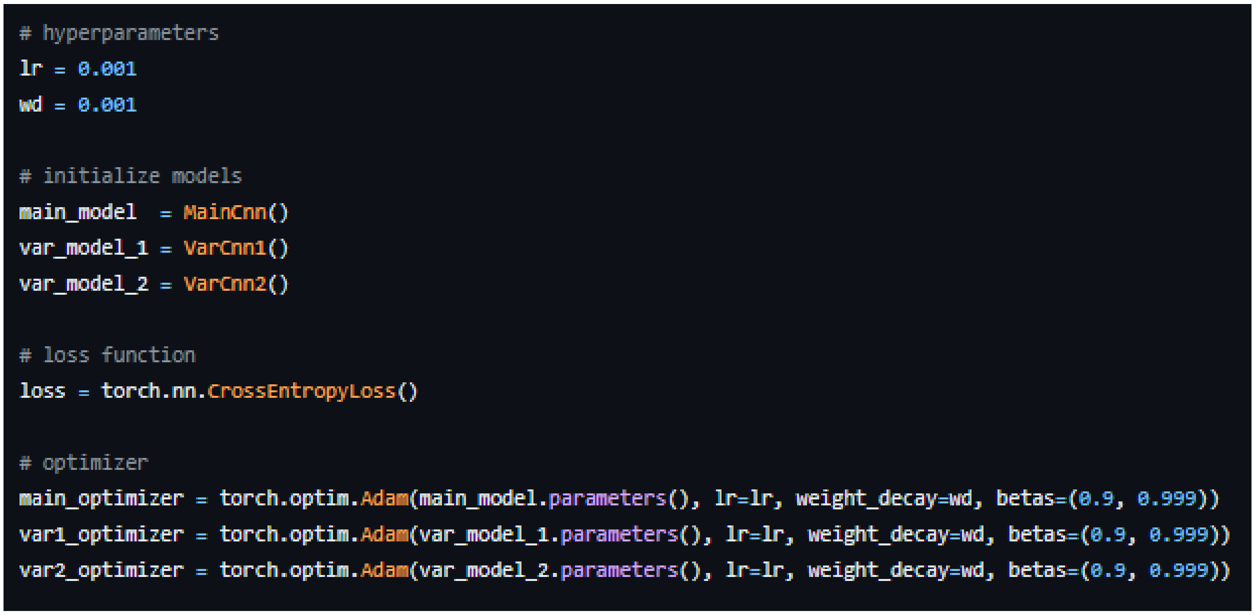


```python
class VarCnn2(torch.nn.Module):
    def __init__(self):
        super(VarCnn2, self).__init__()

        # first convolutional layer with 3x3 filter size
        self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, padding=3//2)
        self.bn1   = torch.nn.BatchNorm2d(16)
        self.relu1 = torch.nn.LeakyReLU(negative_slope=0.01)
        self.pool1 = torch.nn.MaxPool2d(kernel_size=2, stride=2)

        # fully connected layer
        self.flatten = torch.nn.Flatten()
        self.linear  = torch.nn.Linear(in_features=16*24*24, out_features=4)

        # dropout rate
        self.dropout = torch.nn.Dropout(p=0.1)
```

*Figure 5.6. Variant 2 of our Main CNN Model*

### 5.4. Training Process

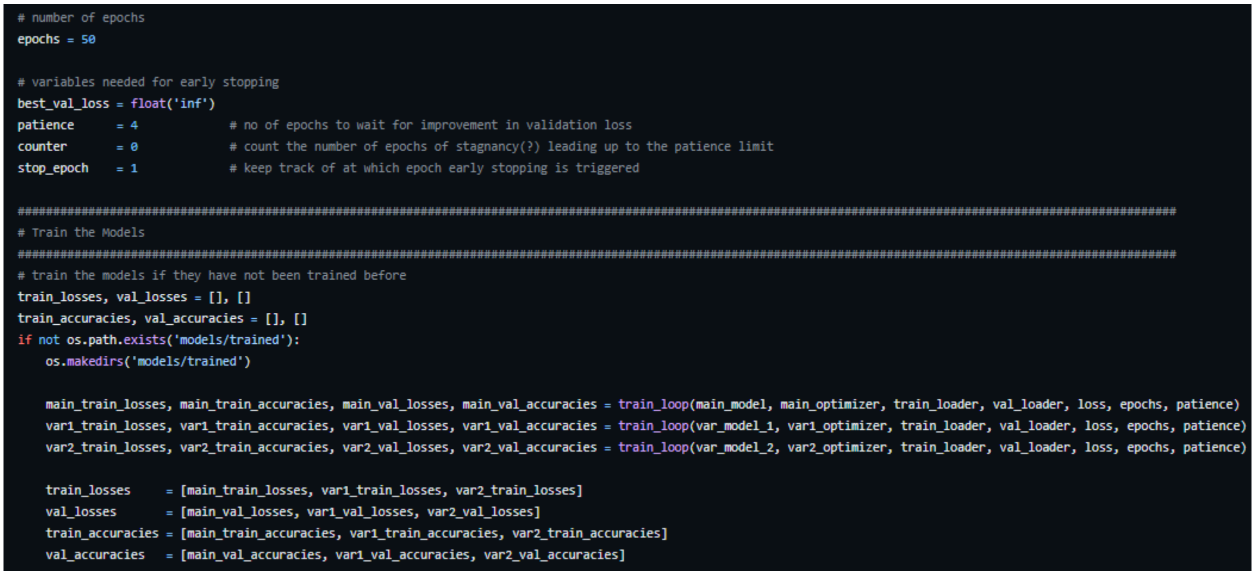
For the training process, we set our learning rate (lr) to 0.001 and a weight decay (wd) to 0.001. This makes sure that our model learns at a good pace without making huge leaps that could potentially mess up the learning process. We used CrossEntropyLoss for our loss function which is good since we have to classify 4 classes. For optimizing the model, we used Adam optimizer for our three models because it adjusts the learning rate as it goes.



```python
# hyperparameters
lr = 0.001
wd = 0.001

# initialize models
main_model  = MainCnn()
var_model_1 = VarCnn1()
var_model_2 = VarCnn2()

# loss function
loss = torch.nn.CrossEntropyLoss()

# optimizer
main_optimizer = torch.optim.Adam(main_model.parameters(), lr=lr, weight_decay=wd, betas=(0.9, 0.999))
var1_optimizer = torch.optim.Adam(var_model_1.parameters(), lr=lr, weight_decay=wd, betas=(0.9, 0.999))
var2_optimizer = torch.optim.Adam(var_model_2.parameters(), lr=lr, weight_decay=wd, betas=(0.9, 0.999))
```

*Figure 5.7. Hyperparameters, CrossEntropyLoss function and Adam optimizer*

We decided to train our models for 50 epochs. However we did set up early stopping with a patience of 4 epochs. This means that we stop the training if our model doesn't get better at predicting the validation set for 4 straight epochs. This will help us save time and avoid the model from overfitting to the training data. For monitoring, we were tracking training/validation losses/accuracies. We were able to know how well the models were adapting and learning.

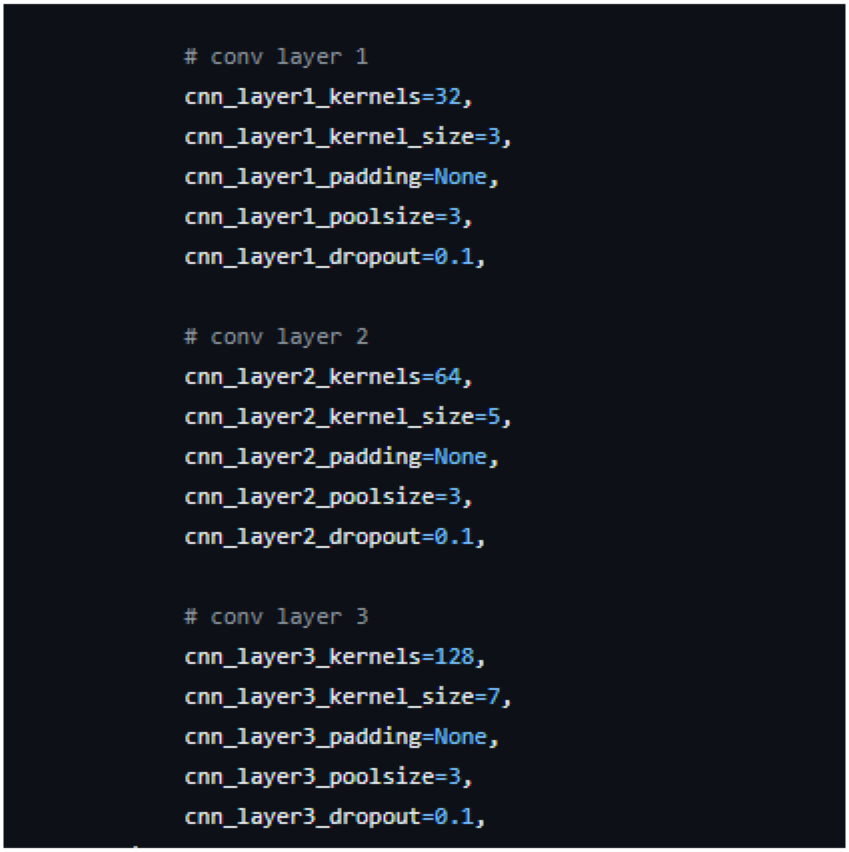


```
# number of epochs
epochs = 50

# variables needed for early stopping
best_val_loss = float('inf')
patience      = 4             # no of epochs to wait for improvement in validation loss
counter       = 0             # count the number of epochs of stagnancy(?) leading up to the patience limit
stop_epoch    = 1             # keep track of at which epoch early stopping is triggered

####################################################################################################################################################################
# Train the Models
####################################################################################################################################################################
# train the models if they have not been trained before
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []
if not os.path.exists('models/trained'):
    os.makedirs('models/trained')

    main_train_losses, main_train_accuracies, main_val_losses, main_val_accuracies = train_loop(main_model, main_optimizer, train_loader, val_loader, loss, epochs, patience)
    var1_train_losses, var1_train_accuracies, var1_val_losses, var1_val_accuracies = train_loop(var_model_1, var1_optimizer, train_loader, val_loader, loss, epochs, patience)
    var2_train_losses, var2_train_accuracies, var2_val_losses, var2_val_accuracies = train_loop(var_model_2, var2_optimizer, train_loader, val_loader, loss, epochs, patience)

    train_losses     = [main_train_losses, var1_train_losses, var2_train_losses]
    val_losses       = [main_val_losses, var1_val_losses, var2_val_losses]
    train_accuracies = [main_train_accuracies, var1_train_accuracies, var2_train_accuracies]
    val_accuracies   = [main_val_accuracies, var1_val_accuracies, var2_val_accuracies]
```

*Figure 5.8. Epochs and early stopping*

### 5.5. Changes to the CNN Architecture

We are always trying to enhance the performance of our Convolutional Neural Network model. For the final version, we decided to double the number of filters in each convolutional layer. The goal of these changes is to increase the capacity of the model to learn more complex features at each layer. For the first convolutional layer, we increased the number of filters from 16 to 32. For the second convolutional layer, we increased the number of filters from 32 to 64. For the third convolutional layer, we increased the number of filters from 64 to 128. We kept the kernel size the same for each convolutional layer and only double the filters. This final version of our CNN Architecture, with increased filters across all convolutional layers will help make our model better at recognizing emotions.



```
        # conv layer 1
        cnn_layer1_kernels=32,
        cnn_layer1_kernel_size=3,
        cnn_layer1_padding=None,
        cnn_layer1_poolsize=3,
        cnn_layer1_dropout=0.1,

        # conv layer 2
        cnn_layer2_kernels=64,
        cnn_layer2_kernel_size=5,
        cnn_layer2_padding=None,
        cnn_layer2_poolsize=3,
        cnn_layer2_dropout=0.1,

        # conv layer 3
        cnn_layer3_kernels=128,
        cnn_layer3_kernel_size=7,
        cnn_layer3_padding=None,
        cnn_layer3_poolsize=3,
        cnn_layer3_dropout=0.1,
```

*Figure 5.9. Final Version of our Convolutional Layers*

# 6. Evaluation

## 6.1 Model Performance By Confusion Matrix:

6.1.1 The Main CNN (3 convolution layers)

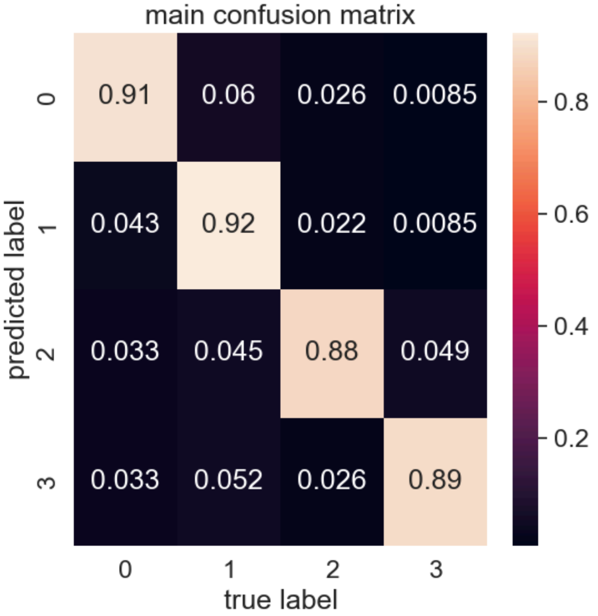


*Figure 6.1. Confusion matrix of 3-layers CNN model*

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Facial expression | focused | happy | neutral | surprised |

- Most frequently confused classes are Focused and Happy. The main reason is there is many similarities between those 2 classes. For example, many individuals with open mouths, smiling or engaged expression. These similarities will lead to overlapping facial features that makes the model difficult to differentiate. Another 2 confusing classes are between neutral and surprised. This is due to the slight facial features resembling such as widen eyes or raised eyebrows.
- Well-recognized classes are happy and focused. Since the facial expression of focused and neutral are quite different from the other two expressions. Therefore, focused and neutral are well separated from the rest. Moreover, the model still be able to capture the different between the very similar facial expressions (focused and neutral) due to the smaller eyes opening in neutral. In the case of happy, the unique facial expression only appear in this class is the widening mouth. This feature of the facial expression contribute to the success in correctly picking up happy faces.

### 6.1.2 The First CNN Variant (2 convolution layers)

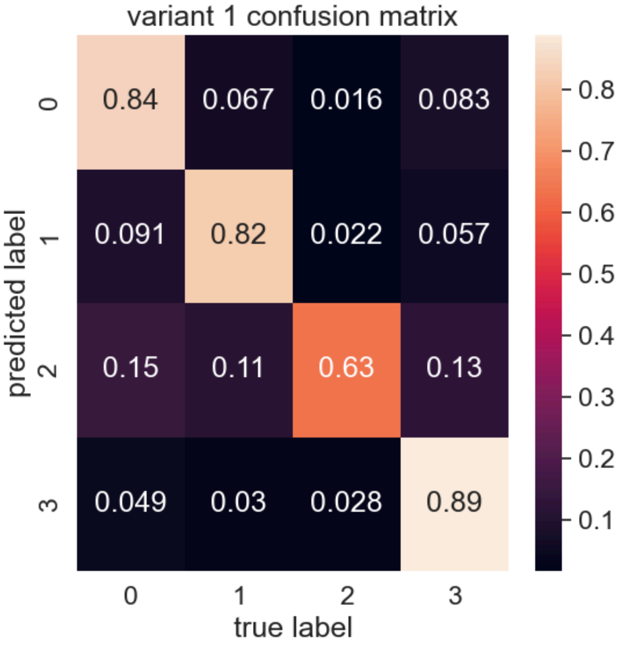


*Figure 6.2. Confusion matrix of 2-layers CNN model*

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Facial expression | focused | happy | neutral | surprised |

- Most frequently confused classes are focused and neutral. This result suggests that these 2 classes share common facial features such as both classes have the most eyes opening (which is not a case in neutral and happy since people tend to close eyes while being bored or neutral and in case of being happy, people's cheek tend to rise and make the eyes smaller than usual). In addition, one convolution layer is missing in the second variant leading to the way less number of chanel. The model has 32 channels comparing to 64 in the main model. This fail the model in generalization.
- Well-recognized classes are focused and surprised. With one less convolution layer, the model still can do good in separating those 2 classes from the rest classes. Since less detail could be captured by the model, the neutral class is more likely to be categorized as neutral and this increase the chance to pick up most of the focused faces. In the case of surprised, the distinct mouth opening feature help model to differentiate surprised from other states.

6.13 The Second CNN Variant (1 convolution layer)

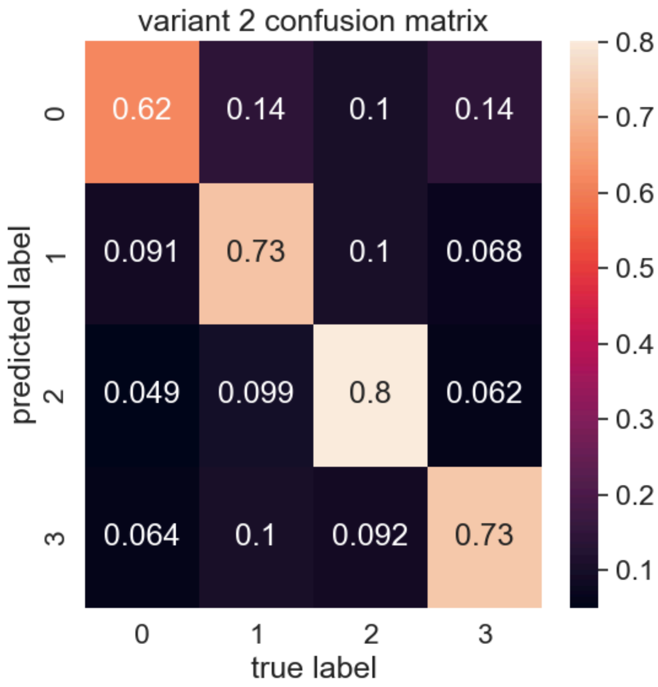


*Figure 6.3. Confusion matrix of 1-layer CNN model*

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Facial expression | focused | happy | neutral | surprised |

- Most frequently confused classes are between focused , happy and surprised. In this model, there is only a convolution layer then the model is only able to recognize very basic line such as vertical, horizontal, diagonal lines, simple curve. Moreover, the similarities between those 3 classes making the poor-equiped CNN model hard to distinguish such as facial expressions have plenty straight lines like neutral eyes tend to be horizontal line, linear shape of the mouth in both facial which is well captured in the first layer of the convolution network. On the top of that, the image is in size of 48 x 48 which is quite poor. Therefore, with one layer, the model will miss a lot of information which is only happen in the next 2 layers where those base lines will be combined to become more abstract pattern.
- Well-recognized class is neutral. Firstly, the neutral has many common features to other classes. Secondly, the model is conservative in picking up the neutral. The model only effective indetecting subtle differences like the presence of eyebrows or open eyes.

### 6.2 Performance Metrics For 3 CNN Models

| Model | Macro | | | Micro | | | Accuracy |
|---|---|---|---|---|---|---|---|
| | P | R | F | P | R | F | |
| Main model | 0.900 | 0.899 | 0.899 | 0.899 | 0.899 | 0.899 | 0.899 |
| Variant 1 (2 conv layers) | 0.804 | 0.796 | 0.793 | 0.794 | 0.794 | 0.794 | 0.794 |
| Variant 2 (1 conv layer) | 0.722 | 0.720 | 0.719 | 0.721 | 0.721 | 0.721 | 0.721 |

- The main model, featuring 3 convolution layers, emerges as the most effective among the three variants, boasting the highest precision, recall, and F1 scores. In contrast, the variant with 2 convolution layers exhibits slightly diminished performance across these metrics, signaling a moderate decrease compared to the main model. Notably, the second variant, equipped with only 1 convolution layer, experiences a substantial decline in performance relative to both the main and first variant models. This observation suggests a proportional relationship between model performance and the complexity of its architecture, whereby an increase in the number of convolution layers corresponds to enhanced predictive capabilities
- The precision scores of all three models is higher their respective recall scores. This suggests a conservative tendency in identifying certain facial expressions, accuracy is preferred over comprehensiveness. Another way of speaking, the three models will more likely to identify the image of facial emotion by their surface pattern but not really understand the true meaning of that facial emotion. For instance, some people will not open mouth when they are surprised and this will be misclassified to neutral state by the models. Consequently, the models show fewer false positives, enhancing confidence in the correctness of their predictions. Conversely, the lower recall scores indicate a higher likelihood of missing true positives, implying that the models may overlook some instances of the target classess.

### 6.3 Impact of architectural variations

#### 6.3.1 The Depth (number of Convolution Layers)

Increasing the depth of a CNN model can have both positive and negative effects on performance. If the model has very few Convolution Layer then increasing the number of layer will bring benefits such as the model is able to learn more abstract patterns which means that the model recognize the relationship between local features. In contrast, if the model has decent number of layer. Then increasing layers will lead model get significant good performance on training set but pretty bad on test set due to overfitting. Furthermore, the model starts to remember specific features and lose the generalization ability. In our project, it is underfitting when model has 1 layer and overfitting when having 5 layers.

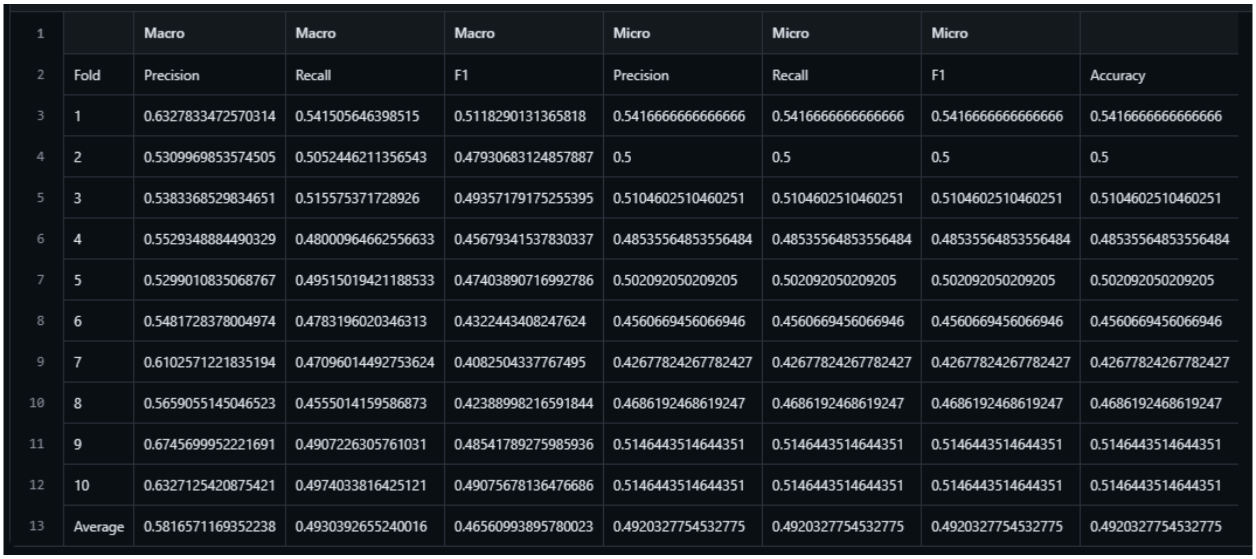
#### 6.3.2 The Kernel Size Variation.

The size of kernel directly influences the receptive field of a model which determines the scale of features the model can detect. Smaller kernel size (3x3 or 5x5) allows the model to capture finer details and localized patterns in the train set. This is beneficial in recognizing small presence of wrinkles or dimples. Meanwhile, larger kernel size (7x7 or 9x9) enable the model to capture broader features and spatial relationships across different parts of faces. This is crucial to detect large facial structures that covers many successive parts of the face such as the head orientation or facial symmetry.

### 6.4 Conclusions and Forward Look

- The main CNN model with 3 convolution layer outperformed its variants which has less layers. With this number of Convolution Layer, the model is able to capture small detail such as basic lines due to the small size kernel (3x3). Then, in the second layer, bigger kernel size (5x5) finds the relationships between those primary lines to form more abstract image like the shape of eyes or lips. Finally, the last layer's kernel (7x7) recognize larger view which is the relationships between parts from second layer. Eventhough, the CNN model scores ~90% for all metrics, there are some aspect still has room for improvement. Moreover, the learning rate is decent to ensure the step is not zigzag nor converge too slow. Some technics are applied to improve the model's performance like setting Drop Out = 10%, Weight Decay and Batch Normalization to void overfitting.
- There are some rooms for improvement.
    - Expand the diversity of dataset by including wide range of age and race, and increasing the size of dataset. This will improve the model's generalization capability, and its robustness to variations in facial apearances.
    - Grid Search will help in finding the best combination of set of hyperparameters.
    - We could apply Ensemble Learning byb combining predictions of many individual models to improve overall performance. Some approaches like voting or bagging.

### 6.5 K-Fold Cross Validation of Final Model.



| | Macro | Macro | Macro | Micro | Micro | Micro | |
|---|---|---|---|---|---|---|---|
| Fold | Precision | Recall | F1 | Precision | Recall | F1 | Accuracy |
| 1 | 0.6327833472570314 | 0.541505646398515 | 0.5118290131365818 | 0.5416666666666666 | 0.5416666666666666 | 0.5416666666666666 | 0.5416666666666666 |
| 2 | 0.5309969853574505 | 0.5052446211356543 | 0.47930683124857887 | 0.5 | 0.5 | 0.5 | 0.5 |
| 3 | 0.5383368529834651 | 0.515575371728926 | 0.49357179175255395 | 0.5104602510460251 | 0.5104602510460251 | 0.5104602510460251 | 0.5104602510460251 |
| 4 | 0.5529348884490329 | 0.48000964662556633 | 0.45679341537830337 | 0.48535564853556484 | 0.48535564853556484 | 0.48535564853556484 | 0.48535564853556484 |
| 5 | 0.5299010835068767 | 0.49515019421188533 | 0.47403890716992786 | 0.502092050209205 | 0.502092050209205 | 0.502092050209205 | 0.502092050209205 |
| 6 | 0.5481728378004974 | 0.4783196020346313 | 0.4322443408247624 | 0.4560669456066946 | 0.4560669456066946 | 0.4560669456066946 | 0.4560669456066946 |
| 7 | 0.6102571221835194 | 0.47096014492753624 | 0.4082504337767495 | 0.42677824267782427 | 0.42677824267782427 | 0.42677824267782427 | 0.42677824267782427 |
| 8 | 0.5659055145046523 | 0.4555014159586873 | 0.42388998216591844 | 0.4686192468619247 | 0.4686192468619247 | 0.4686192468619247 | 0.4686192468619247 |
| 9 | 0.6745699952221691 | 0.4907226305761031 | 0.48541789275985936 | 0.5146443514644351 | 0.5146443514644351 | 0.5146443514644351 | 0.5146443514644351 |
| 10 | 0.6327125420875421 | 0.4974033816425121 | 0.49075678136476686 | 0.5146443514644351 | 0.5146443514644351 | 0.5146443514644351 | 0.5146443514644351 |
| Average | 0.5816571169352238 | 0.4930392655240016 | 0.46560993895780023 | 0.4920327754532775 | 0.4920327754532775 | 0.4920327754532775 | 0.4920327754532775 |

*Figure 6.4. 10-fold cross validation*

There are some significant observations and trends as follow:

- Despite the variability across folds, the Micro and Macro metrics for Precision, Recall, F1, Accuracy and Average are consistent across folds. This suggests that the model's performance is stable.
- The performance is variety between folds. For instance, in Fold 9, Precision is relatively high (0.674), but in Fold 5, it drops to lowest value (0.529). Similarly, Recall varies from 0.455 to 0.541 and F1 varies from 0.408 to 0.511. This variance occurs due to the differences in the distribution of classes of each fold.

## 7. Updates from part 2

### 7.1 Objective:

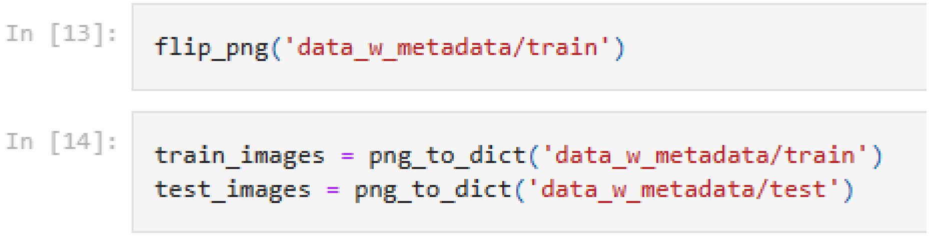
Following the project demonstration in part 2, finding and fixing the unintended data leakage between the train and test set before moving forward to the stage of bias analysis.

### 7.2 Identify Issues:

As recommended by the TA and carefully reviewing the training model pipeline, there is unwanted data leakage happens. Initially, we are aiming at enrich our dataset by some technique to augmentate dataset such as flipping, rotate and brightnenss adjust images. The data augmentation is done before splitting train and test sets, and this leads to the data memorizing in the model. As a result, the metrics are skeptical high.

### 7.3 Improve Explanation:

With the well stated issue, there is a solution to mitigate the issue which is only doing the data augmentation after splitting the train and test sets. (The following implementation is at github: CNN-EmotionDetector/my_notebook.jpynb)

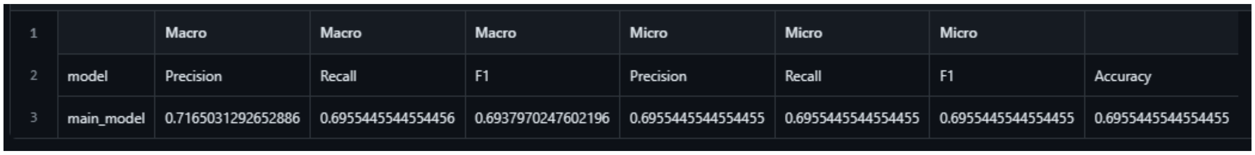


```
In [13]:  flip_png('data_w_metadata/train')

In [14]:  train_images = png_to_dict('data_w_metadata/train')
          test_images = png_to_dict('data_w_metadata/test')
```

*Figure 7.1. Splitting train and test stes are done before augmentation.*

### 7.4 Validate Results:

The following is the metrics frm main model with 3 Conv Layers after fixing the data leakage.



| | Macro | Macro | Macro | Micro | Micro | Micro | |
|---|---|---|---|---|---|---|---|
| model | Precision | Recall | F1 | Precision | Recall | F1 | Accuracy |
| main_model | 0.7165031292652886 | 0.6955445544554456 | 0.6937970247602196 | 0.6955445544554455 | 0.6955445544554455 | 0.6955445544554455 | 0.6955445544554455 |

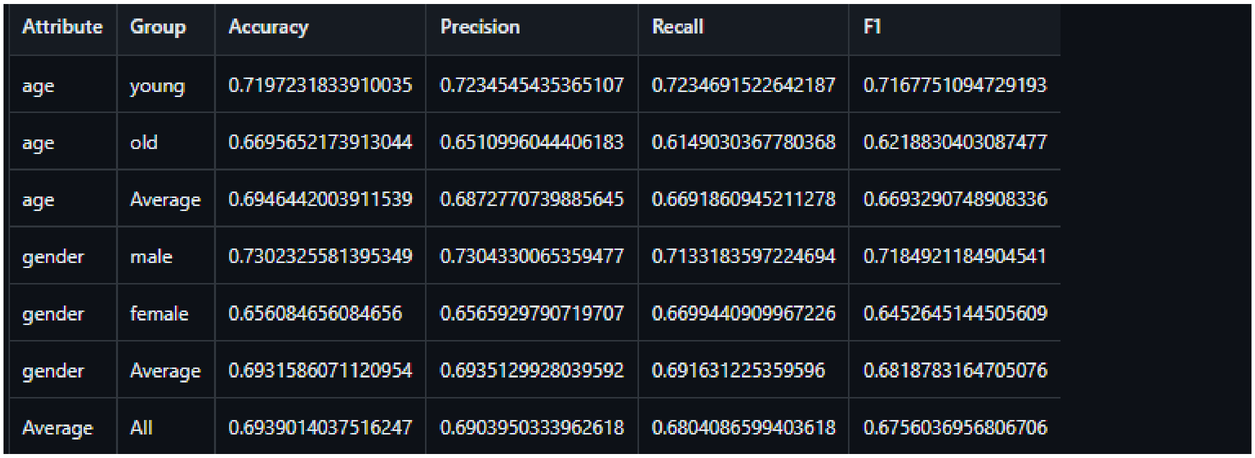
*Figure 7.2. Main model’s performance after data leakage fixing*

# 8. Bias Analysis

## 8.1. Introduction

In our analysis, we investigated biases in our CNN model which is tasked to detect emotions in images. We specifically targeted potential disparities in our model performance based on age and gender attributes. We categorized those bias attributes into distinct groups, “young” and “old” for age, and “male” and “female” for gender. Our approach to analyze any potential bias was to evaluate the accuracy, precision, recall and F1 score for each group.

## 8.2. Bias Detection Results



| Attribute | Group | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|---|
| age | young | 0.7197231833910035 | 0.7234545435365107 | 0.7234691522642187 | 0.7167751094729193 |
| age | old | 0.6695652173913044 | 0.6510996044406183 | 0.6149030367780368 | 0.6218830403087477 |
| age | Average | 0.6946442003911539 | 0.6872770739885645 | 0.6691860945211278 | 0.6693290748908336 |
| gender | male | 0.7302325581395349 | 0.7304330065359477 | 0.7133183597224694 | 0.7184921184904541 |
| gender | female | 0.656084656084656 | 0.6565929790719707 | 0.6699440909967226 | 0.6452645144505609 |
| gender | Average | 0.6931586071120954 | 0.6935129928039592 | 0.691631225359596 | 0.6818783164705076 |
| Average | All | 0.6939014037516247 | 0.6903950333962618 | 0.6804086599403618 | 0.6756036956806706 |

*Figure 8.1. Bias Analysis Table*

The bias detection results are compiled in the table above. We can observe a notable difference in model performance across the bias attributes groups. For the age attribute, the young group outperformed the old group. The young group have higher accuracy (0.719 vs. 0.669), precision (0.723 vs. 0.651), recall (0.723 vs. 0.614), and F1 score (0.717 vs. 0.628). The disparity in performance between the two groups shows signs of age bias. Our model tends to predict more accurately for younger individuals. The gender assessment shows a similar difference in performance favoring males over females. The male group had higher scores in accuracy (0.730 vs. 0.656), precision (0.730 vs. 0.656), recall (0.713 vs. 0.669), and F1 score (0.718 vs. 0.645). The average scores for all bias attributes (age and gender) were calculated and they are similar with barely any difference.

### 8.3. Bias Mitigation Steps

Upon finding these biases, we decided to do some corrective measures. For example, we re-evaluate and rebalance our dataset to make sure that the representation of each age and gender group are proportional.

…

## References

**Dataset Section:**

[1] A. Ananthu, "Emotion Detection FER," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer/data. [Accessed: Feb. 27, 2024]

**Data Cleaning Section:**

[1] "The Role of Data Cleaning in Computer Vision," Data Heroes, 2021. [Online]. Available: https://dataheroes.ai/blog/the-role-of-data-cleaning-in-computer-vision/#:~:text=The%20practice%20of%20discovering%20and,more%20accurate%20and%20understandable%20outputs. [Accessed: Feb. 27, 2024].

[2] ["PyTorch Tutorial: Develop Deep Learning Models." *Machine Learning Mastery*. Available online: https://machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-models/]

[3] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019, doi: 10.1186/s40537-019-0197-0.

[4] H. Talebi and P. Milanfar, "Learning to Resize Images for Computer Vision Tasks," in *Proceedings of the [Conference Name]*, 2021.

**Data Visualization Section:**

[1] C. Zheng and D.-W. Sun, "Image Segmentation Techniques," in *Computer Vision Technology for Food Quality Evaluation*, ed. D.-W. Sun, Academic Press, 2008, pp. 37-56, doi: 10.1016/B978-012373642-0.50005-3.

[2] J. A. Onofrey, D. I. Casetti-Dinescu, A. D. Lauritzen, et al., "Generalizable Multi-site Training and Testing of Deep Neural Networks Using Image Normalization," in *Proceedings of the IEEE International Symposium on Biomedical Imaging (ISBI)*, 2019, pp. 348-351, doi: 10.1109/ISBI.2019.8759295.