COMP472: Artificial Intelligence
Concordia University, Montreal, Quebec

Project "Education MindA.I.lytics"

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

March 11th, 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 3
1. Dataset 4
1.1. Overview 4
1.2. Dataset Justification 5
1.3. Provenance Information: 6
2. Data Cleaning 6
2.1. Data Selection 6
2.1. Image Conversion 7
2.2. Data Augmentation 7
2.3. Image Resizing 8
3. Labeling 9
3.1. Overview 9
3.2. Methods 9
3.3 Ambiguities 10
4. Data Visualization 10
4.1. Class Distribution 10
4.2. Sample Images 11
4.3. Pixel Density Distribution 14
4.4. Further data processing 16
References 18

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	7178	
	disgusted	111		
	fearful	1024		
	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

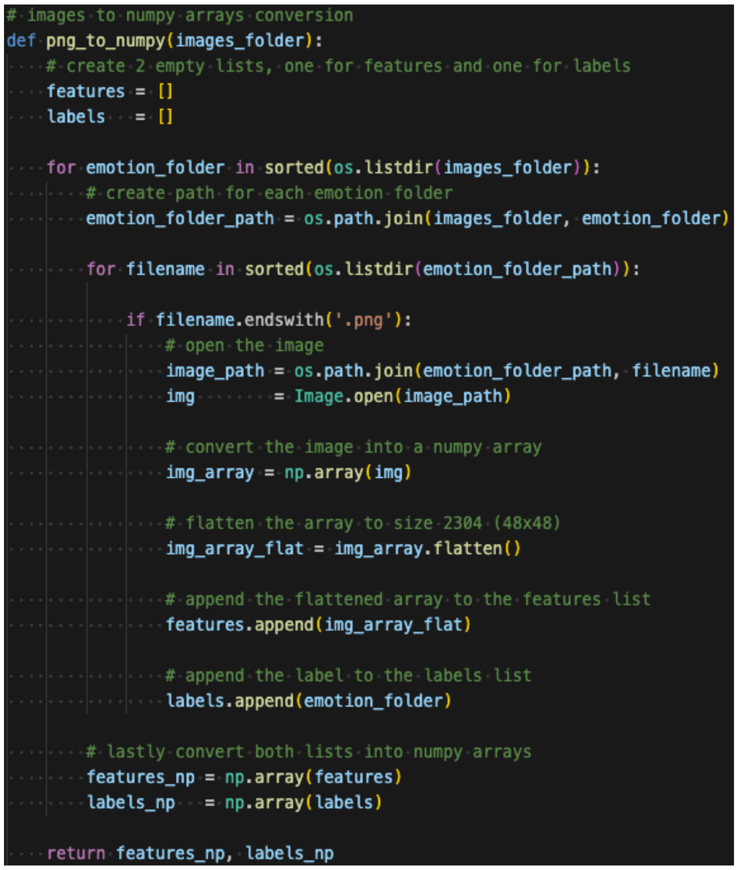
2.1. Data Selection

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].



```
# 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.

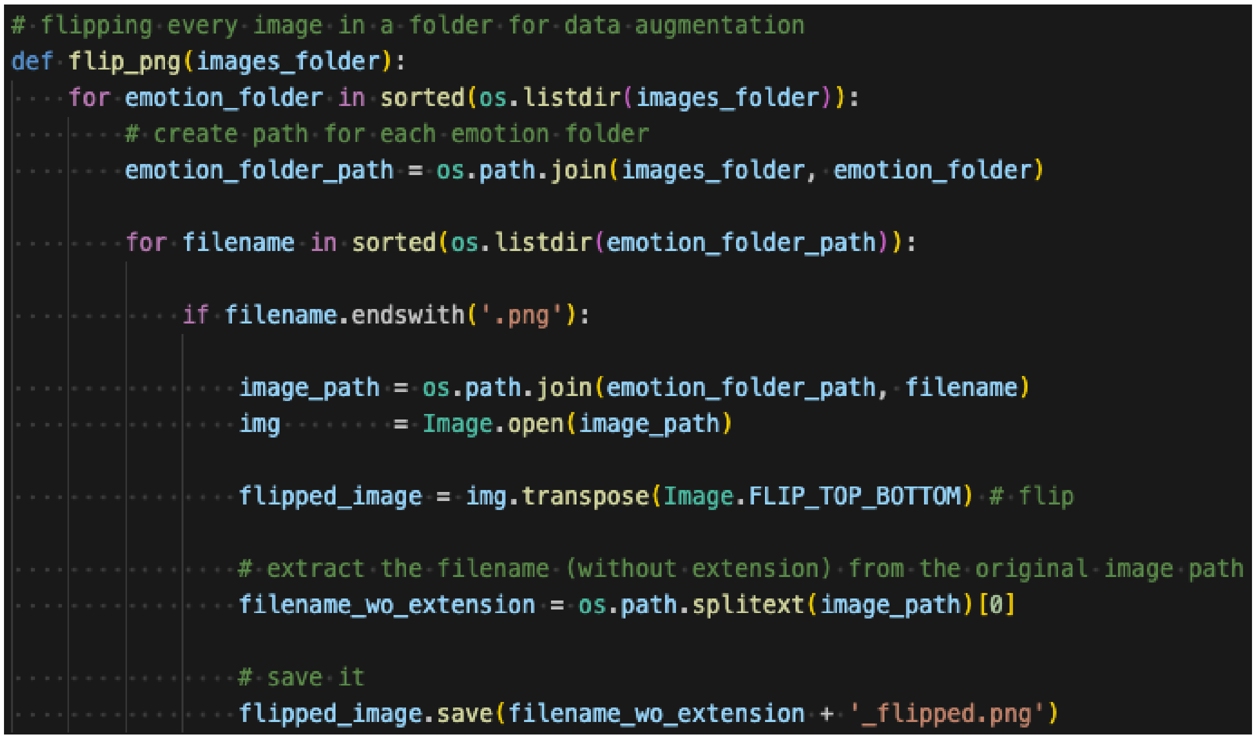

```python
# flipping every image in a folder for data augmentation
def flip_png(images_folder):
    for emotion_folder in sorted(os.listdir(images_folder)):
        # create path for each emotion folder
        emotion_folder_path = os.path.join(images_folder, emotion_folder)

        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.

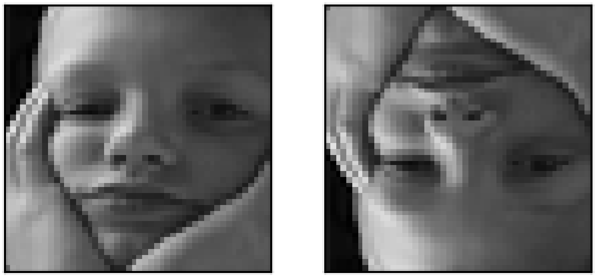


Figure 2.3. Right (original) vs left (flipped)

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).

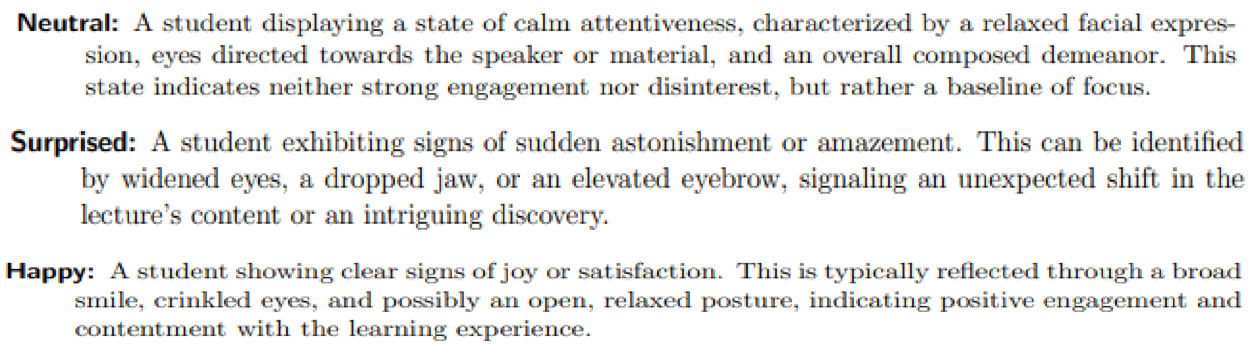
3. Labeling

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:

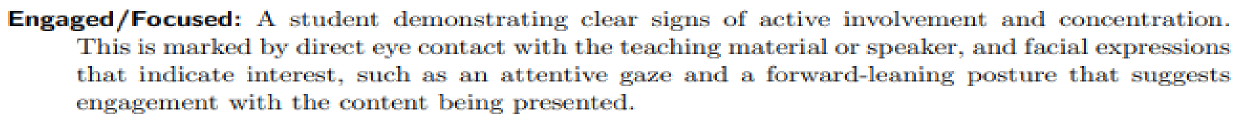

Engaged/Focused: A student demonstrating clear signs of active involvement and concentration. This is marked by direct eye contact with the teaching material or speaker, and facial expressions that indicate interest, such as an attentive gaze and a forward-leaning posture that suggests engagement with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

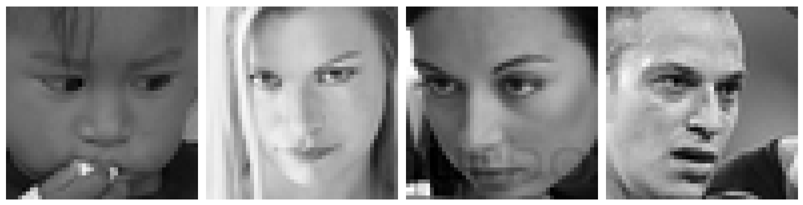


Figure 3.3. Sample images from our Engaged data

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.*

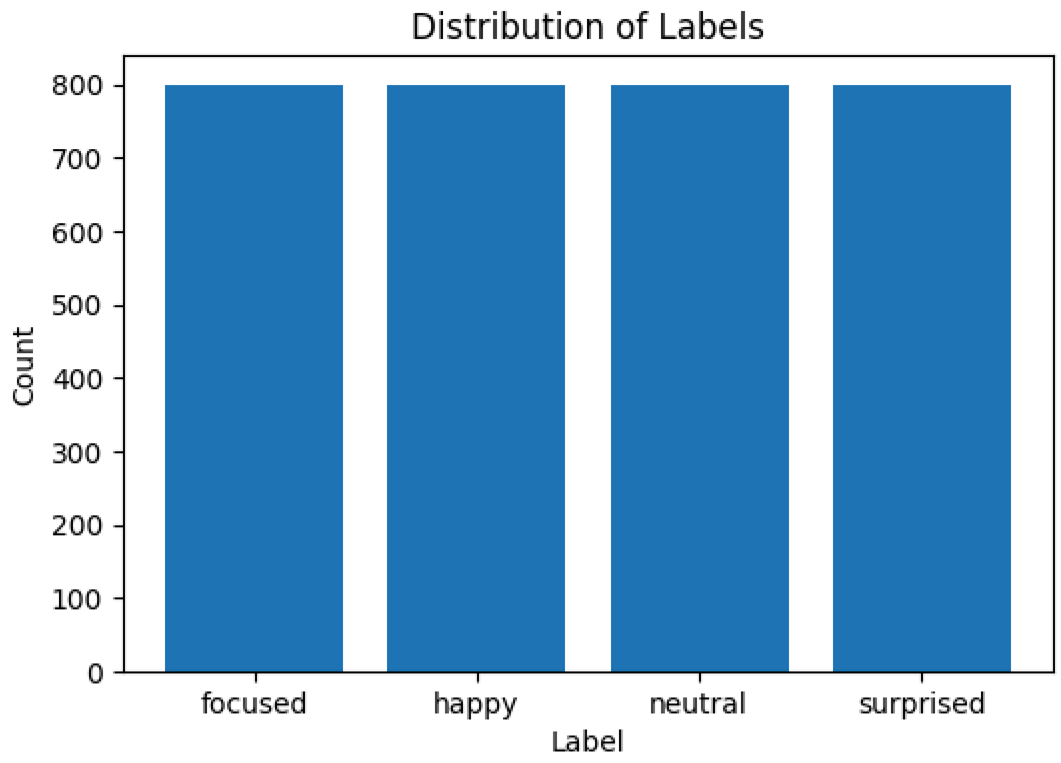
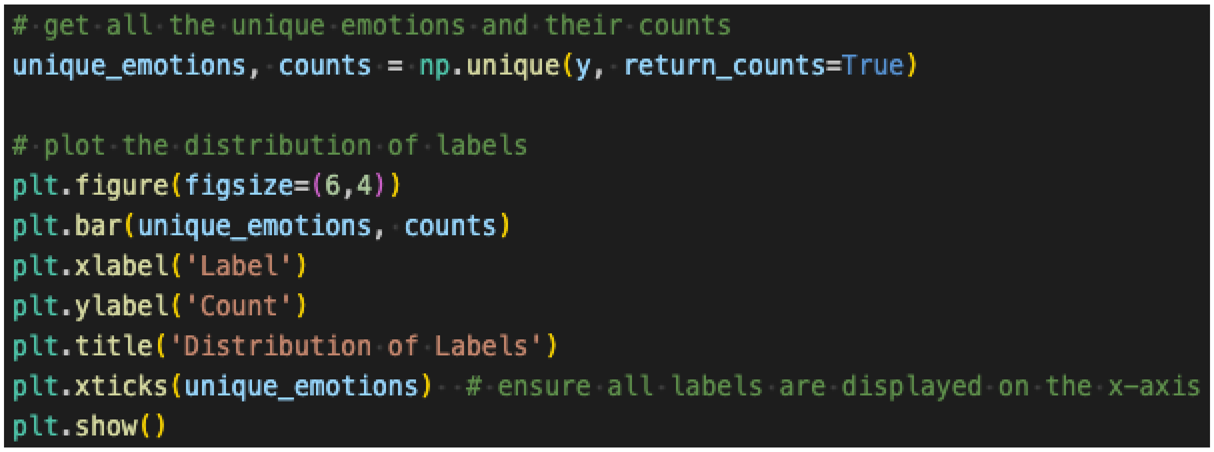


Figure 4.1. Distribution of images per emotion



```python
# 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

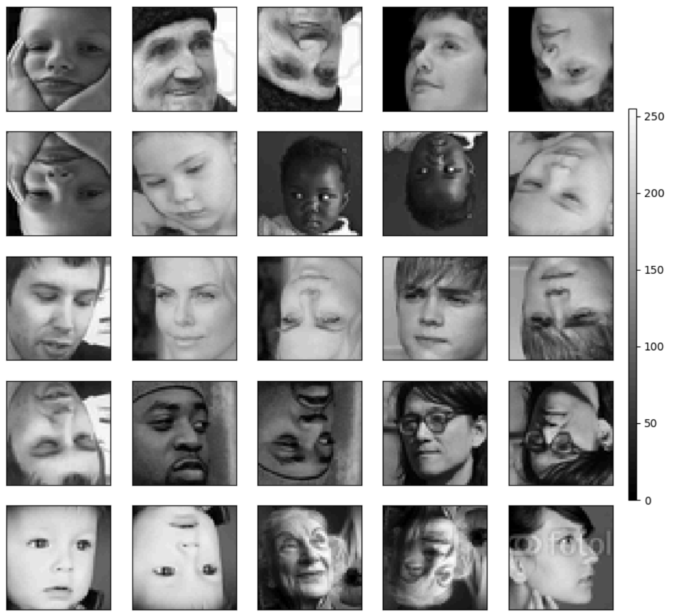


Figure 4.3. 25 “focused” images

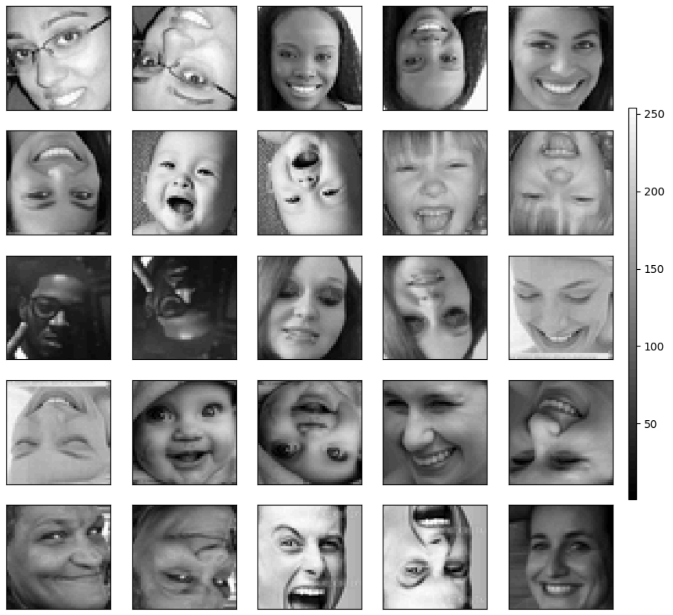


Figure 4.4. 25 “happy” images

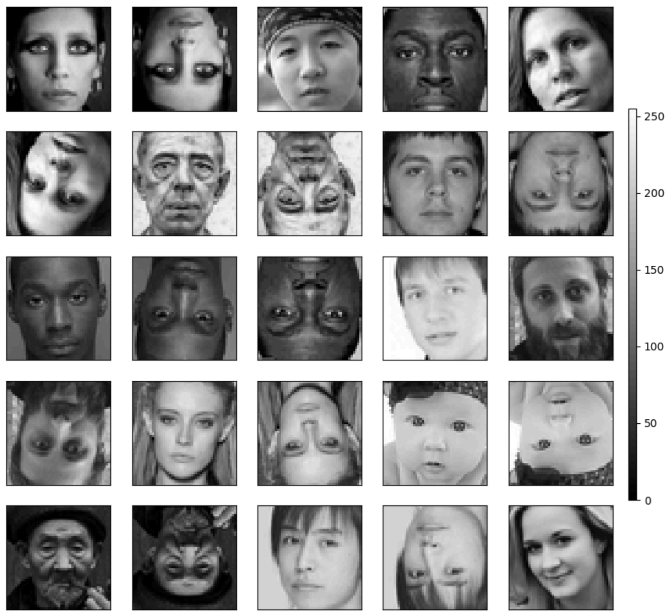


Figure 4.5. 25 "neutral" images

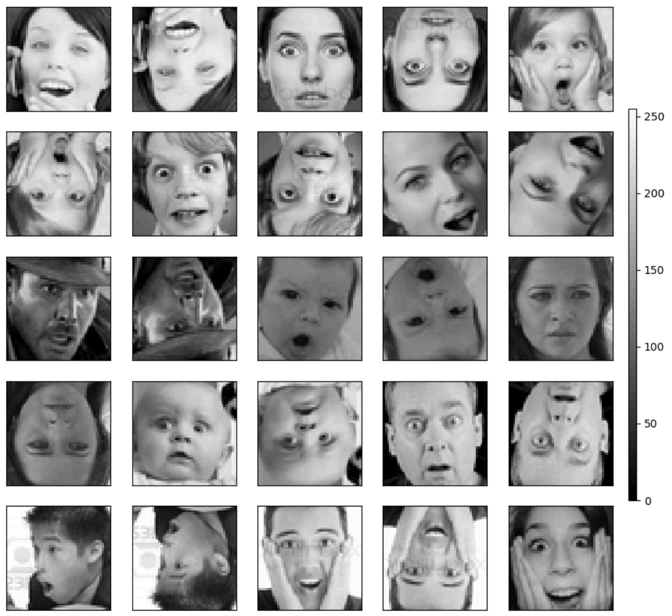
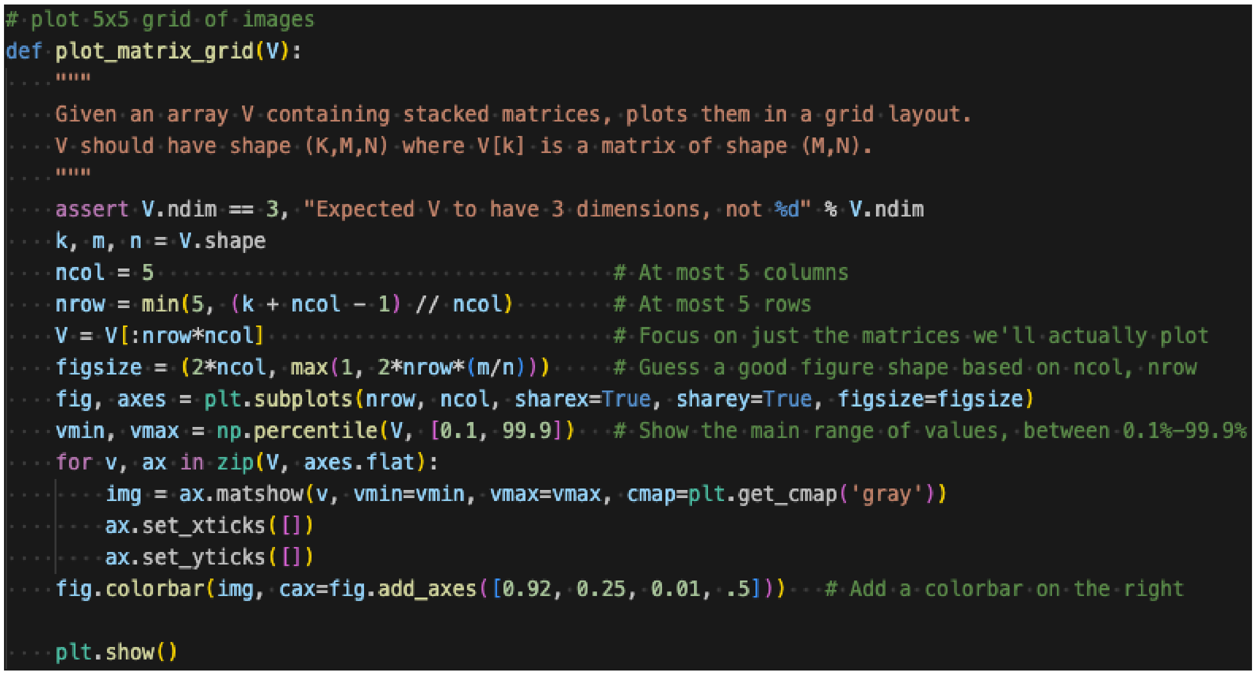
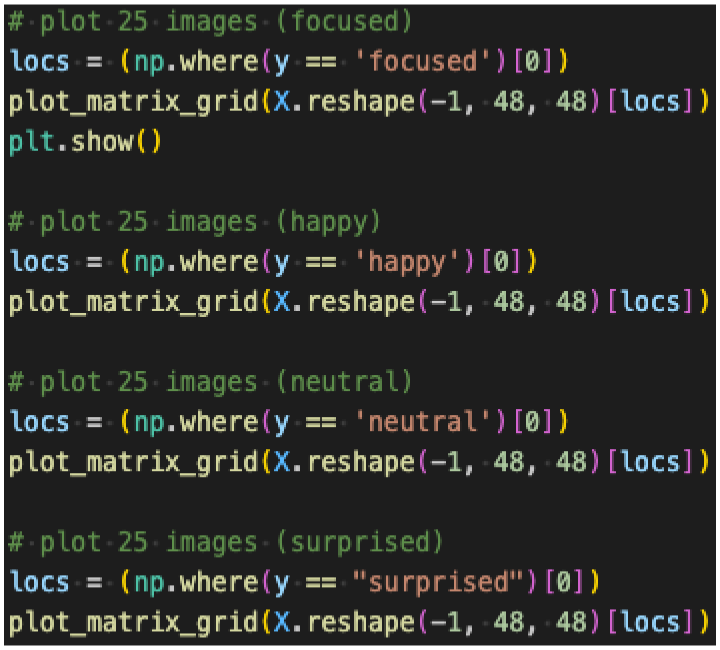


Figure 4.6. 25 "surprised" images



```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].

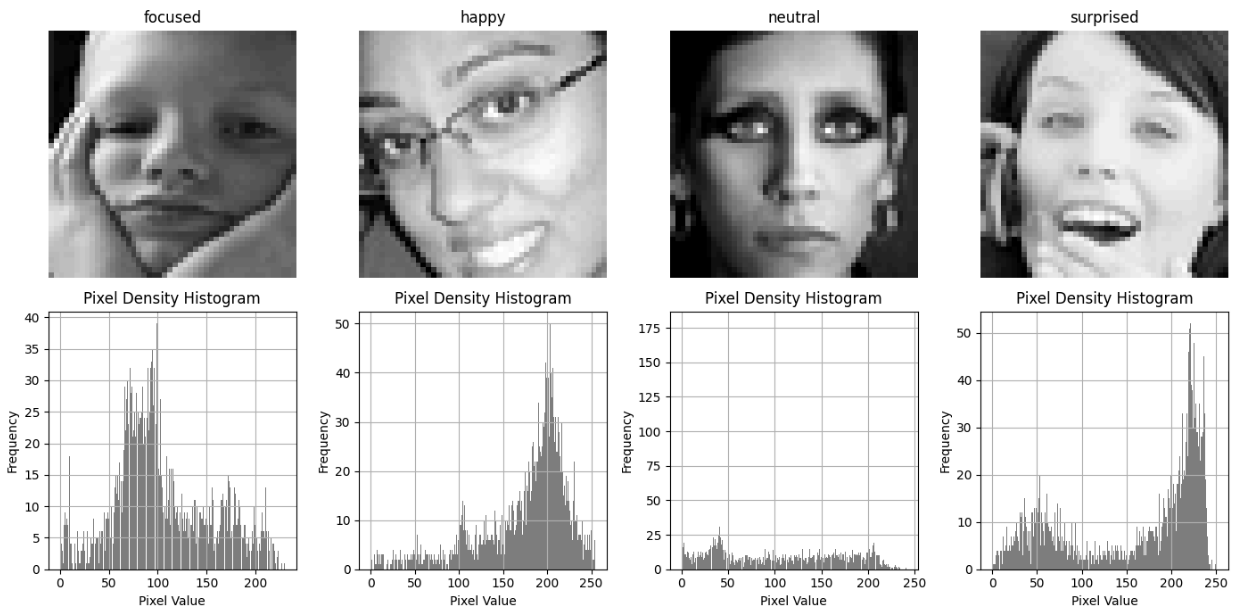
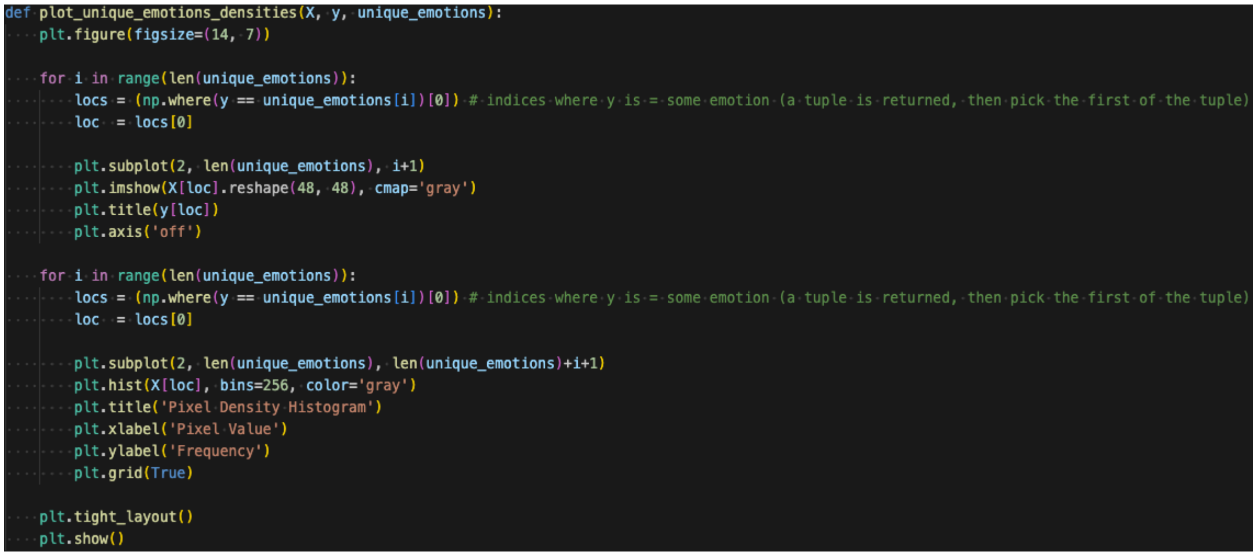


Figure 4.8. Pixel density histogram for 4 different emotions



```python
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].

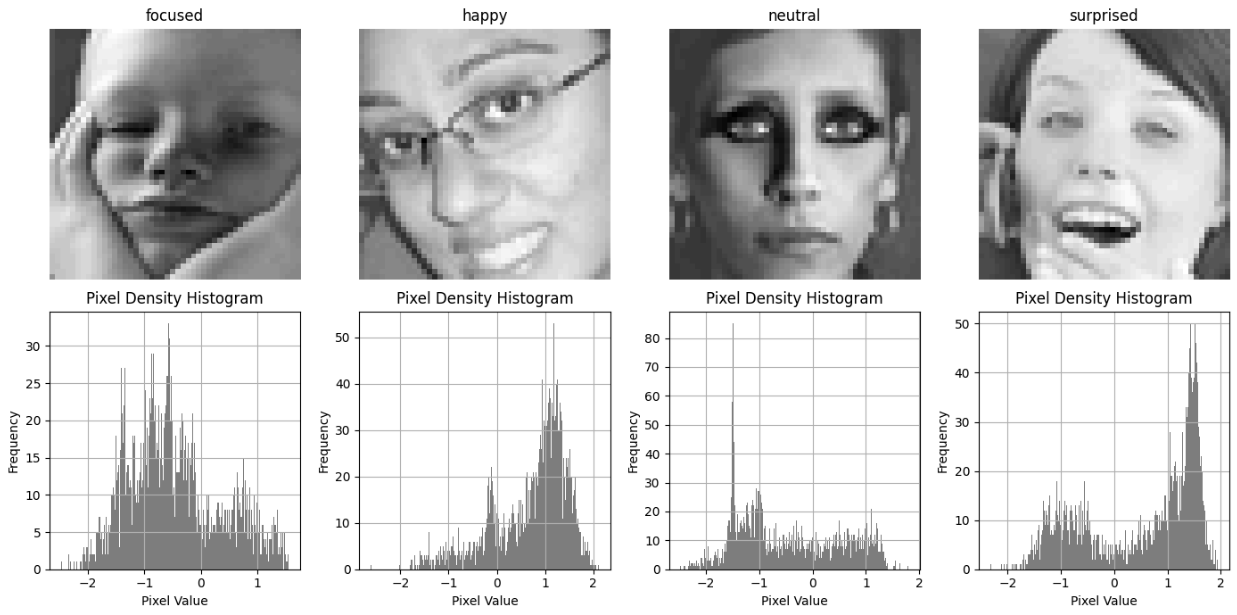
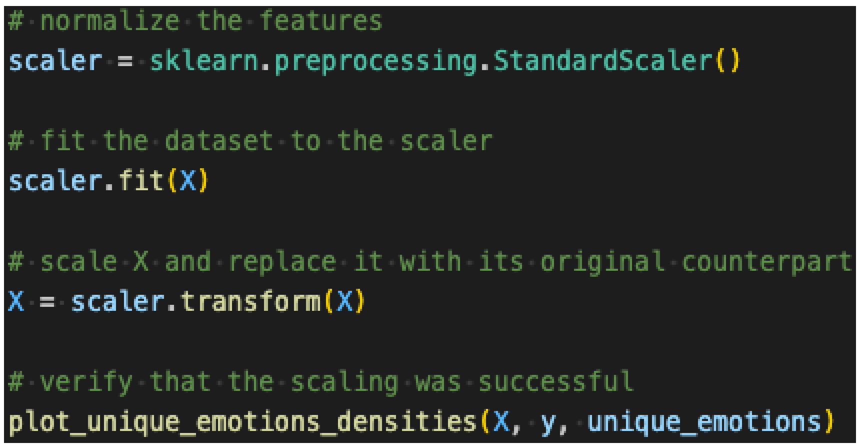


Figure 4.10. Pixel density histogram for 4 different emotions (after normalization)



```
# 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.

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.