# Phase 1 Report: Analysis, Design, and Data Preparation

**Project: Facial Expression Recognition (FER) System**

Course: Artificial Intelligence - K. N. Toosi University of Technology

Instructor: Dr. Pishgoo

Team Members: Soroush Soleimani, Parham Kootzari, Amirhossein Babaee

---

## 1. Problem Definition & Scope

The core objective of this project is to develop a robust Computer Vision pipeline capable of classifying human emotions into eight distinct categories. Facial Expression Recognition (FER) is a pivotal task in Affective Computing, enabling machines to interpret human psychological states. Our system processes grayscale facial images (48x48 pixels) to detect: **Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral, and Contempt.** The addition of the 'Contempt' class allows for a more nuanced understanding of complex social facial signals.

## 2. Dataset Selection & Exploratory Data Analysis (EDA)

While the foundation of our work is the FER-2013 dataset, we specifically utilized the **FERPlus** labels. The original FER-2013 dataset is known to contain significant labeling noise due to the ambiguity of facial expressions.

**FERPlus** addresses this by providing refined labels generated through a crowd-sourcing effort where 10 independent taggers evaluated each image. This results in a much cleaner ground truth, which is essential for reaching high accuracy (~90%) in later phases. We conducted an extensive EDA to understand the data's underlying patterns before model design.

**Data Preparation & Cleaning Pipeline:**

Our pipeline (implemented in **src/data_preparation.py**) handles the raw-to-cleaned transition:

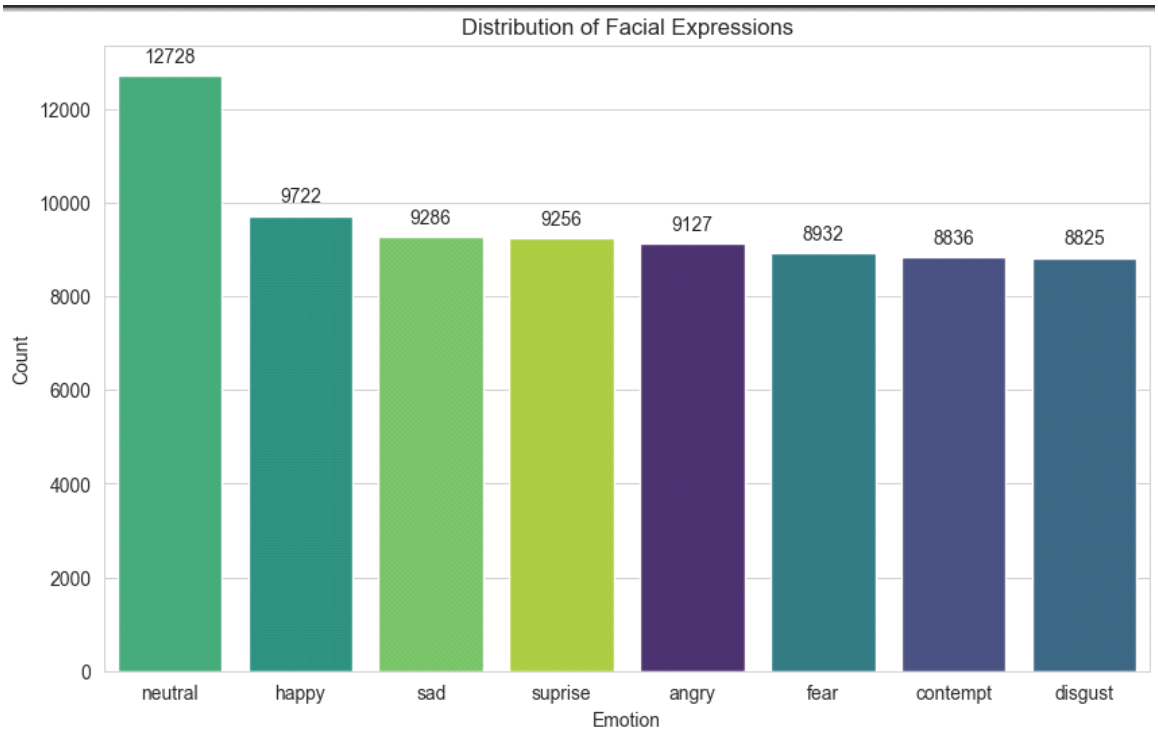
- **Noise Filtering:** We intentionally simulated and then removed "dirty data" (invalid IDs and corrupted labels) to ensure a robust cleaning process.
- **Dataset Flattening:** The complex FERPlus directory structure was flattened into a unified **data/raw** folder.
- **Final Ground Truth:** A **dataset_cleaned.csv** was generated after verifying image integrity, ensuring no missing pixel data enters the training phase.

### 2.1. Class Imbalance Analysis

Our analysis (as shown in the chart below) revealed a significant variance in the number of samples per class. For instance, the 'Happy' and 'Neutral' classes have a high density, while **'Disgust' and 'Contempt'** are severely underrepresented.

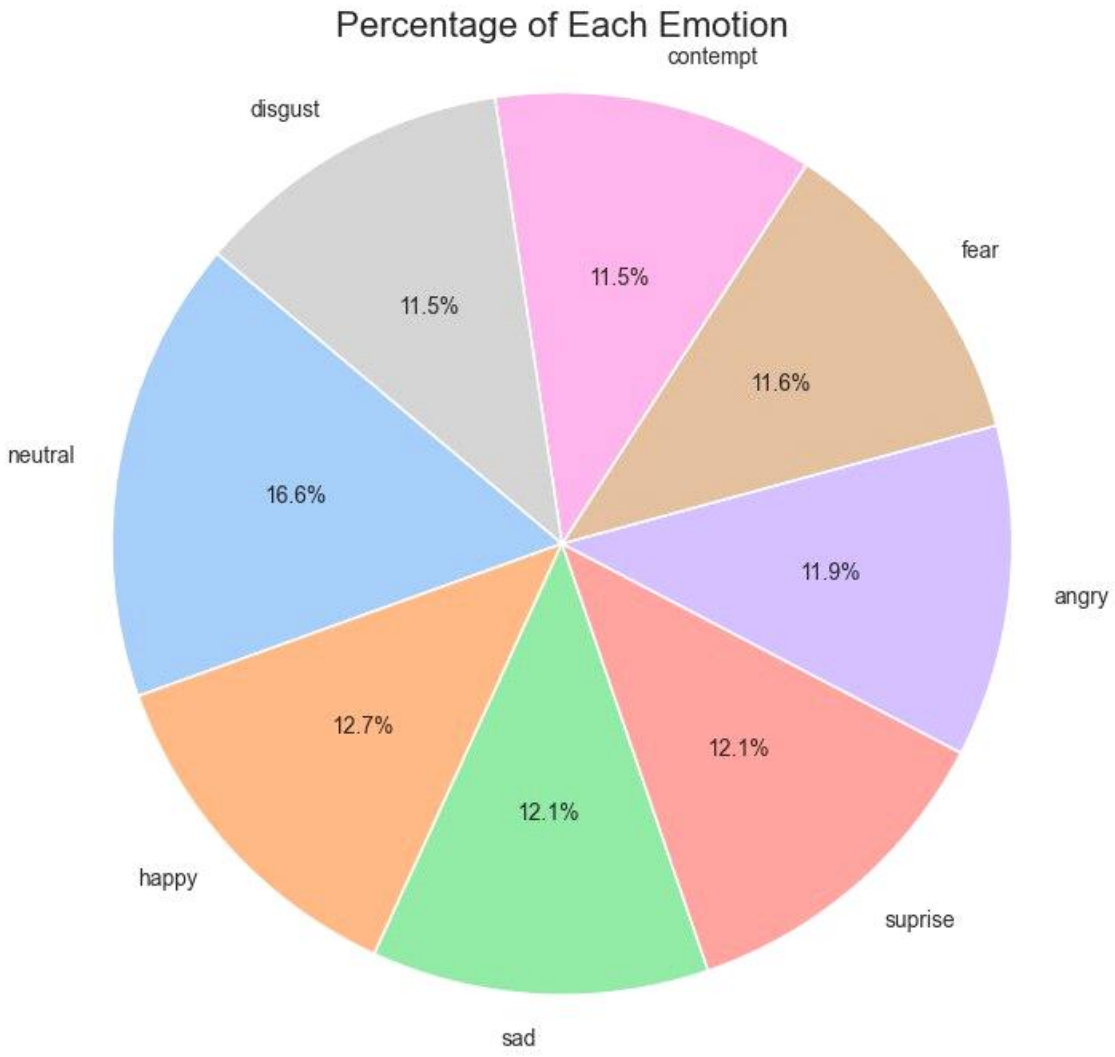
**Challenge:** This imbalance poses a risk of the model becoming biased toward majority classes.

**Strategy:** To mitigate this, we decided to implement **Heavy Data Augmentation** for minority classes and utilized **Categorical Cross-Entropy** as our loss function to ensure the model learns features from all emotional categories equally.



### 2.2. Percentage of each emotion

This code counts the occurrences of each emotion and calculates their percentages to check for class imbalance. Its purpose is to ensure the dataset provides enough examples for every category so the model doesn't become biased toward a single expression.

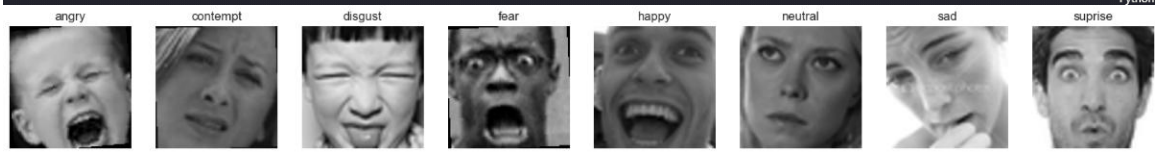


## 2.3. Visualizing Sample Images

The main objective of this code is visual verification. Before training any AI model, you must confirm that the images are loading correctly and that the labels match the content (e.g., ensuring a file labeled "happy" actually shows a smiling face).

```python
IMAGE_DIR = "../data/raw"
unique_labels = df['label'].unique()

fig, axes = plt.subplots(1, len(unique_labels), figsize=(20, 5))

for i, label in enumerate(unique_labels):
    sample_row = df[df['label'] == label].sample(1).iloc[0]
    image_name = sample_row['filename']
    image_path = os.path.join(IMAGE_DIR, image_name)

    try:
        img = cv2.imread(image_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        axes[i].imshow(img)
        axes[i].set_title(label)
        axes[i].axis('off')
    except Exception:
        axes[i].text(0.5, 0.5, "Not Found", ha='center')
        axes[i].axis('off')

plt.show()
```

Python



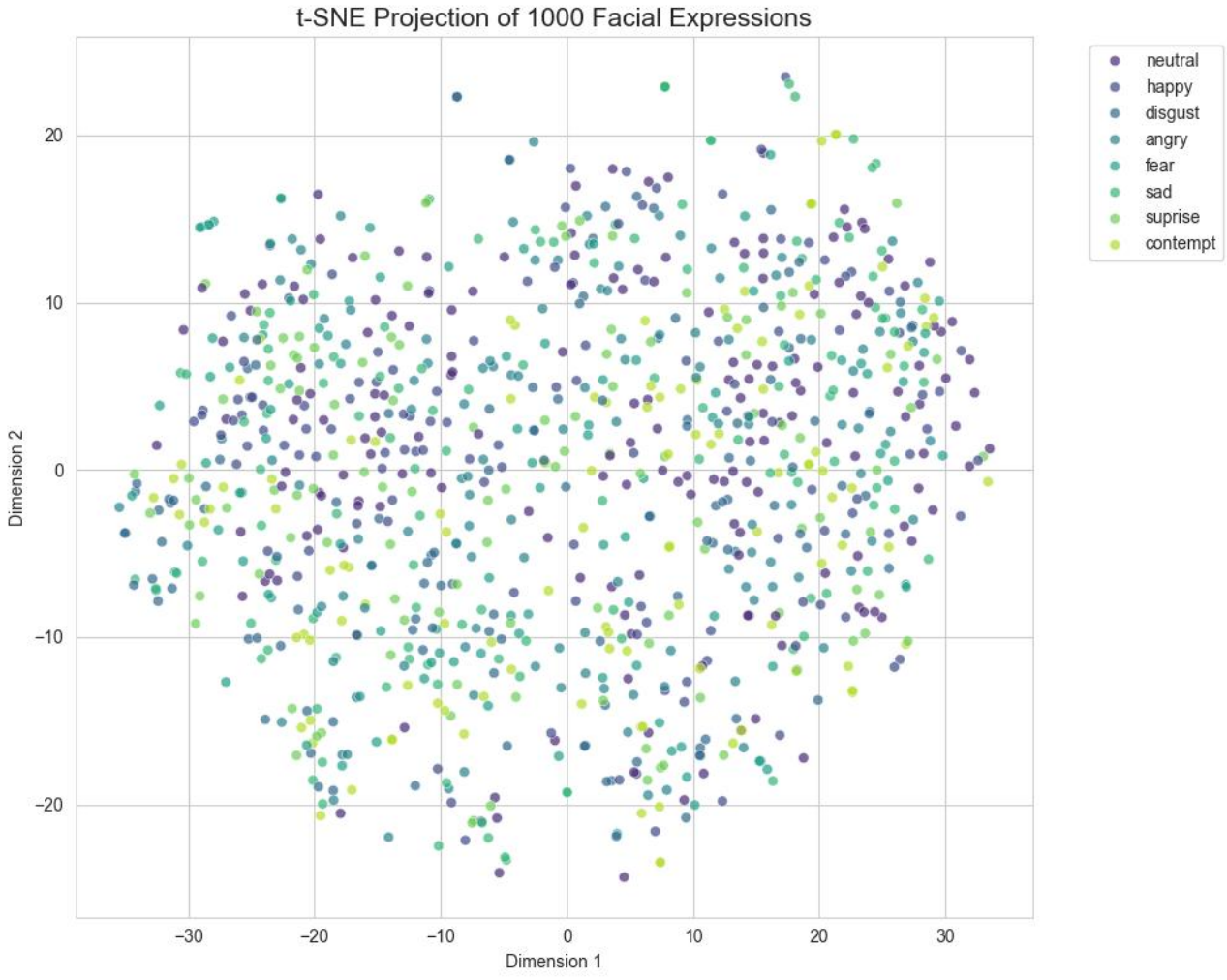

## 2.4. Visual Data Inspection

To understand the noise levels and variations in head poses/lighting, we visualized samples from each class. This confirmed that the dataset contains real-world challenges like occlusion and low-resolution (48x48) artifacts.

## 2.5. Visualizing Feature Separability with t-SNE

**Purpose**

The goal is to evaluate how well different emotions can be separated based on raw pixel values. A subset of images is analyzed, converted to grayscale, resized to a uniform resolution, and flattened into feature vectors. These high-dimensional pixel features are then reduced to two dimensions using t-SNE for visualization. This helps determine whether certain emotions (such as "Angry" and "Disgust") appear similar to the model, which may make classification more challenging.



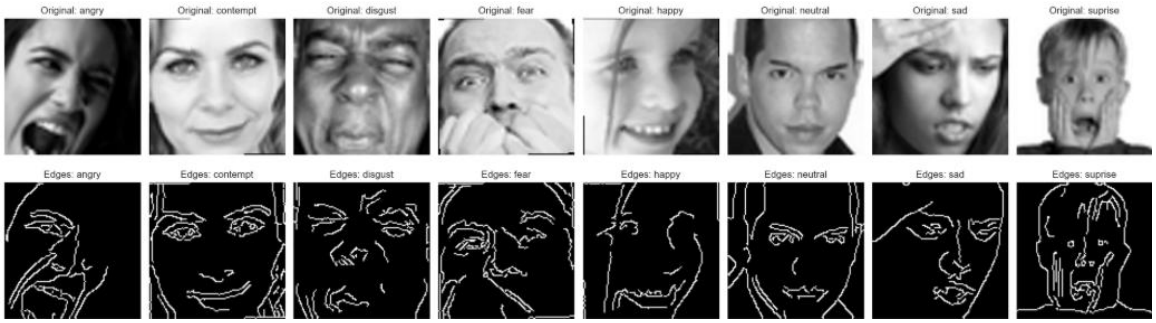
**Conclusions from the Output**

- **Cluster Formation:**
  If clear clusters of the same color appear, it indicates that raw pixel values can meaningfully distinguish between those emotions.
- **Overlap Analysis:**
  If there is noticeable overlap between emotion classes, it suggests that raw pixel information alone is not sufficient, and a more advanced model (such as a CNN) may be necessary.
- **Data Quality and Structure:**
  If the points are completely mixed with no clear structure, it may indicate high noise in the dataset or very subtle differences between emotion categories.

## 2.6. Edge Detection

### Purpose

This section focuses on understanding feature extraction in facial expression recognition. One representative image from each emotion category is selected and displayed in two forms: the original grayscale image and its edge-detected version. Edge detection highlights sharp changes in intensity, emphasizing important facial contours such as the eyes, eyebrows, and mouth. By visualizing these structural features, we can verify whether the essential expression-related information is preserved while reducing background noise and lighting effects.




### Conclusions from the Output

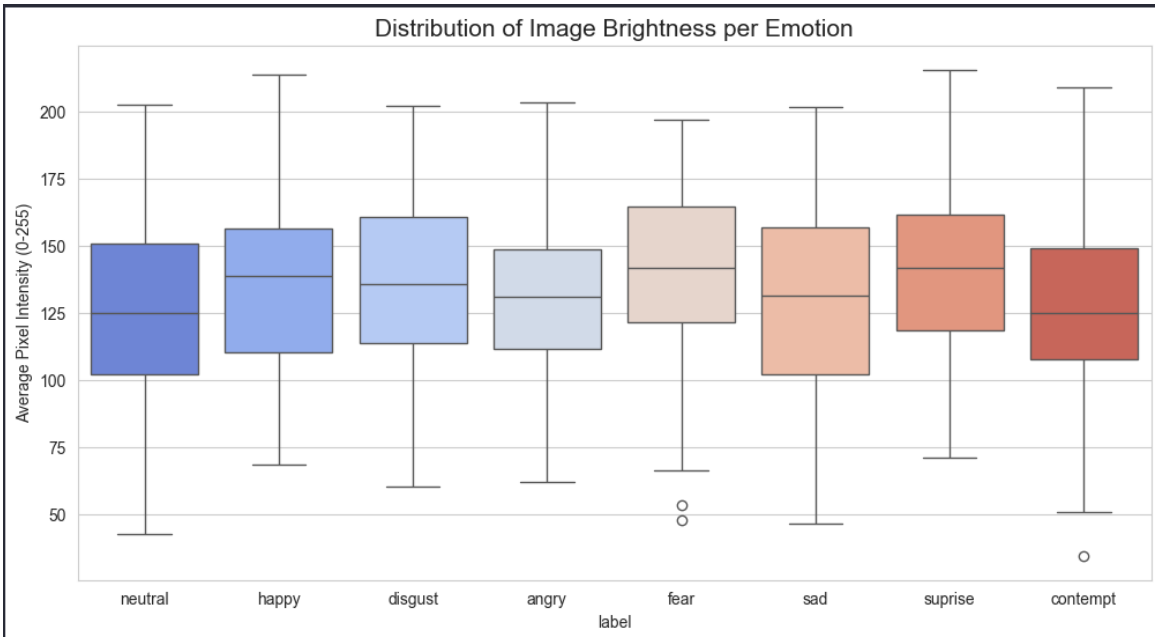
- **Feature Prominence:**
  If the contours of key facial regions (such as lips and eyes) appear sharp and well-defined, it indicates that the dataset contains strong structural features suitable for training a Convolutional Neural Network (CNN).
- **Noise Assessment:**
  If the edge-detected images contain excessive random lines or background artifacts, additional preprocessing such as denoising or tighter face cropping may be required.
- **Overall Conclusion:**
  The results demonstrate that the defining characteristics of each emotion (for example, the curve of a smile or the tension in a brow) are structurally detectable, suggesting that the model can effectively learn these patterns during training.

### 2.7. Image Brightness & Quality Assessment

The box plot analysis of average pixel intensities across different classes (Figure X) shows a relatively consistent distribution. Most emotions have a median brightness centered around 125-150.

## Purpose

The goal is to determine if the lighting conditions are consistent across different emotion categories. In a robust dataset, "Happy" images shouldn't be significantly brighter than "Sad" images just because of the camera settings. If one category is much darker than others, the model might accidentally learn to associate "darkness" with that emotion rather than the actual facial features.




**Conclusion:** This consistency indicates that the dataset is well-balanced in terms of lighting conditions across different emotions. Consequently, the model is expected to focus on **facial morphology and landmarks** (like mouth and eye shapes) rather than being misled by environmental lighting or exposure differences. To further standardize this, a Global Normalization **[0, 1]** was applied during preprocessing.

## 3. Data Preprocessing & Engineering

To transform raw pixels into a format suitable for Deep Learning, we implemented the following pipeline:

**Normalization**: Pixel intensities were rescaled from [0, 255] to [0, 1] to optimize gradient descent.

**Feature Cleaning**: We addressed missing values and ensured all images followed the (48, 48, 1) tensor shape.

**Label Transformation**: Categorical labels were One-Hot encoded to facilitate multi-class cross-entropy calculation.

## 4. Advanced Data Augmentation

To prevent the model from memorizing the training set (Overfitting), we implemented a real-time augmentation layer. This effectively increases the dataset's diversity by simulating different camera angles and lighting conditions:

**Geometric Transforms**: Random rotations (15°), width/height shifts, and horizontal flips.

**Intensity Transforms**: Random zoom and brightness adjustments.

## 5. Architectural Design of the Baseline Model

Our baseline (baseline_model.py) is a CNN that captures spatial hierarchies.

**Activation:** We use ReLU for hidden layers and Softmax for the 8-class output.

**Optimization:** The Adam optimizer was chosen for its adaptive learning rate, and Categorical Cross-Entropy is used to measure the loss between the 8 emotion probabilities.

Following the modularity requirements of the course, we designed a Convolutional Neural Network (CNN). The architecture consists of:

**Convolutional Blocks**: For hierarchical spatial feature extraction.

**Batch Normalization**: To stabilize the learning process.

**Dropout Layers**: To enhance the model's generalization capability.

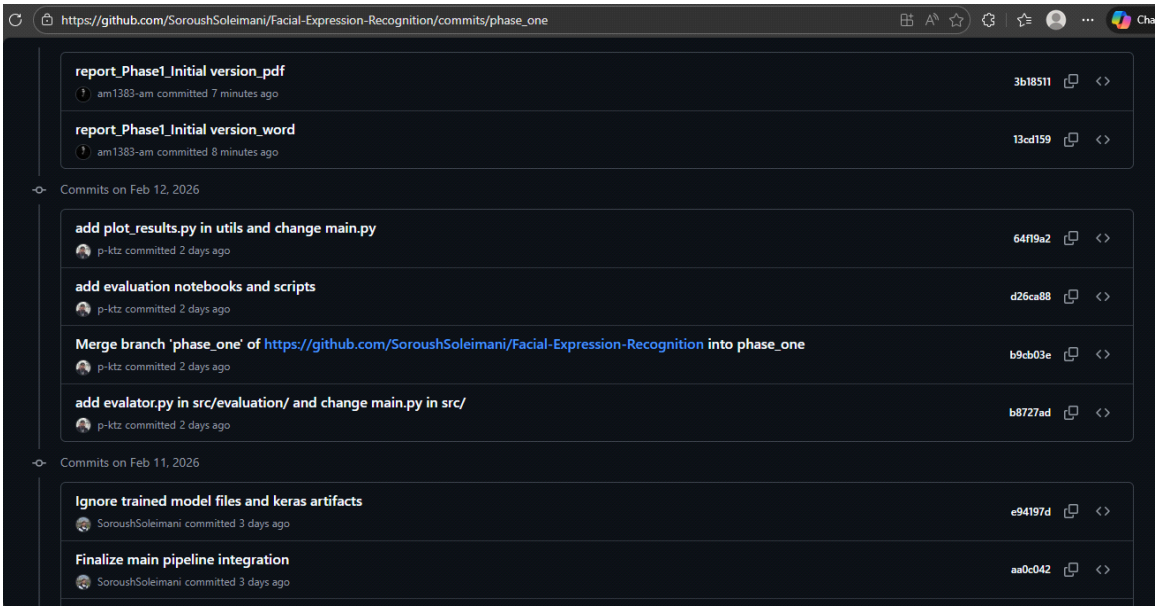
"The project follows a **Modular Design**. Source code is separated from data and notebooks, ensuring that the training pipeline (src/training/train.py) is independent of the model definitions."

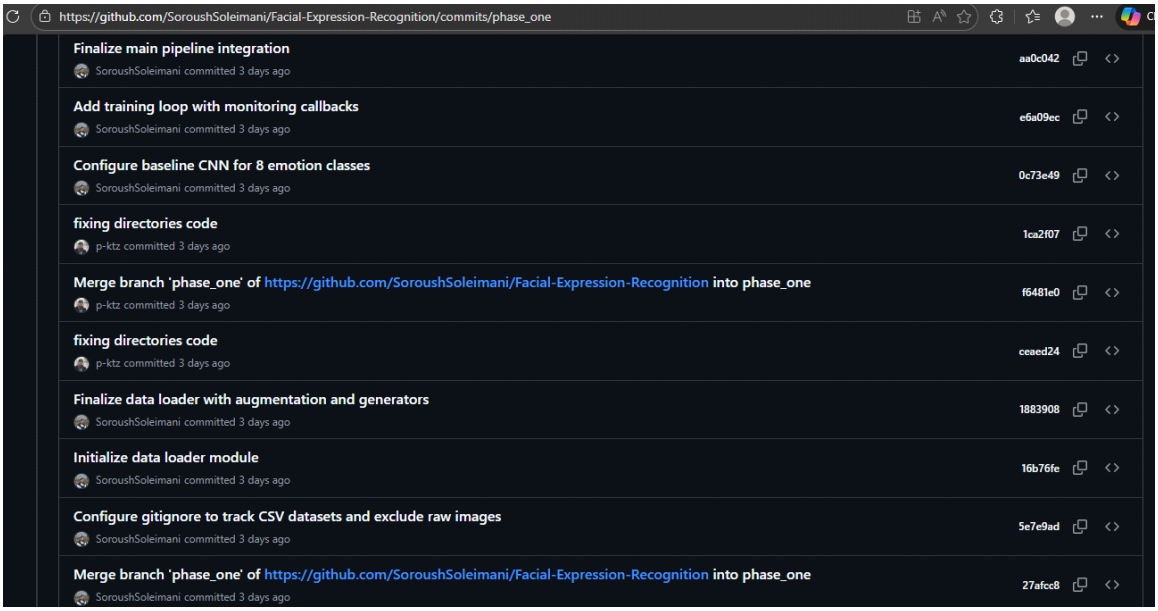
```
├── data/
│   └── raw/                    # Contains dataset and CSV files (Content ignored by .gitignore)
├── models/                     # Stores trained .keras models and history logs (Content ignored by .gitignore)
├── notebooks/                  # Jupyter notebooks for step-by-step analysis
│   ├── data_preparation.ipynb
│   ├── EDA.ipynb
│   ├── evaluation.ipynb
│   └── Visualizing during training.ipynb
├── results/                    # Generated plots, metrics, training figures, and reports
├── src/                        # Core source code modules
│   ├── data_preparation/       # Scripts for data cleaning and preparation
│   │   └── data_preparation.py
│   ├── evaluation/             # Evaluation classes and metric calculations
│   │   └── evaluator.py
│   ├── models/                 # Deep Learning architecture definitions
│   │   ├── baseline_model.py
│   │   └── final_model.py
│   ├── preprocessing/          # Data loaders and augmentation logic
│   │   └── data_loader.py
│   ├── training/               # Training loop implementation
│   │   └── train.py
│   ├── utils/                  # Helper functions (e.g., plotting results)
│   │   └── plot_results.py
│   └── training_pipeline.py    # Orchestrator script to run the full pipeline
├── app.py                      # Streamlit web application (Demo)
├── main.py                     # Main entry point for console execution
├── requirements.txt            # Project dependencies
└── .gitignore                  # Git configuration
```

It should be noted that the project structure has been established, but not all modules and files have been fully implemented yet.

## 6. GitHub Integration & Version Control

As part of our professional workflow, the project is maintained on GitHub with a modular structure. We used Git for collaborative development, ensuring all changes in preprocessing and model design are tracked through meaningful commits.



### 6.1. Team Roles & Responsibilities

To ensure maximum efficiency and high-quality results, the responsibilities for Phase 1 were distributed based on the core pillars of an AI project:

**Soroush Soleimani (Lead Developer):**

Designed and implemented the core **Preprocessing and Data Augmentation** pipeline.

Developed the **Baseline Model architecture** and integration logic in the src directory.

Orchestrated the modular structure of the project to ensure code reusability.

**Parham Kootzari (Data Scientist):**

Conducted the **Exploratory Data Analysis (EDA)** and statistical assessments of the FERPlus dataset.

Generated class distribution visualizations and performed **Brightness & Quality analysis.**

Managed the data labeling transition from FER-2013 to the refined FERPlus format.

**Amirhossein Babaee (DevOps & Documentation):**

Automated the development environment using **Makefile** and managed dependency configuration (requirements.txt).

Supervised **Version Control (Git)** and maintained the GitHub repository structure.

Authored the technical reports and maintained the comprehensive project **README.**


p-ktz
29 commits 424,030 ++ 318,767 --
#1
Contributions
20
10
0
29 Dec
12 Jan
26 Jan
9 Feb
am1383-am
26 commits 532 ++ 107 --
#2
Contributions
20
10
0
29 Dec
12 Jan
26 Jan
9 Feb
SoroushSoleimani
21 commits 233,949 ++ 155,144 --
#3
Contributions
20
10
0
29 Dec
12 Jan
26 Jan
9 Feb