Below is a **comprehensive Part 2 document** that follows your requested **structure** and **format**. It encapsulates all relevant details from our conversation and the scripts you have developed (baseline, softmax/logistic, fully connected network, advanced CNN, and live inference). Adapt **file names** or **paths** as needed for your specific project folder structure.

# **Facial Expression Recognition Project**

**Date:** *[Insert Current Date]*

## **1. Introduction**

### **1.1 Project Goals**

The primary objective of this project is to **accurately classify facial expressions** (emotions) from images. This classification spans multiple emotion categories, such as *angry, disgust, fear, happy, neutral, sad, surprise*. The project aims to:

* **Establish a baseline** and demonstrate improvement through more advanced methods.
* **Compare** simpler approaches (baseline, softmax/logistic) to deeper neural network architectures (fully connected NN and an advanced CNN).
* **Provide** a real-time detection and classification demo (live inference) that applies the best model to webcam input.

### **1.2 Data Review**

* **Features**: 48×48 grayscale pixel values for each face (numerical).
* **Labels**: One of **seven** emotion categories.
* **Task**: **Multiclass Classification** (7 distinct classes).
* **Splits**:
  + **Train**: used to learn model parameters.
  + **Validation**: used to monitor overfitting/tune hyperparameters.
  + **Test**: optional final evaluation.

Typical data distribution involves **70%** training, **15%** validation, **15%** testing, although exact ratios vary depending on the dataset. The data is saved in .pt files (train\_data.pt, val\_data.pt, etc.) for efficient PyTorch usage.

### **1.3 Problem Definition**

1. **Classification**: The goal is to classify a given 48×48 image into one of the 7 emotions.
2. **Evaluation**: Compare various metrics (accuracy, precision, recall) across multiple models.
3. **Context**: The classification can be used in real-time or on static images.

## **2. Main Content**

Below are the main methods tried, from simplest baseline to an advanced CNN, with **sample results**.

### **2.1 Baseline Model**

#### **Description**

* **Basic Idea**: Always predict the single most frequent class in the training data (e.g., "happy").
* **Implementation**:
  + Identify the most frequent emotion from the training set distribution.
  + Predict that emotion for all validation/test samples.
* **Evaluation**:
  + Typically yields around **25%** accuracy if "happy" is the largest subset.

#### **Results**

* An example run showed an **Accuracy** of about **25–26%**.
* Other classes have zero precision/recall, except the predicted frequent one (leading to partial recall but minimal overall performance).

**Comparison**:

* Serves as a **minimum** to beat.
* No learning occurs; it's a naive reference.

### **2.2 Linear/Logistic Regression or Softmax**

#### **Description**

* **Softmax** logistic regression:
  1. Flattens each 48×48 = 2304 pixel values into a single vector.
  2. Applies a linear transformation plus softmax to get probabilities across 7 classes.
* **Implementation**:
  1. Use **CrossEntropyLoss** for training on the flattened vectors.
  2. Optimize with SGD or Adam.
  3. Evaluate on validation set.

#### **Results**

* Validation accuracy improved to around **36–38%** (depending on exact runs).
* Some classes (like *happy* or *surprise*) gain better recall, but *disgust* or *fear* remain poorly recognized.
* Overfitting is minimal but capacity is also limited.

**Comparison**:

* Already beats the baseline’s ~25%.
* Underfitting for complex image tasks, so we proceed to deeper networks.

### **2.3 Fully Connected Network (MLP)**

#### **Description**

* A **multi-layer perceptron**:
  1. Input: Flattened 48×48 pixels → fully connected layers (e.g., 128 or 256 hidden units) → output 7 classes.
  2. **Activation**: ReLU
  3. **Dropout** or weight decay to reduce overfitting.
* **Implementation**:
  1. **CrossEntropyLoss** again, with Adam or SGD optimizer.
  2. Evaluate training vs. validation loss for signs of overfitting.

#### **Results**

* Validation accuracy often reaches **42–45%**.
* Some classes (like *disgust*) remain rarely detected. Overfitting can appear if hidden dimensions are large without proper regularization.

**Comparison**:

* Outperforms softmax. The hidden layer(s) can capture more complex non-linearities.
* Still less effective than a CNN for image data due to ignoring spatial structure.

### **2.4 Advanced Model (CNN)**

#### **Description**

* A **convolutional neural network** (3 convolutional layers + max pooling + dropout + 2 FC layers).
* **Techniques**:
  1. **Weighted cross entropy** to address class imbalance (e.g., weights = [1.0, 5.0, ...]).
  2. **Dropout** (p=0.4) to reduce overfitting.
  3. **Early Stopping**: monitor validation loss, restore best weights.
  4. **Learning Rate Scheduler** (StepLR) to refine training in later epochs.

#### **Results**

* Validation accuracy can reach **55–60%** or more, depending on dataset specifics.
* Overfitting is controlled by dropout and early stopping.
* Hardest classes like *fear* or *disgust* still less recognized but improved overall.

**Comparison**:

* Significantly best performer vs. baseline, softmax, and basic MLP.
* Spatial convolution is well-suited for image tasks, capturing local features.

## **3. Improvements**

### **3.1 Initial Attempts and Issues**

1. **Baseline**: Too naive, ~25% accuracy.
2. **Softmax**: Underfitting for complex image data, ~36–38%.
3. **Basic NN**: Overfitting after ~5–6 epochs, ~42–45% accuracy.
4. **Advanced CNN**: Some difficulty with small classes (like *disgust*).
5. **Real-Time**: Attempted to do inference on entire frames or via face detection.

### **3.2 Fixes and Tuning**

1. **Weighted Loss**: Increased penalty for rare classes. Helped recall for *disgust*.
2. **Dropout**: Higher p=0.4 mitigated overfitting.
3. **Early Stopping**: Freed us from unnecessary epochs.
4. **Normalization**: Dividing pixel intensities by 255 helped training stability.
5. **LR Scheduler**: StepLR or CosineAnnealing reduced learning rate and improved final accuracy.

### **3.3 Graphs, Screenshots, and Relevant Code**

* **Graphs**:
  + Train vs. Validation Loss show dropping train loss and partial overfitting.
* **Screenshots**:
  + Baseline console logs with ~25% accuracy.
  + Softmax final val accuracy ~37%.
  + MLP hitting ~43–45%.
  + Advanced CNN final ~58–60%.

**Key scripts**:

* baseline.py — Basic model.
* softmax.py — Flattened logistic regression approach.
* basic\_nn.py — Fully connected neural network.
* advanced\_network.py — Weighted cross entropy + CNN + early stopping.
* live\_inference.py — Real-time webcam classification using Haar Cascade or entire-frame classification.

## **4. Conclusions and Summary**

### **4.1 Achievements**

* Successfully built **four** progressive approaches: baseline, softmax, basic NN, and advanced CNN.
* Achieved **significant accuracy gains** from ~25% (baseline) to ~55–60% (advanced CNN).
* Demonstrated a **live** real-time inference script that classifies faces on the webcam feed.

### **4.2 Lessons Learned**

* **Baseline** sets a naive reference.
* **Softmax** is an improvement but limited for high-dimensional image data.
* **MLPs** can learn non-linearities but are still less optimal for images than CNNs.
* **CNNs** excel at capturing local image features (edges, textures). Weighted loss helps with class imbalance.
* Real-time usage requires both **face detection** (Haar or DNN) and **fast** classification.

### **4.3 Future Improvements**

1. **Data Augmentation**: Rotations, flips, brightness changes to expand training variety.
2. **Batch Normalization**: After each conv or fully connected layer.
3. **Advanced Architectures**: Transformers, Vision Transformers, or RNN-based approaches if analyzing sequential frames.
4. **Hyperparameter Tuning**: More extensive searching for learning rates, dropout rates, etc.
5. **Multiple Face Detection**: Real-time classification for each face individually rather than single-face assumption.

## **Document Format and References**

* **Language**: English, objective style (no personal pronouns).
* **Appendices**: Provide relevant Python code snippets or entire script references (e.g., baseline.py, softmax.py, basic\_nn.py, advanced\_network.py, live\_inference.py).
* **External**: For face detection in real-time, referencing **OpenCV** Haar cascades or DNN-based detection.

# **End of Part 2 Document**