**Name: Hashim Abbas  
ID: CSC-21F-041  
Course: Machine Learning  
Section: 8A**

**Introduction:**

The dataset I selected for this assignment is sourced from Kaggle and focuses on facial emotion recognition. It contains six distinct classes:

Angry  
Fear  
Happy  
Neutral  
Sad  
Surprise

Each class folder contains approximately 1,000 facial images, and to add an extra layer of complexity and insight, I further divided each class into two age-based categories:

Above 50 years (> 50)  
Below or equal to 50 years (<= 50)

This classification allows us to examine emotion recognition with age as a secondary factor. Before feeding these images into any facial recognition or emotion classification model, it is essential to perform preprocessing to clean, standardize, and optimize the dataset. Preprocessing enhances the dataset’s usability and significantly boosts model accuracy and robustness by eliminating irrelevant noise and reducing variability.

**Grayscale conversion**

The first preprocessing step involves converting the RGB (colored) images into grayscale. This is a standard step in most facial recognition systems because:

It reduces the complexity of the data by removing color information.  
It focuses on texture and contrast, which are more relevant for recognizing facial patterns.  
  
Models trained on grayscale images can generalize better, especially under varying lighting conditions.

We use OpenCV to perform the grayscale transformation.

**def to\_grayscale(img):**

**return cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

In grayscale, each pixel holds a single intensity value, simplifying both computation and storage. The model is less likely to overfit by memorizing color distributions instead of learning facial features.

**Resizing**

The next critical step is resizing every image to a fixed dimension, for example, 48x48 pixels. This helps create consistency across the entire dataset and reduces the computational burden.

**def resize\_image(img, size=(48, 48)):**

**return cv2.resize(img, size)**

Why it is important:

Reduced Computational Load: Smaller images speed up both training and prediction time.  
Data Consistency: Prevents the model from making biased decisions based on image size.  
Efficiency: Ideal for deploying models on mobile or edge devices with limited resources.

**Face Detection and Cropping**

Raw images often contain background noise or multiple faces. To make the model focus only on the primary subject (the face), we perform face detection and cropping. This step uses Haar Cascades from OpenCV to locate faces in an image and crop out the rest.

**def detect\_and\_crop\_face(img):**

**face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')**

**faces = face\_cascade.detectMultiScale(img, 1.1, 4)**

**for (x, y, w, h) in faces:**

**return img[y:y+h, x:x+w]**

**return img # Return original if no face is found**

**1)** It Removes distraction  
**2)** Improves accuracy  
**3)** Reduces Noise

**Data Augmentation**

A common problem in real-world datasets is class imbalance. For example, we may have more “Happy” images than “Fear” images. To address this, we use data augmentation, which artificially increases the size and diversity of our dataset by applying slight modifications to existing images.

**from keras.preprocessing.image import ImageDataGenerator**

**datagen = ImageDataGenerator(**

**rotation\_range=10, # Random rotation within 10 degrees**

**zoom\_range=0.1, # Random zoom within 10%**

**width\_shift\_range=0.1, # Horizontal shift**

**height\_shift\_range=0.1, # Vertical shift**

**horizontal\_flip=True, # Flip image horizontally**

**fill\_mode='nearest' # Filling strategy for new pixels**

**)**

**Why Augmentation?**

1) Increases dataset size  
2) Improves model generalization  
3) Reduces overfitting  
4) Makes the model more robust to rotations, flips