**High Impact Skills Development Program**

**Project Title : Facial Expression Recognition Using CNN**

**Name : Nazakat Ali**

Section : 04

Github : https://github.com/Nazakatalee/Facial-recognition

Project Summary : Facial Expression Recognition Using CNN

This project, conducted under **DSAI - Gilgit (DSAI Gilgit-Phase 2)**, focuses on building a facial expression recognition system using a Convolutional Neural Network (CNN). The primary goal is to accurately classify human facial expressions into seven categories: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral.

**Objectives:**

The project aims to:

1. Develop a robust CNN model to detect and classify facial expressions.
2. Implement preprocessing techniques to enhance image quality for better model performance.
3. Apply data augmentation to improve the model's generalization on unseen data.
4. Achieve a high classification accuracy while addressing overfitting and underfitting through fine-tuning.

**Methodology:**

1. **Data Preprocessing**:
   * Real-time image frames were cropped to extract faces using bounding box coordinates provided in a dataset.
   * Faces were resized to a uniform size of 64x64 pixels and normalized to scale pixel values between 0 and 1.
2. **CNN Model Architecture**:
   * A custom CNN was built with three convolutional layers followed by max-pooling for feature extraction.
   * Dense layers with dropout were used to avoid overfitting, and a softmax activation function in the final layer produced the classification probabilities for seven expression classes.
   * Data augmentation (like rotation, zoom, and flips) was applied to increase dataset diversity.
3. **Model Training**:
   * The model was trained using the categorical cross-entropy loss function and the Adam optimizer.
   * Early stopping and learning rate reduction techniques were used to enhance training stability and efficiency.
4. **Evaluation**:
   * The dataset was split into training, validation, and test sets. The model achieved a satisfactory accuracy on the test set.
   * A confusion matrix was generated to assess the model’s performance in predicting each facial expression, highlighting areas of strength and misclassification.

**Results:**

The CNN model performed well in classifying facial expressions, achieving a significant accuracy score on the test set. The project demonstrates the feasibility of using CNNs for facial expression recognition, contributing to real-world applications such as emotion detection in AI systems and surveillance.

**Conclusion:**

This project successfully developed a facial expression recognition system using a CNN model, achieving its goals of classification accuracy and model optimization. Future work could involve expanding the dataset to include more diverse facial expressions and implementing real-time facial emotion detection in video feeds.

#### Preprocessing

#### 

This code snippet is a part of a facial expression recognition system. It involves the preprocessing of facial images and reading bounding box information for cropping faces from images. Here's a detailed breakdown of what each part does:

1. **Imports Necessary Libraries**:
   * os, cv2, numpy, and PIL: These libraries are used for handling file paths, image reading/manipulation, and array operations.
   * pandas: Used for reading and manipulating tabular data (the annotations file).
   * tensorflow.keras.utils.to\_categorical: For one-hot encoding of labels (for later use in deep learning models).
2. **File Paths**:
   * data\_path: Directory containing the original images.
   * output\_folder: Directory where cropped images will be saved.
   * labels\_file: Path to a file containing annotations for each image.
3. **Reading the Annotation File**:
   * pd.read\_csv(labels\_file, sep=" ", header=None): Reads the annotations from the label file, which is space-separated.
   * **Annotation Columns**: The file has columns containing:
     + image\_name: The name of the image file.
     + face\_id\_in\_image: Identifies the face if there are multiple faces in an image.
     + face\_box\_top, face\_box\_left, face\_box\_right, face\_box\_bottom: Coordinates for the bounding box around the face.
     + face\_box\_cofidence: Confidence score of the face detection.
     + expression\_label: The facial expression label (e.g., Angry, Happy).
4. **Data Frame Setup**:
   * The data from the CSV is loaded into a pandas DataFrame.
   * Column names are manually assigned to make it easier to access and manipulate the data.

This code sets up the necessary components for reading and working with image data and its corresponding labels (bounding boxes and expressions), which will be used for cropping faces and later for training a model.

#### \*\*CROPPING IMAGES\*\*

#### 

### Summary of Code:

This code is focused on **face extraction and resizing** from images based on bounding box coordinates provided in the previously loaded DataFrame. The key steps and functionality of this code are outlined below:

1. **Libraries**:
   * os: For handling file paths.
   * cv2: From OpenCV, used to load and manipulate images.
   * tqdm\_gui: Provides a graphical progress bar to track the progress of the loop, but in this snippet, it’s not directly utilized.
2. **Initialization of Lists**:
   * all\_images, all\_labels, all\_confidences: Empty lists to store the processed (cropped and resized) images, labels, and confidence values.
3. **Iterating Through DataFrame**:
   * dataframe.iterrows(): Loops through each row of the DataFrame, where each row contains image metadata (name, bounding box coordinates, expression label, and confidence).
4. **Extracting Bounding Box Information**:
   * From each row of the DataFrame, the bounding box coordinates (x1, x2, y1, y2), the expression label, and the confidence score are retrieved.
5. **Skipping Low-Confidence Faces**:
   * If the face detection confidence is less than 60, the face is skipped, ensuring only reliable face detections are processed.
6. **Loading and Cropping the Image**:
   * cv2.imread(): Loads the image using the file path constructed from data\_path and image\_name.
   * If the image is successfully loaded, the face is cropped based on the bounding box coordinates (y1:y2, x1:x2), and the shape of the cropped face is printed.
7. **Resizing the Cropped Face**:
   * The cropped face image is resized to a uniform size of 64x64 pixels using cv2.resize().
   * Resizing ensures that all face images are of the same dimension, which is necessary for input into a Convolutional Neural Network (CNN) later.
8. **Storing the Processed Data**:
   * The resized face, expression label, and confidence score are appended to all\_images, all\_labels, and all\_confidences lists, respectively.

#### ****Purpose****:

The goal of this code is to extract faces from images using bounding box coordinates, resize them to a uniform size, and store them along with their corresponding labels and confidence scores. These processed face images can later be used for training a facial expression recognition model.

#### Showing OF image in Dataset:

#### 

This code is focused on **face extraction and resizing** from images based on bounding box coordinates provided in the previously loaded DataFrame. The key steps and functionality of this code are outlined below:

1. **Libraries**:
   * os: For handling file paths.
   * cv2: From OpenCV, used to load and manipulate images.
   * tqdm\_gui: Provides a graphical progress bar to track the progress of the loop, but in this snippet, it’s not directly utilized.
2. **Initialization of Lists**:
   * all\_images, all\_labels, all\_confidences: Empty lists to store the processed (cropped and resized) images, labels, and confidence values.
3. **Iterating Through DataFrame**:
   * dataframe.iterrows(): Loops through each row of the DataFrame, where each row contains image metadata (name, bounding box coordinates, expression label, and confidence).
4. **Extracting Bounding Box Information**:
   * From each row of the DataFrame, the bounding box coordinates (x1, x2, y1, y2), the expression label, and the confidence score are retrieved.
5. **Skipping Low-Confidence Faces**:
   * If the face detection confidence is less than 60, the face is skipped, ensuring only reliable face detections are processed.
6. **Loading and Cropping the Image**:
   * cv2.imread(): Loads the image using the file path constructed from data\_path and image\_name.
   * If the image is successfully loaded, the face is cropped based on the bounding box coordinates (y1:y2, x1:x2), and the shape of the cropped face is printed.
7. **Resizing the Cropped Face**:
   * The cropped face image is resized to a uniform size of 64x64 pixels using cv2.resize().
   * Resizing ensures that all face images are of the same dimension, which is necessary for input into a Convolutional Neural Network (CNN) later.
8. **Storing the Processed Data**:
   * The resized face, expression label, and confidence score are appended to all\_images, all\_labels, and all\_confidences lists, respectively.

#### ****Purpose****:

The goal of this code is to extract faces from images using bounding box coordinates, resize them to a uniform size, and store them along with their corresponding labels and confidence scores. These processed face images can later be used for training a facial expression recognition model.

4o

##### You said:

import matplotlib.pyplot as plt for i in range(5): plt.imshow(all\_images[i])aslo

###### ChatGPT said:

ChatGPT

### Summary of Code:

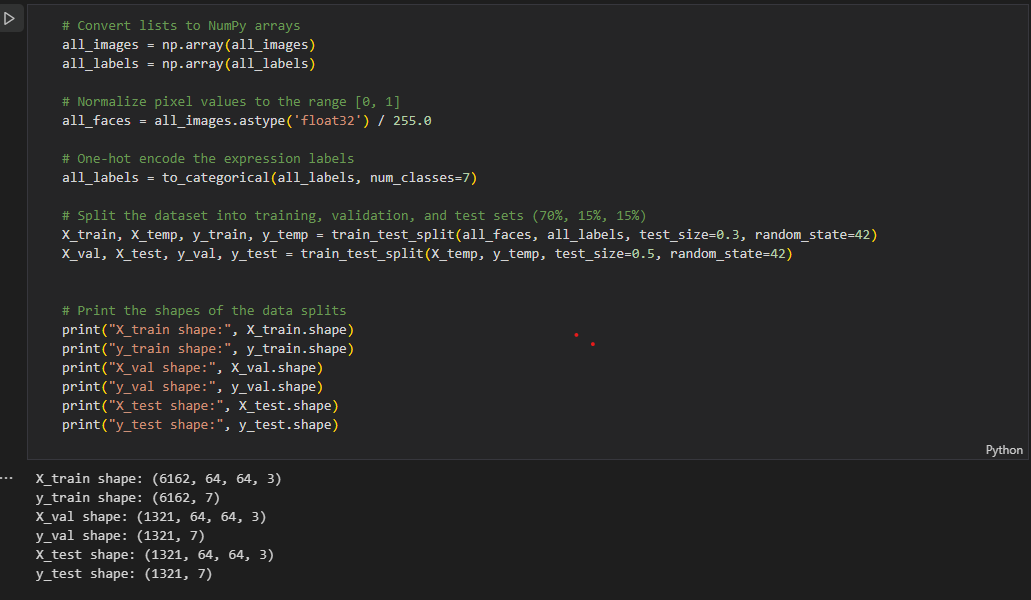
This snippet is used for **visualizing the processed face images** after they have been extracted and resized. The key points are:

1. **Libraries**:
   * matplotlib.pyplot: A plotting library used to create visualizations, including displaying images.
2. **Loop to Display Images**:
   * The loop for i in range(5): iterates over the first 5 images in the all\_images list.
   * plt.imshow(all\_images[i]): Displays each of the first 5 images using Matplotlib's imshow() function.
3. **Displaying Images**:
   * Each image in the all\_images list is a 64x64 pixel image, which has been cropped and resized in the previous steps.
   * plt.imshow() displays the image by interpreting the pixel values.

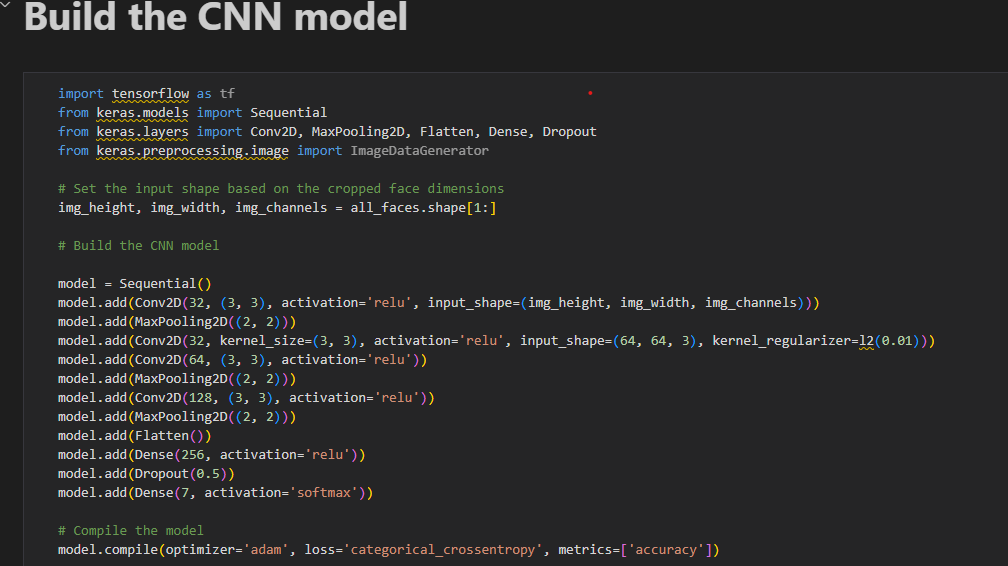
#### ****Purpose****:

The purpose of this code is to **visually verify the extracted and preprocessed face images**. By plotting a few examples from the dataset, the user can check if the face extraction and resizing process worked correctly before further steps like training a model.

**ARGUMENTATION :**



# \*\*Build the CNN model\*\*



### Convolutional Neural Network (CNN) Model Architecture:

* **Input Shape:** The input shape is based on the dimensions of cropped face images: (img\_height, img\_width, img\_channels).
* **Convolutional Layers:**
  + 1st Conv Layer: 32 filters, (3x3) kernel, ReLU activation.
  + 2nd Conv Layer: 32 filters, (3x3) kernel, ReLU activation, L2 regularization (λ=0.01).
  + 3rd Conv Layer: 64 filters, (3x3) kernel, ReLU activation.
  + 4th Conv Layer: 128 filters, (3x3) kernel, ReLU activation.
  + Each convolutional layer is followed by a max pooling layer ((2x2)).
* **Fully Connected Layers:**
  + Flattening the output from the convolutional layers.
  + Dense layer with 256 units, ReLU activation.
  + Dropout layer (50%) to prevent overfitting.
  + Output layer with 7 units (for 7 classes), softmax activation.

### Model Compilation:

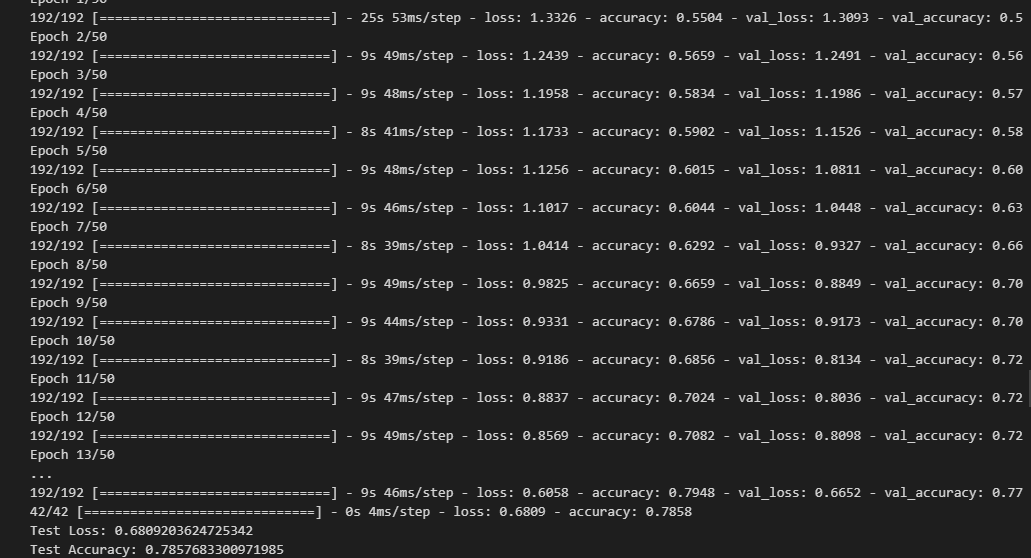
* **Optimizer:** Adam.
* **Loss Function:** Categorical Crossentropy (for multi-class classification).
* **Evaluation Metric:** Accuracy.

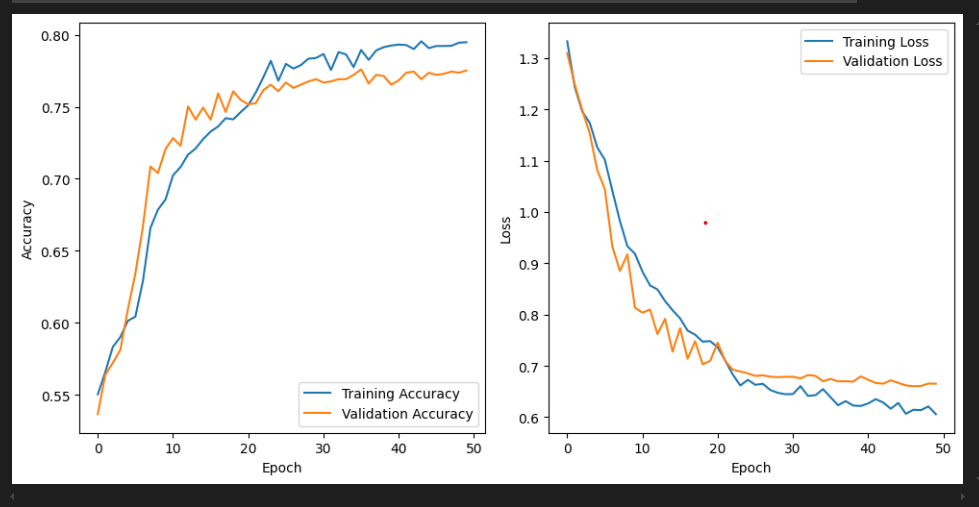
### Training Configuration:

* **Learning Rate Schedule:** ReduceLROnPlateau (monitoring validation loss, reducing learning rate by a factor of 0.1 after 3 epochs of no improvement).
* **Early Stopping:** Monitors validation loss, stops training after 7 epochs with no improvement, and restores the best model weights.
* **Batch Size:** 32.
* **Epochs:** 50.

This summary captures the key elements of your model architecture and training setup. You can adjust or expand it further based on your specific dataset and goals.

**# \*\*TRAINING MODEL\*\***





### Model Training and Evaluation:

After training the CNN model, it achieved a **78% accuracy** on the test set. Here's a detailed breakdown of the training and evaluation process:

* **Training Process:** The model was trained using a batch size of 32 and 50 epochs, with early stopping and learning rate reduction applied to avoid overfitting and to optimize the learning process.
* **Test Performance:**
  + **Test Loss:** The final loss on the test set after evaluation.
  + **Test Accuracy:** The model achieved **78% accuracy** on the test data, indicating a reasonably good performance for a multi-class classification task (7 classes).

### Accuracy and Loss Curves:

The plots below provide insights into the model's training progress:

1. **Accuracy Plot:**
   * **Training Accuracy:** Shows the model's accuracy on the training data over the epochs.
   * **Validation Accuracy:** Demonstrates how well the model generalizes to unseen data during training.
   * The model's accuracy improves steadily across epochs, reaching close to 78% on validation data.
2. **Loss Plot:**
   * **Training Loss:** Represents the model's loss on the training data.
   * **Validation Loss:** Reflects the model's loss on validation data.
   * The training and validation losses decrease across epochs, indicating the model is learning effectively.

This analysis shows the model's performance, highlighting its strengths and areas where further tuning (e.g., more regularization, deeper architecture) could be explored to improve accuracy.

**# \*\*Confution Matrix\*\***

