Digital Egypt Pioneers Initiative-DEPI
Microsoft Machine Learning Engineer-Round Two
ONL2_AIS2_S1 Machine Learning Group

Instructor: Zeyad Mohamed

Facial Recognition System

[Milestone One: Data Collection, Exploration, and Preprocessing]

Introduction:

The Mialiston 1 phase of the Facial Recognition System project focuses on laying the foundational groundwork necessary for building a robust and accurate face recognition model. This milestone centers on collecting, exploring, and preprocessing a diverse and high-quality facial image dataset. The overarching goal is to prepare data that supports effective model training and ensures generalization across varied real-world conditions such as lighting, angles, and facial expressions. Through careful selection and preprocessing—including face detection, normalization, and data augmentation—this phase aims to enable the subsequent development of a facial recognition system that is both secure and performant. The work completed in Mialiston 1 serves as a critical step toward deploying a reliable facial authentication and identification system for security and access control applications.

Dataset Description:

The Labeled Faces in the Wild (LFW) dataset is a well-known benchmark in facial recognition research, containing over 13,000 labeled images of faces collected from the web. These images represent 5,749 individuals, with a subset of 1,680 people having two or more images. Each image is labeled with the person's name, and the dataset includes variations in pose, lighting, facial expressions, age, gender, and background, making it ideal for evaluating face recognition algorithms under unconstrained conditions. The images are in JPEG format and have been cropped and aligned using the Viola-Jones face detector, ensuring consistency for model training. Widely used in academic and industry settings, LFW remains a standard for testing identity verification and face matching systems.

Digital Egypt Pioneers Initiative-DEPI
Microsoft Machine Learning Engineer-Round Two
ONL2_AIS2_S1 Machine Learning Group

Code Structure:

Explain the Technique of loading the data:

The dataset is imported from Kaggle using kagglehub into the notebook by the following block to start working on it.

```
import kagglehub

# Update this line to the LFW dataset on Kaggle
lfw_dataset_path = kagglehub.dataset_download('jessicali9530/lfw-dataset')

print('Data source import complete.')
```

The following block shows that the path to the dataset is defined, and all subdirectories (each representing a unique individual) are listed. num_classes gives the total number of unique identities in the dataset.

```
# Define paths
data_dir = '/kaggle/input/lfw-dataset/lfw-deepfunneled/lfw-deepfunneled/'  #

# Remove the extra 'lfw-deepfunneled' from the path
Faces = os.listdir(data_dir)
num_classes = len(Faces)
```

Explain some visuals for more insights:

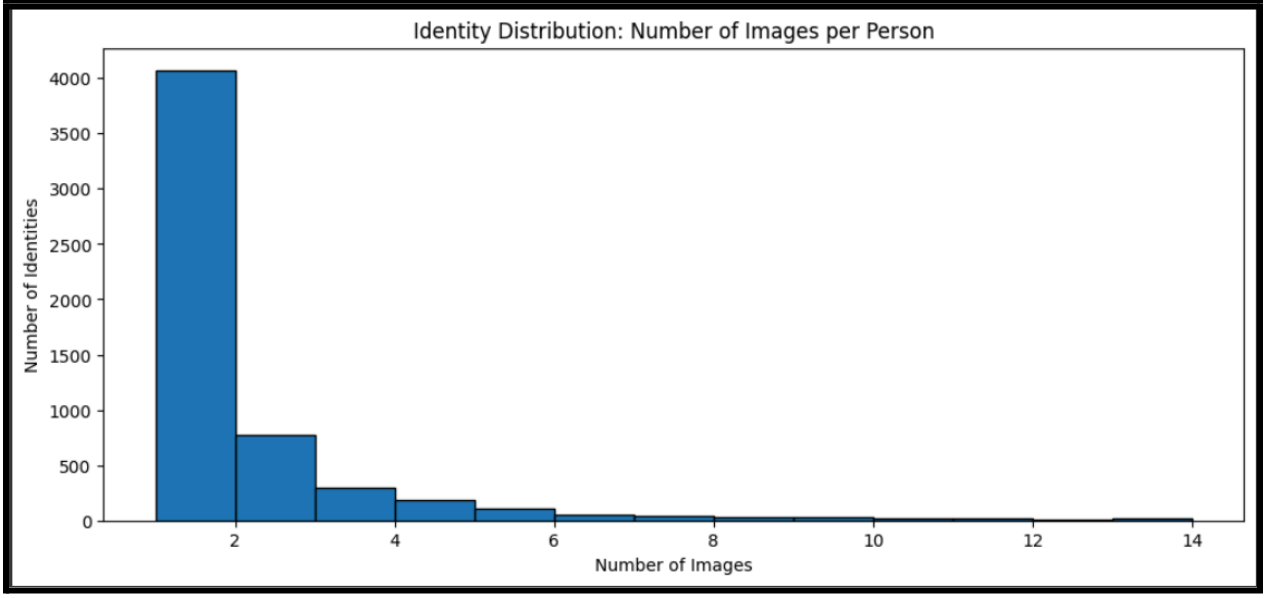
The following block selects 12 images randomly for different individuals. It provides a quick visual confirmation of the data quality and diversity in facial features, lighting, and expressions.

```
plt.figure(figsize=(15, 10))
for i in range(12):

    # Select a random face directory
    random_face_dir = np.random.choice(Faces)

    # Choose an image file from within the face directory
    img_file = np.random.choice(os.listdir(os.path.join(data_dir, random_face_dir)))

    # Construct the full image path
    img_path = os.path.join(data_dir, random_face_dir, img_file)
    img = tf.keras.preprocessing.image.load_img(img_path)
```

Digital Egypt Pioneers Initiative-DEPI
Microsoft Machine Learning Engineer-Round Two
ONL2_AIS2_S1 Machine Learning Group

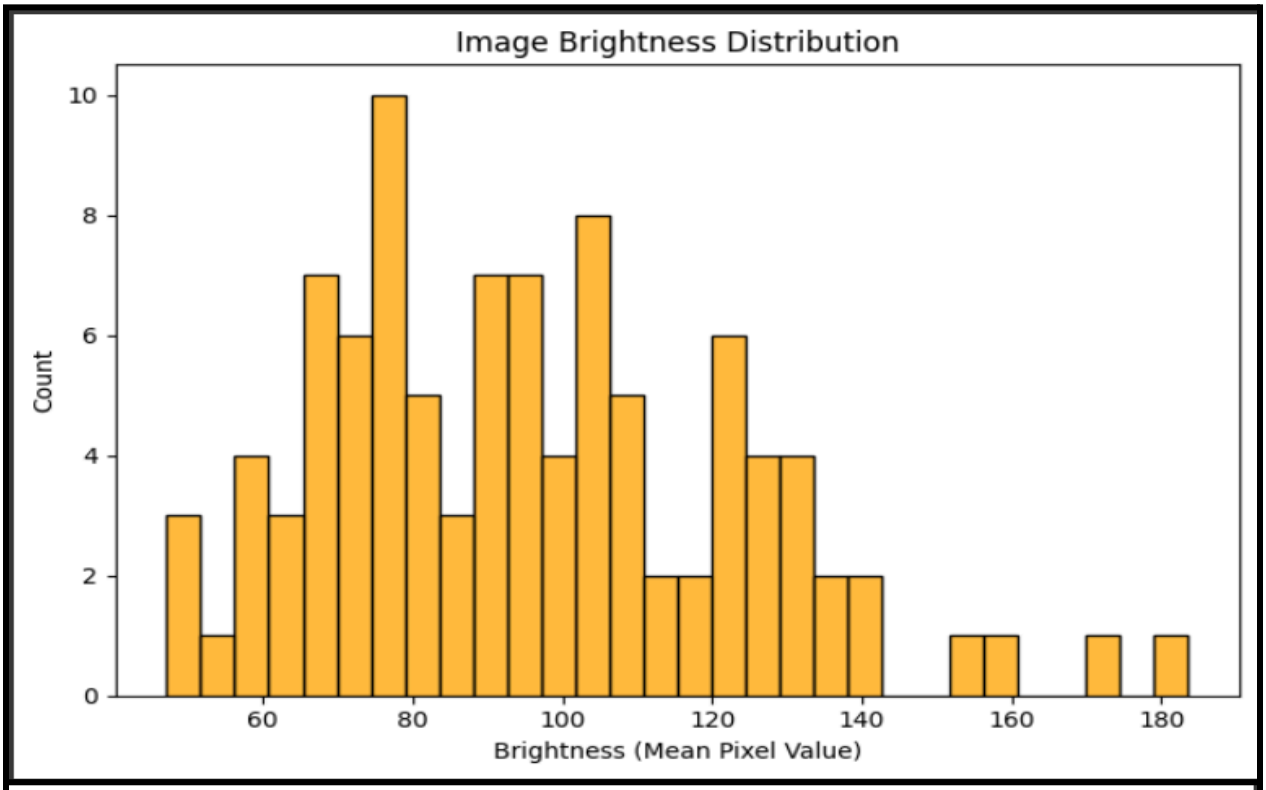
The following block gets the count of images for each individual, then stores the result in a data frame then starts to plot them using the histogram. We can see from the chart that the signal image for an individual is a problem faced by the trained data, also see the maximum number of images for each individual and count the unique images.

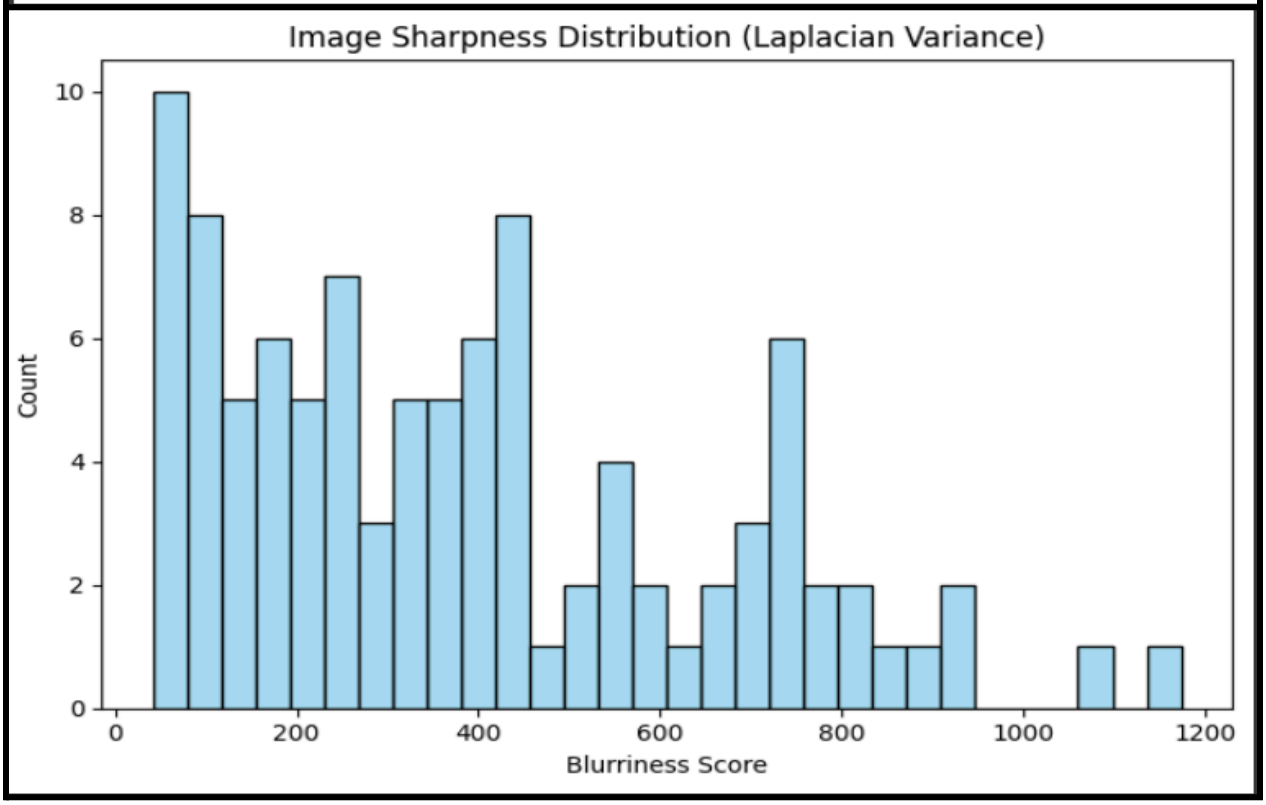
```python
from collections import Counter

identity_distribution = {}
for person in Faces:
    img_files = os.listdir(os.path.join(data_dir, person))
    identity_distribution[person] = len(img_files)

# Convert to DataFrame for visualization
id_df = pd.DataFrame(list(identity_distribution.items()), columns=["Identity", "Image_Count"])
id_df = id_df.sort_values(by="Image_Count", ascending=False)

# Summary
print("Total identities:", len(id_df))
print("Max images for one identity:", id_df['Image_Count'].max())
print("Identities with only 1 image:", (id_df['Image_Count'] == 1).sum())
```



The following block randomly samples approximately 1,000 images from the dataset to analyze key image-level characteristics that impact model performance. These characteristics include the resolution of each image (measured by its width and height), the brightness (calculated as the average pixel intensity), and the blurriness, which is estimated using the variance of the Laplacian method from OpenCV—a common approach for measuring image sharpness. The extracted features are then summarized

using descriptive statistics and visualized through histograms to better understand the overall quality and consistency of the dataset.





Digital Egypt Pioneers Initiative-DEPI
Microsoft Machine Learning Engineer-Round Two
ONL2_AIS2_S1 Machine Learning Group

The following block displays a set of 10 randomly selected facial images to allow for a quick visual assessment of the dataset. This inspection focuses on identifying variations in facial expressions, head poses, and the overall clarity and framing of the images. By visually confirming these variations, we can better understand the dataset's diversity, which is crucial for training a robust and generalizable facial recognition model. This step ensures that the data reflects real-world conditions, thereby enhancing the model's ability to perform accurately across different scenarios.

```
plt.figure(figsize=(15, 5))
sample_paths = np.random.choice(sample_imgs, 10)
for i, path in enumerate(sample_paths):
    img = Image.open(path)
    plt.subplot(2, 5, i+1)
    plt.imshow(img)
    plt.title(os.path.basename(os.path.dirname(path)))
    plt.axis("off")
plt.suptitle("Sample Images to Observe Facial Expressions", fontsize=16)
plt.tight_layout()
plt.show()
```

The final block in this section identifies and flags images considered blurry by applying an empirical threshold—specifically, images with a blurriness score below 100 (based on the variance of the Laplacian) are marked as blurry. It then calculates the percentage of blurry images within the sampled dataset to quantify the extent of the issue. To support this analysis, a small set of these blurry images is displayed for visual confirmation. This step is essential in evaluating the need for additional preprocessing or data cleaning, as excessive image blur can negatively impact the accuracy and reliability of a facial recognition model.

```
img_df["Is_Blurry"] = img_df["Blurriness"] < 100
blurry_ratio = img_df["Is_Blurry"].mean()
print(f"Percentage of blurry images in sample: {blurry_ratio*100:.2f}%")

# Show some blurry images
blurry_paths = [sample_imgs[i] for i in range(len(sample_imgs)) if img_df["Is_Blurry"].iloc[i]]
plt.figure(figsize=(12, 6))
for i, path in enumerate(blurry_paths[:8]):
    img = Image.open(path)
    plt.subplot(2, 4, i+1)
    plt.imshow(img)
    plt.title("Blurry")
    plt.axis("off")
plt.tight_layout()
plt.show()
```

Digital Egypt Pioneers Initiative-DEPI
Microsoft Machine Learning Engineer-Round Two
ONL2_AIS2_S1 Machine Learning Group

Face Detection and Cropping Setup:

This block begins by importing the necessary libraries for image processing, neural network input preparation, and visualization. It then defines important constants such as the target image size (224, 224), a limit on the number of people (MAX_PEOPLE = 100), and the number of images per person (MAX_IMAGES_PER_PERSON = 3). To perform face detection, the code loads OpenCV's Haar Cascade classifier, a widely used algorithm for detecting frontal human faces. A custom function, detect_and_crop_face() is defined to convert images to grayscale, detect faces using the cascade, crop the face region, resize it to the target size, and normalize the pixel values by dividing by 255. This ensures that the model input focuses on the face region and maintains a consistent format.

```python
# Constants
IMG_SIZE = (224, 224)
MAX_PEOPLE = 100         # Safe upper limit to avoid RAM crashes
MAX_IMAGES_PER_PERSON = 3

# Haar Cascade for Face Detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

```python
# Face detection and cropping
def detect_and_crop_face(img_array):
    gray = cv2.cvtColor(img_array, cv2.COLOR_RGB2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    if len(faces) > 0:
        x, y, w, h = faces[0]
        cropped = img_array[y:y+h, x:x+w]
        resized = cv2.resize(cropped, IMG_SIZE)
        return resized / 255.0
    else:
        # No face found – fallback
        return cv2.resize(img_array, IMG_SIZE) / 255.0
```

Data Preprocessing Stage:

The preprocess_lfw_dataset_safe() function processes the LFW dataset by iterating through a limited number of people and images to avoid memory issues. For each selected person, the function loads up to three images, converts them into arrays, applies the face detection and cropping routine, and appends the results to a list of processed images and corresponding labels. This approach efficiently prepares a manageable dataset of normalized facial images suitable for model training or testing.

Finally, it converts the collected data into NumPy arrays and prints their shapes to confirm successful preprocessing.

```
def preprocess_lfw_dataset_safe(data_dir, max_people=MAX_PEOPLE, max_images_per_person=MAX_IMAGES_PER_PERSON):
    images = []
    labels = []

    people = sorted(os.listdir(data_dir))[:max_people]
    for person in people:
        person_path = os.path.join(data_dir, person)
        image_files = os.listdir(person_path)[:max_images_per_person]
```

```
        for file in image_files:
            img_path = os.path.join(person_path, file)
            img = tf.keras.preprocessing.image.load_img(img_path)
            img_array = tf.keras.preprocessing.image.img_to_array(img).astype(np.uint8)

            processed_img = detect_and_crop_face(img_array)
            images.append(processed_img)
            labels.append(person)

    return np.array(images), np.array(labels)
```

```
# Run preprocessing
X, y = preprocess_lfw_dataset_safe('/kaggle/input/lfw-dataset/lfw-deepfunneled/lfw-deepfunneled')
print("Data shape:", X.shape, "Labels:", len(y))
```

Once the image data is preprocessed, the next step involves preparing for data augmentation using TensorFlow's ImageDataGenerator. The configuration includes random transformations such as rotation, zoom, horizontal shifts, vertical shifts, and horizontal flips. These augmentations simulate real-world image variations and help improve the model's generalization by exposing it to diverse examples. One image from the dataset (X[31]) is selected and reshaped to fit the expected input format for the generator. Although not used directly in model training here, the setup can be seamlessly integrated into a training pipeline using .flow().

```
augmentor = ImageDataGenerator(
    rotation_range=30,
    zoom_range=0.3,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

img = X[31]  # One image from the dataset
img = np.expand_dims(img, axis=0)  # Required shape (1, 224, 224, 3)
```

Digital Egypt Pioneers Initiative-DEPI
Microsoft Machine Learning Engineer-Round Two
ONL2_AIS2_S1 Machine Learning Group

To demonstrate the effect of augmentation, this block generates eight augmented versions of a single input image using the configured ImageDataGenerator. It then plots the original image alongside its augmented variants in a 3x3 grid. This visualization showcases how the same image can be modified through various transformations while preserving its semantic content, such as facial features. This diversity helps train a more robust model that performs well under real-world conditions involving changes in pose, lighting, or orientation.

```
augmented_images = [next(augmentor.flow(img, batch_size=1))[0] for _ in range(8)]

# Plot original and augmented
plt.figure(figsize=(12, 6))
plt.subplot(3, 3, 1)
plt.imshow(X[31])
plt.title("Original")
plt.axis('off')
```

The following picture demonstrates an example of plotting the augmented images with the original one to show the difference between them and the impact of augmentation.