Names

1. George Reda George 49-0301
2. Fady Mounir Fouad 49-0716
3. Youssef Atef Mourad 49-5215

Emails:

George.reda@student.guc.edu.eg,

fady.elsokary@student.guc.edu.eg,

Youssef.mourad@student.guc.edu.eg

Milestone 2

Model Visualization and Analysis Report

1. Model Architecture Visualization:

Convolutional Layers:

- **Convolutional Layers:**
  - ConvBlock 1:
    - Number of Filters: 5
    - Filter Size: (3, 3, 3)
    - Pooling Size: (2, 2)
  - ConvBlock 2:
    - Number of Filters: 5
    - Filter Size: (3, 3, 3)
    - Pooling Size: (2, 2)
  - ConvBlock 3:
    - Number of Filters: 5
    - Filter Size: (3, 3, 3)
    - Pooling Size: (2, 2)
- **Flattening Layer:** Converts the output of the convolution blocks to a flattened form

```python
# Create the network
class SimpleCNN:
    def __init__(self):
        """
        Initialize the SimpleCNN network.
        """
        self.conv_block1 = ConvBlock(5)  # Filter size is (3, 3, 3)
        self.conv_block2 = ConvBlock(5)  # Filter size is (3, 3, 3)
        self.conv_block3 = ConvBlock(5)  # Filter size is (3, 3, 3)
        self.flatten_layer = FlattenLayer()

    def forward(self, input_data):
        """
        Perform forward pass for the SimpleCNN network.

        :param input_data: Input data (batch_size, height, width, channels).
        :return: Output data after passing through the network.
        """
        # Convolution block 1
        conv1_output = self.conv_block1.forward(input_data)

        # Convolution block 2
        conv2_output = self.conv_block2.forward(conv1_output)

        # Convolution block 3
        conv3_output = self.conv_block3.forward(conv2_output)

        # Flatten
        flattened_output = self.flatten_layer.forward(conv3_output)

        # Downsampling to size 1x128 (assuming the flattened size is 1x128 or larger)
        downsampled_output = flattened_output[:, :128]

        return downsampled_output
```

```python
# Define the Convolution block
class ConvBlock:
    def __init__(self, num_filters, filter_size=(3, 3, 3), pool_size=(2, 2)):
        """
        Initialize the ConvBlock.

        :param num_filters: Number of filters.
        :param filter_size: Size of the filters (height, width, channels). Default is (3, 3, 3).
        :param pool_size: Size of the pooling filter (height, width). Default is (2, 2).
        """
        self.conv_layer = ConvLayer(num_filters, filter_size)
        self.pool_layer = PoolingLayer(pool_size)

    def forward(self, input_data):
        """
        Perform forward pass for the ConvBlock.

        :param input_data: Input data (batch_size, height, width, channels).
        :return: Output data after convolution and pooling.
        """
        conv_output = self.conv_layer.forward(input_data)
        pool_output = self.pool_layer.forward(conv_output)
        return pool_output


# Define the Flattening layer
class FlattenLayer:
    def forward(self, input_data):
        """
        Flatten the input data.

        :param input_data: Input data (batch_size, height, width, channels).
        :return: Flattened data.
        """
        batch_size = input_data.shape[0]
        return input_data.reshape(batch_size, -1)
```

2. Layer Dimensions and In/Out Data:

- ConvBlock 1:
    - Input: Batch size x 512 x 512 x 3 (assuming images are resized to 512x512x3)
    - Output after Convolution: Batch size x (512 - 3 + 1) x (512 - 3 + 1) x 5 (due to 5 filters)
    - Output after Pooling: Batch size x (256) x (256) x 5 (due to 2x2 pooling)
- ConvBlock 2:
    - Input: Batch size x 256 x 256 x 5 (output from ConvBlock 1)
    - Output after Convolution: Batch size x (256 - 3 + 1) x (256 - 3 + 1) x 5 (due to 5 filters)
    - Output after Pooling: Batch size x (128) x (128) x 5 (due to 2x2 pooling)
- ConvBlock 3:
    - Input: Batch size x 128 x 128 x 5 (output from ConvBlock 2)

- Output after Convolution: Batch size x (128 - 3 + 1) x (128 - 3 + 1) x 5 (due to 5 filters)
- Output after Pooling: Batch size x (64) x (64) x 5 (due to 2x2 pooling)

- Flattening Layer:
  - Input: Batch size x 64 x 64 x 5 (output from ConvBlock 3)
  - Output: Batch size x 20480 (flattened to 1D array)

```python
import os
import cv2
import numpy as np

folder_path1 = r'G:\Deep Learning in Computer Vision (DMET1067)\Milestone 2\Our Dataset\Our Dataset\Karim Abdelaziz Cropped 2\KarimTraining'
folder_path2 = r'G:\Deep Learning in Computer Vision (DMET1067)\Milestone 2\Our Dataset\Our Dataset\Hend Sabry Cropped\HendTraining'
folder_path3 = r'G:\Deep Learning in Computer Vision (DMET1067)\Milestone 2\Our Dataset\Our Dataset\Hany Ramzy Cropped\HanyTraining'

# Function to read images from a folder and resize them
def read_images_from_folder(folder_path, label, image_size=(512, 512)):
    images = []
    labels = []

    for filename in os.listdir(folder_path):
        img_path = os.path.join(folder_path, filename)
        img = cv2.imread(img_path)

        if img is not None:
            # Resize image to 512x512x3
            img = cv2.resize(img, image_size)
            images.append(img)
            labels.append(label)

    return images, labels

# Paths to your image folders and labels
folder_paths = [folder_path1, folder_path2, folder_path3]
labels = ["Karim", "Hend", "Hany"]  # Example labels for each folder

# Initialize empty lists to store images and corresponding labels
test_images = []
test_true_labels = []
```

```python
class ConvLayer:
    def __init__(self, num_filters, filter_size, filter_weights=None):
        self.num_filters = num_filters
        self.filter_size = filter_size

        if filter_weights is None:
            # Initialize random filter weights
            self.filters = np.random.randn(num_filters, *filter_size) / (filter_size[0] * filter_size[1])
        else:
            # Use predefined filter weights

            self.filters = filter_weights

    @classmethod
    def from_filters(cls, filters):
        """
        Initialize the ConvLayer with predefined filter weights.

        :param filters: Predefined filters (num_filters, height, width, channels).
        """
        num_filters, height, width, channels = filters.shape
        conv_layer = cls(num_filters, (height, width, channels))
        conv_layer.filters = filters
        return conv_layer
    def iterate_filters(self, input_data):
        """
        Iterate over filters and apply them to the input data.

        :param input_data: Input data (batch_size, height, width, channels).
        :return: Convolved output.
        """
        batch_size, height, width, channels = input_data.shape

        # Iterate over each filter
        for f in range(self.num_filters):
```

Model Hyper-parameters and Training Details

- **Model Hyper-parameters:**
  - Filter Sizes: (3, 3, 3) for all ConvBlocks
  - Pooling Sizes: (2, 2) for all ConvBlocks

```
# Model hyper-parameters
filter_size = (3, 3, 3)
pool_size = (2, 2)
```

additional notes = """

- The model architecture consists of three ConvBlocks with 5 filters each, followed by pooling and flattening layers.

- Initially, Sigmoid activation function was used, but it was found that ReLU produced better accuracy.

<u>- K-means clustering is applied to extract features and predict labels, achieving an accuracy of 44.53%.</u>

"""

```
# Create a mapping dictionary to map cluster labels to actor names
unique_labels = np.unique(test_true_labels)
label_map = {i: actor_name for i, actor_name in enumerate(unique_labels)}

# Map cluster labels to actor names for predicted labels
predicted_actor_names = [label_map[label] for label in predicted_labels]

# Calculate accuracy
accuracy = accuracy_score(test_true_labels, predicted_actor_names)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 44.53%
```

The predefined Model

1. Introduction:

In this report, we present a Convolutional Neural Network (CNN) approach for the classification of images belonging to three distinct actors: Karim AbdelAziz, Hend Sabry, and Hany Ramzy. The task involves training a model to accurately identify which actor appears in a given image. This report outlines the methodology used, including data preparation, model architecture, training procedure, evaluation metrics, and concluding remarks.

2. Data Preparation:

- The dataset consists of images of the mentioned actors, divided into training, testing, and validation sets. The images are loaded from predefined folder paths, resized, and converted into numpy arrays. Additionally, labels are assigned to each image based on the actor depicted.

The number of input data for each actor/actress is the following:

```
df_train.label.value_counts()
```

```
Karim AbdelAziz    2390
Hany Ramzy         1574
Hend Sabry          826
Name: label, dtype: int64
```

```
unique_labels = df['label'].unique()
print("Unique labels in the DataFrame:")
print(unique_labels)
```

```
Unique labels in the DataFrame:
['Karim AbdelAziz' 'Hend Sabry' 'Hany Ramzy']
```

```
label2id = {label: idx for idx, label in enumerate(unique_labels)}
id2label = {idx: label for label, idx in label2id.items()}
print("label2id mapping:")
print(label2id)
print("id2label mapping:")
print(id2label)
```

```
label2id mapping:
{'Karim AbdelAziz': 0, 'Hend Sabry': 1, 'Hany Ramzy': 2}
id2label mapping:
{0: 'Karim AbdelAziz', 1: 'Hend Sabry', 2: 'Hany Ramzy'}
```

Also, each Actor is indexed for classification as shown above.

- **Data Shuffling:**

  To introduce randomness and prevent biases during training, the training and testing datasets are shuffled using a random seed. Shuffling ensures that the model does not inadvertently learn patterns based on the order of data samples.

```
# Map labels to ids

df_train['class'] = df_train['label'].map(label2id)
df_test['class'] = df_test['label'].map(label2id)

# shuffling samples
df_train = df_train.sample(frac=1, random_state=SEED)
df_test = df_test.sample(frac=1, random_state=SEED)
```

- **Image Processing:**

  The **fix_image** function converts image formats to RGB, ensuring consistency in color channels across all images. This step is crucial as it standardizes the input format expected by the CNN model.

```
def fix_image(img):

    rgb_img = img.convert("RGB")

    return np.array(rgb_img)
```

- **Data Conversion:**

  The images and corresponding labels are converted into numpy arrays for efficient handling and processing during model training and evaluation. Additionally, labels are

converted into one-hot encoded vectors using the **to_categorical** function, enabling multi-class classification.

```
train_images = np.array([fix_image(img) for img in df_train['image'].to_list()])
test_images = np.array([fix_image(img) for img in df_test['image'].to_list()])


train_labels = np.array([label for label in df_train['class'].to_list()])
test_labels = np.array([label for label in df_test['class'].to_list()])


train_labels = to_categorical(train_labels, 3)
test_labels = to_categorical(test_labels, 3)
```

3. Model Architecture:

The CNN model architecture used for actor classification comprises convolutional layers, pooling layers, and fully connected layers. The exact configuration of the model is as follows:

- Input layer: Accepts images of predefined dimensions.
- Convolutional layers: Extract features from input images.
- Pooling layers: Reduce spatial dimensions of feature maps.
- Fully connected layers: Perform classification based on extracted features.

4. Training:

The training process involves mapping labels to corresponding numerical IDs and shuffling the samples to ensure randomness. Additionally, the images are processed to fix any inconsistencies in format, such as converting to RGB. The preprocessed images and

labels are then saved into numpy arrays for efficient handling during training.

The CNN model architecture consists of multiple convolutional layers followed by max-pooling layers for feature extraction and spatial reduction, respectively. The specific configuration of the model includes convolutional layers with varying filter sizes and activations, followed by max-pooling operations. The model is then flattened to transition from feature extraction to classification stages, with fully connected layers facilitating the classification task. Dropout regularization is applied to mitigate overfitting.

```
# Convert labels to one-hot encoding
num_classes = 3
# CNN model Architecture
# CNN model Architecture
model = Sequential()

# Feature Extraction Stage
model.add(Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), padding="valid", activation='relu', input_shape=(512, 512, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), padding="valid", activation='relu', input_shape=(512, 512, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), padding="valid", activation='relu')) # Number of parameters = 16
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Conv2D(filters=32, kernel_size=(5, 5), strides=(1, 1), padding="valid", activation='relu', input_shape=(512, 512, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Conv2D(filters=16, kernel_size=(7, 7), strides=(1, 1), padding="valid", activation='relu')) # Number of parameters = 32
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
```

5. Training and Evaluation:

The compiled model is trained using the training images and labels, with a portion of the data reserved for validation to monitor performance and prevent overfitting. The Adam optimizer is employed with a learning rate of 0.001, optimizing the categorical cross-entropy loss function. The model undergoes

training for 5 epochs with a batch size of 32, during which the weights are adjusted to minimize the loss.
Following training, the trained model is saved for future use. To evaluate the model's performance, it is tested on the unseen testing dataset. Predictions are generated for the testing images, and the accuracy is computed by comparing the predicted labels to the ground truth labels.

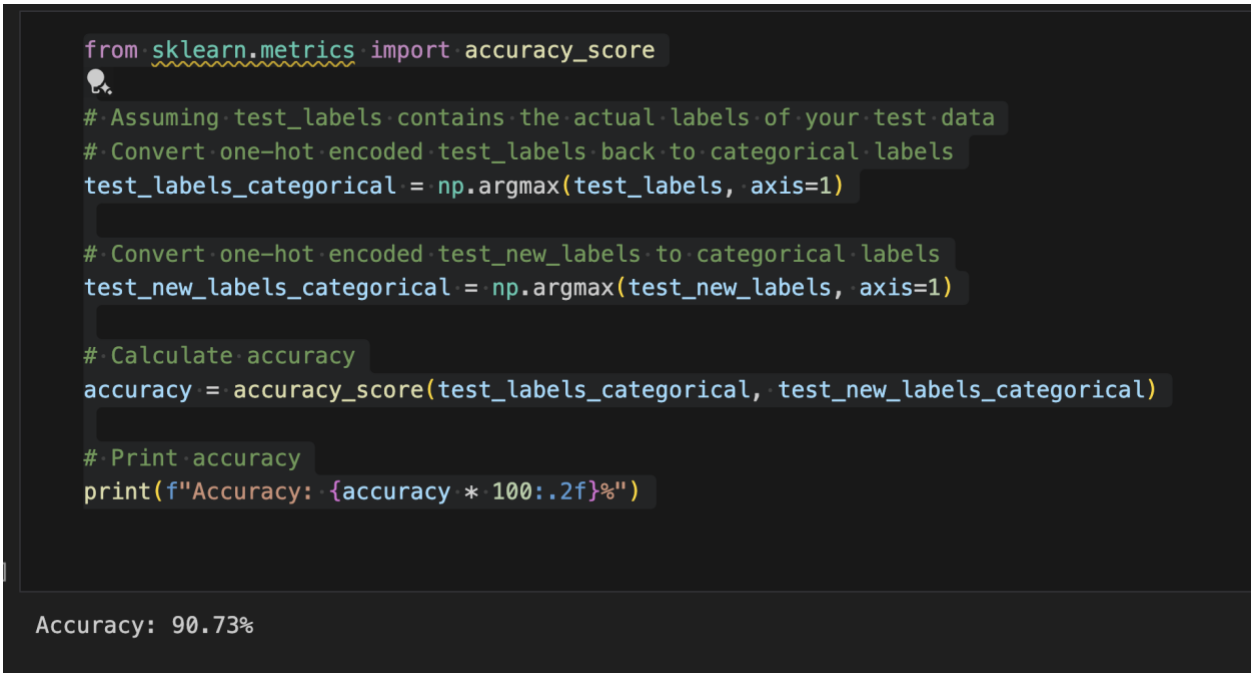
```
# Classification Stage
model.add(Dense(units=128, activation='relu')) # Should be sigmoid but sigmoid has decresed the accuracy thats why we used relu
model.add(Dropout(0.5))
model.add(Dense(num_labels, activation='softmax'))
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x=train_images, y=train_labels, validation_split=0.2, batch_size=32, epochs=5, shuffle=True)

# Save the trained model
model.save('trained_model.h5')
```

```
Epoch 1/5
120/120 [==============================] - 439s 4s/step - loss: 1.0490 - accuracy: 0.5598 - val_loss: 0.6282 - val_accuracy: 0.7296
Epoch 2/5
120/120 [==============================] - 440s 4s/step - loss: 0.6197 - accuracy: 0.7343 - val_loss: 0.3581 - val_accuracy: 0.8800
Epoch 3/5
120/120 [==============================] - 437s 4s/step - loss: 0.2859 - accuracy: 0.8972 - val_loss: 0.2574 - val_accuracy: 0.9081
Epoch 4/5
120/120 [==============================] - 436s 4s/step - loss: 0.2008 - accuracy: 0.9175 - val_loss: 0.1910 - val_accuracy: 0.9186
Epoch 5/5
120/120 [==============================] - 435s 4s/step - loss: 0.2016 - accuracy: 0.9165 - val_loss: 0.2538 - val_accuracy: 0.9050
```

The calculated accuracy indicates the model's ability to correctly classify actor images, yielding an accuracy of 90.73%.



6. Conclusion:

In conclusion, the developed CNN model demonstrates promising results in accurately classifying images of actors Karim AbdelAziz, Hend Sabry, and Hany Ramzy. The methodology outlined in this report encompasses data preparation, model architecture design, training, and evaluation. The achieved accuracy of 90.73% signifies the effectiveness of the proposed approach in actor classification tasks. Further enhancements could involve fine-tuning hyperparameters, exploring alternative architectures, or incorporating data augmentation techniques to improve robustness and generalization.