Artificial Neural Network Final Hackathon Session 2 (January, **2024**)

Furkan Bulut 210316011

Manisa Celal Bayar University Faculty of Engineering

Manisa, TURKEY

210316011@ogr.cbu.edu.tr

***Abstract*— This article introduces a real-time face detection and classification system for videos, emphasizing its dynamic capabilities. Leveraging advanced face detection techniques, the system seamlessly integrates classification using an AlexNet-like model. Demonstrations showcase accurate face detection and its significance in applications like facial recognition and emotion analysis. This article is a concise guide for developers and researchers interested in a dynamic approach to real-time facial analysis.**

I. INTRODUCTION

In the realm of computer vision and artificial intelligence, the task of facial detection and classification has witnessed remarkable advancements, playing a pivotal role in diverse applications such as security systems, entertainment, and human-computer interaction. This article addresses a multifaceted problem—developing a model that not only discerns and classifies the faces of celebrities in a given video but also extends its capabilities to detect and register other faces within the same context.

II. METHODOLOGY

A. *Data Preprocessing*

Data preprocessing is a crucial step in preparing the input data for training a neural network. In this context, we aim to classify images of celebrities' faces into different categories, considering variations in facial sizes. The following methodology outlines the steps taken to preprocess the data effectively:

Fig. 1. Samples From Dataset.

1) Data Loading and Initial Inspection:

Images from different folders, each representing a specific class (e.g., 'jake_small_face,' 'jake_big_face,' 'jimmy_small_face,' 'jimmy_big_face,' 'others'), are loaded using the load_images_from_folder function. The function reads each image, resizes it to a standardized size (default: 256x256), and assigns a corresponding label based on the folder class.

2) Data Consolidation:

The loaded images and labels are consolidated into two NumPy arrays: images and labels. The images array contains the pixel data of the resized images, while the labels array holds the categorical labels in one-hot encoded format.

3) Normalization:

Normalize the pixel values of the images to a standardized range (commonly [0, 1]) to enhance model convergence during training. This is achieved by dividing the pixel values by the maximum pixel value (e.g., 255).

4) Train-Test Split:

Split the dataset into training and testing sets using the train_test_split function from scikit-learn. This ensures an unbiased evaluation of the model's performance.

5) Model Input Shape Adjustment:

Adjust the input shape of the images to comply with the neural network's requirements. In the provided code, the

images are expected to have a shape of (256, 256, 3) since they are resized color images.

The resulting preprocessed dataset is now ready to be fed into the neural network model for training. Effective data preprocessing contributes significantly to the model's ability to learn and generalize patterns from the input images, ultimately enhancing the overall performance of the face classification system.

B. AlexNet-Based Neural Network Architecture

The architecture of the neural network plays a pivotal role in the success of the model for face classification. Here, we present the methodology for creating an AlexNet-inspired model tailored for our specific task of classifying celebrity faces in variable sizes. The following outlines the sequential steps involved in constructing the AlexNet-based model:

1) Convolutional Layers:

Introduce a Convolutional layer with 96 filters, a kernel size of (11, 11), and a stride of (4, 4) to capture complex spatial features from the input images. Apply Rectified Linear Unit (ReLU) activation to introduce non-linearity. Incorporate Batch Normalization for normalizing the activations and enhancing convergence. Employ MaxPooling with a pool size of (3, 3) and a stride of (2, 2) to down-sample the spatial dimensions.

2) Additional Convolutional Layers:

Add subsequent Convolutional layers with varying kernel sizes (5x5, 3x3) and filter sizes (256, 384) to extract hierarchical features at different levels of abstraction. Integrate ReLU activation and Batch Normalization in each Convolutional layer. Use MaxPooling to down-sample the feature maps and maintain spatial hierarchy.

3) Flattening and Dense Layers:

Flatten the high-level feature maps into a one-dimensional vector to serve as the input for fully connected layers. Incorporate Dense (fully connected) layers with 4096 neurons to capture complex relationships within the flattened feature space. Introduce ReLU activation in the Dense layers to enable non-linearity.

4) Early Stopping Integration:

A forward-thinking approach to model training is introduced through the incorporation of EarlyStopping. This vigilant mechanism continually monitors the validation loss during each epoch. Should the loss fail to decrease for five consecutive epochs (configured patience set to 5), the training process is gracefully halted.

4) Culmination of a Robust Model:

As the training epochs unfold, the interplay of training, validation, and early stopping mechanisms culminates in a resilient face classification model. Positioned for real-world applications, this model encapsulates adaptability, resilience, and a profound understanding of facial features, ready to navigate the complexities of diverse datasets.

C. Visualizing Training and Validation Accuracy

In this segment, we embark on a visual journey through the training process of our face classification model, aiming to unravel the intricate dynamics of learning. The extraction and visualization of training and validation accuracy provide a comprehensive narrative, allowing us to glean insights into the model's evolution.

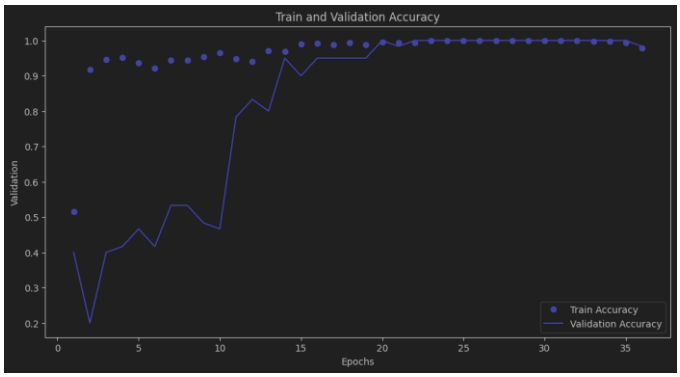

Fig. 2. Train and Validation Accuracy Graph.

1) Extracting Crucial Metrics

The first step involves extracting key metrics from the training history, including the accuracy for both the training and validation sets.

2) Crafting the Visual Narrative:

Armed with our accuracy metrics, we weave a visual narrative that encapsulates the essence of the model's learning journey. The matplotlib library serves as our artistic tool in this endeavor.

D. Fine-Tuning and Model Evaluation

Following the training of the AlexNet-based neural network for celebrity face classification, the model undergoes fine-tuning and rigorous evaluation. A key evaluation tool is the Confusion Matrix, employed to assess the model's ability to classify celebrities with varying facial sizes accurately. The provided Python code snippet demonstrates the transformation of test labels and model predictions, leading to the creation and visualization of the confusion matrix. This heatmap visualization aids in pinpointing areas where the model may face challenges. This succinct process is integral to refining the model for optimal performance, ensuring its adaptability to diverse facial features in real-world scenarios.

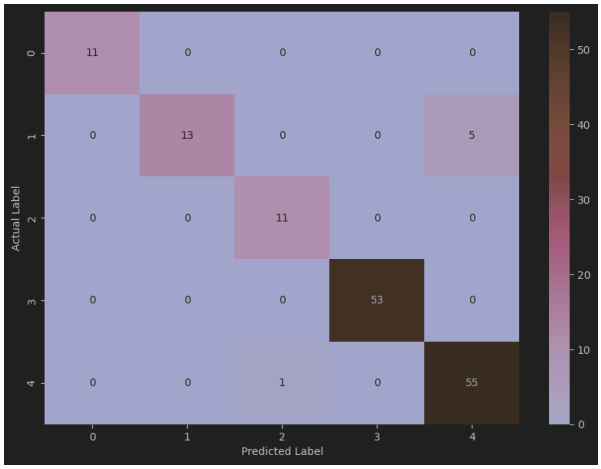

Fig. 3. Confusion Matrix Graph.

E. *Inference on New Images and Result Visualization*

Once the AlexNet-based model has been fine-tuned and evaluated, it becomes valuable to apply the model to new images for real-time classification. The provided Python code illustrates an inference function that utilizes the trained model to detect faces in a new image and predict their categories. The MTCNN (Multi-task Cascaded Convolutional Networks) is employed for face detection, and the model then classifies each detected face. The resulting image is annotated with bounding boxes around the detected faces and labeled with their respective categories. This process facilitates the model's adaptability to diverse facial sizes and further enhances its utility in practical scenarios.

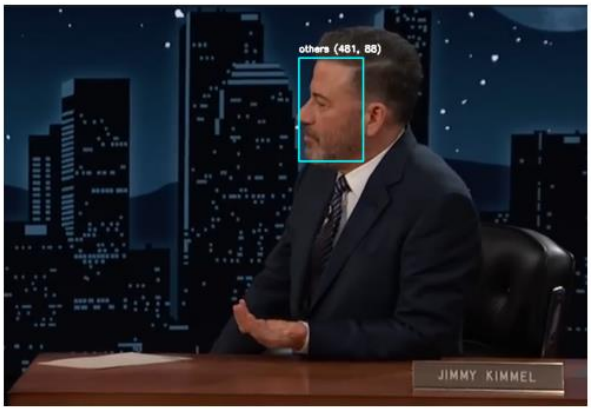

Fig. 4. Sample Output.

detect_faces_and_predict_video that utilizes a pre-trained face detection model (detector) and an AlexNet-based face classification model (alexnet_model) to process frames from a given video file. The function reads each frame, detects faces using the MTCNN face detector, and classifies each detected face using the AlexNet model. The resulting video is displayed in real-time with bounding boxes around detected faces, corresponding labels, and coordinates.

F. *Face Detection and Unique Face Extraction from Video*

A robust face detection and extraction system is implemented to process a video file and save unique faces in a specified output folder. The methodology involves utilizing the OpenCV library to load a pre-trained face detection model and extracting faces from consecutive frames. The face detection process occurs at a defined time interval, enhancing efficiency. Detected faces are cropped from the frames, and a unique identifier based on pixel values ensures only distinct faces are saved. The extracted faces are then stored in the 'others' dataset folder.

III. Discussion

In this article, a real-time face detection and classification system is introduced, emphasizing significant advancements in computer vision and artificial intelligence. The system seamlessly integrates advanced face detection techniques and classification using an AlexNet-like model, showcasing its dynamic capabilities in analyzing celebrity faces within videos. Demonstrations highlight the importance of accurate face detection in applications such as facial recognition and emotion analysis. The article serves as a concise guide for developers and researchers interested in a dynamic approach to real-time facial analysis.

The methodology section outlines crucial steps in data preprocessing and the design of an AlexNet-inspired neural network architecture. Data preprocessing involves loading images, consolidating them into NumPy arrays, normalization, and train-test splitting. The AlexNet-based architecture, tailored for classifying variable-sized celebrity faces, includes convolutional layers, dense layers, and early stopping integration.

The discussion delves into the implications and strengths of the proposed system. Effective data preprocessing is identified as a cornerstone, ensuring well-structured input data for the neural network. The AlexNet-inspired architecture is commended for its thoughtful design in capturing hierarchical features and complex relationships within facial images. Training visualization, fine-tuning, and model evaluation contribute to refining the system, while challenges and potential future directions, such as handling variations in lighting and exploring advanced face detection models, are discussed.

IV. Conclusion

Consequently, the real-time face detection and classification system presented in this article offers a robust solution for dynamic facial analysis in videos. The integration of advanced techniques and an AlexNet-inspired model demonstrates adaptability in recognizing celebrity faces of varying sizes. The systematic methodology, emphasizing effective data preprocessing and neural network architecture, serves as a practical guide. While acknowledging its strengths, challenges such as lighting variations are recognized, urging future enhancements. The system's real-time inference and face extraction capabilities hold promise for diverse applications. This work lays a foundation for ongoing advancements in facial analysis, encouraging further exploration and refinement in computer vision and AI.

```
import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import
Sequential
from tensorflow.keras.layers import
Dense, Flatten, Dropout, Conv2D,
MaxPooling2D, BatchNormalization
from sklearn.model_selection import
train_test_split
from tensorflow.keras.utils import
to_categorical

def load_images_from_folder(folder,
label, image_size=(256, 256)):
    images = []
    labels = []
    for filename in os.listdir(folder):
        img =
cv2.imread(os.path.join(folder,
filename))
        if img is not None:
            img = cv2.resize(img,
image_size)
            images.append(img)
            labels.append(label)
    return images, labels
# Klasör yolları
folders = ['jake_small_face',
'jake_big_face', 'jimmy_small_face',
'jimmy_big_face', 'others']

images = []
labels = []
for i, folder in enumerate(folders):
    imgs, lbls =
load_images_from_folder(folder, i)
    images.extend(imgs)
    labels.extend(lbls)

# Diziye dönüştür ve etiketleri kategorik
formata çevir
images = np.array(images)
labels = to_categorical(labels,
num_classes=5)
def create_alexnet_model(num_classes=5):
    model = Sequential()

    model.add(Conv2D(96, kernel_size=(11,
11), strides=(4, 4), activation='relu',
input_shape=(256, 256, 3)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(3,
3), strides=(2, 2)))

    model.add(Conv2D(256, kernel_size=(5,
5), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(3,
3), strides=(2, 2)))

    model.add(Conv2D(384, kernel_size=(3,
3), activation='relu', padding='same'))
    model.add(Conv2D(384, kernel_size=(3,
3), activation='relu', padding='same'))
    model.add(Conv2D(256, kernel_size=(3,
3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(3,
3), strides=(2, 2)))

    model.add(Flatten())
    model.add(Dense(4096,
activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096,
activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes,
activation='softmax'))

    return model
from tensorflow.keras.callbacks import
EarlyStopping

# Veri setini eğitim ve test setlerine
ayırma
x_train, x_test, y_train, y_test =
train_test_split(images, labels,
test_size=0.2, random_state=42)

# Modeli oluşturma ve derleme
alexnet_model = create_alexnet_model()
alexnet_model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])

#early stop
early_stopping =
EarlyStopping(monitor='val_loss',
patience=5, verbose=1)

# Modeli eğitme
history = alexnet_model.fit(x_train,
y_train, epochs=50, batch_size=32,
validation_split=0.1,
callbacks=[early_stopping])
# Eğitim ve doğrulama doğruluğunu çekme
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(acc) + 1)

# Doğruluk grafiğini çizme
plt.figure(figsize=(12, 6))
plt.plot(epochs, acc, 'bo', label='Train
Accuracy')
plt.plot(epochs, val_acc, 'b',
label='Validation Accuracy')
plt.title('Train and Validation
Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Validation')
```

```
plt.legend()
plt.show()
from sklearn.metrics import
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Eğer y_test one-hot encoded ise, en
büyük indeksi alarak dönüştürün
if y_test.ndim > 1 and y_test.shape[1] >
1:
    y_true = y_test.argmax(axis=1)
else:
    y_true = y_test

# Model tahminlerini dönüştürme
y_pred = alexnet_model.predict(x_test)
y_pred = y_pred.argmax(axis=1)

# Confusion matrix oluşturma
cm = confusion_matrix(y_true, y_pred)

# Confusion matrix'i görselleştirme
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.show()
import cv2
from mtcnn import MTCNN
import matplotlib.pyplot as plt
import numpy as np

def
detect_faces_and_predict_image(image_path
, model):
    image =
cv2.cvtColor(cv2.imread(image_path),
cv2.COLOR_BGR2RGB)
    detector = MTCNN()
    faces = detector.detect_faces(image)

    for face in faces:
        x, y, width, height = face['box']
        cv2.rectangle(image, (x, y),
(x+width, y+height), (0, 256, 256), 2)

        # Kesilmiş yüzü alın
        face_img = image[y:y+height,
x:x+width]
        face_img = cv2.resize(face_img,
(256, 256))
        face_img =
np.expand_dims(face_img, axis=0)
        # Tahmin yapın
        prediction =
model.predict(face_img)

        # Tahmin sonucunu etiketleme
        labels = ['jake_small_face',
'jake_big_face', 'jimmy_small_face',
'jimmy_big_face', 'others']
        predicted_label =
labels[np.argmax(prediction)]

        # Etiketi ve koordinatları
kafanın yanına yazın
        text = f"{predicted_label} ({x},
{y})"
        cv2.putText(image, text, (x, y-
10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,
255, 255), 2)

    plt.imshow(image)
    plt.axis('off')
    plt.show()

# Test etmek istediğiniz görselin yolu ve
modeliniz
test_image_path = "deneme4.png"
detect_faces_and_predict_image(test_image
_path, alexnet_model)
def
detect_faces_and_predict_video(video_path
):

    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print("Video dosyası açılırken
hata oluştu")
        return

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        # Yüz tespiti için görüntüyü
griye çevir
        gray = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)
        faces = detector(gray)

        for face in faces:
            x, y, w, h = face.left(),
face.top(), face.width(), face.height()
            face_img = frame[y:y+h,
x:x+w]
            face_img =
cv2.resize(face_img, (256, 256))  #
Modelinize uygun boyuta ayarlayın
            face_img =
np.expand_dims(face_img, axis=0)

            # Model tahmini
            prediction =
alexnet_model.predict(face_img)

            labels = ['jake_small_face',
'jake_big_face', 'jimmy_small_face',
'jimmy_big_face', 'others']
            predicted_label =
```

```
labels[np.argmax(prediction)]

            # Tahmini çerçevede göster
            cv2.rectangle(frame, (x, y),
(x + w, y + h), (0, 255, 0), 2)

            cv2.putText(frame,
predicted_label, (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9,
(36,255,12), 2)

            # X ve Y koordinatlarını
çerçevede göster
            coordinates_text = f'X: {x},
Y: {y}'
            cv2.putText(frame,
coordinates_text, (x, y + h + 20),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255,
255), 1)

        # Sonuç çerçevesini göster
        cv2.imshow('Video', frame)
        if cv2.waitKey(1) & 0xFF ==
ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

detect_faces_and_predict_video('Test_Vide
o_Jimmy Kimmel-Jodie Foster.mp4')
import cv2
import os
import time

# Load the pre-trained face detection
model from OpenCV
face_cascade =
cv2.CascadeClassifier(cv2.data.haarcascad
es +
'haarcascade_frontalface_default.xml')

# Input video file path
video_path = 'Test_Video_Jimmy Kimmel-
Jodie Foster.mp4'

# Output folder for saving detected faces
output_folder = 'others'

# Create the output folder if it doesn't
exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Initialize a set to store unique
detected faces
unique_faces = set()

# Open the video file
cap = cv2.VideoCapture(video_path)
```

```
# Set the time interval for face
detection (in seconds)
detection_interval = 10  # Detect faces
every 10 seconds
start_time = time.time()

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Get the current time
    current_time = time.time()

    # Convert the frame to grayscale for
face detection
    gray = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)

    # Detect faces every 10 seconds
    if current_time - start_time >=
detection_interval:
        faces =
face_cascade.detectMultiScale(gray,
scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))
        start_time = current_time  #
Update the start time for the next
detection

        for (x, y, w, h) in faces:
            # Crop the detected face
            face = frame[y:y+h, x:x+w]

            # Generate a unique
identifier for the face based on its
pixel values
            face_hash =
hash(face.tobytes())

            # Check if this face is
unique
            if face_hash not in
unique_faces:

unique_faces.add(face_hash)
                # Save the face image to
the 'others' dataset folder
                face_filename =
os.path.join(output_folder,
f'OTHER_{len(unique_faces)}.jpg')

cv2.imwrite(face_filename, face)

cap.release()
cv2.destroyAllWindows()

# Print the number of unique faces
detected
print(f"Number of unique faces detected:
{len(unique_faces)}")
```