جـامعة الجـلالة
GALALA UNIVERSITY

Fall 2024

Face Recognition Using ESP32-CAM

AIE231 || Neural networks

Under supervision of: ***Prof. Amgad Bayoumy***

Project Team members

Name	ID
Amina Omar	222100306
Sherif Farag	222100822
Ziad Abdulrahman	221101043
Aser Mohamed	222102487
Ahmed Mohamed	221100116
Mohamed Ahmed	221101699

Fall 2024

Abstract:

- The project focuses on the development and implementation of a facial recognition system using neural networks, leveraging an ESP32-CAM, o-led, TTL. The initial phase involved testing the system with a celebrity faces dataset to assess and our images dataset then its accuracy and performance. Subsequently, the system was adapted to recognize the user's own faces, incorporating real-time processing for practical application scenarios.
- The project explores various stages of training the neural network, from dataset preprocessing to model evaluation. Additionally, it emphasizes optimizing the hardware components for efficient face detection and recognition, making the system suitable for deployment in diverse environments. The report provides a detailed analysis of the system's architecture, challenges faced during implementation, and the effectiveness of the (CNN) neural network model in accurately identifying faces in real-world conditions patterns and trends within the data.

➢ Introduction and related work:

Facial recognition technology is increasingly used in fields like security and authentication, driven by advances in deep learning and neural networks. This project aims to develop a facial recognition system using affordable hardware components, such as the ESP32-CAM, Oled, and TTL, focusing on real-time performance and accuracy.

Traditional facial recognition methods, like Eigenfaces and Fisherfaces, were effective in controlled environments but struggled with variations in lighting, pose, and expression. The advent of deep learning, especially convolutional neural networks (CNNs), has significantly improved facial recognition, allowing systems to handle complex variations in real-world scenarios.

This project follows a similar approach, combining the ESP32-CAM with a Oled display to create a practical and cost-effective facial recognition system. While previous works have demonstrated real-time face recognition with limited resources, this project aims to further optimize the balance between processing power, model accuracy, and hardware constraints.

➢ Aim:

The aim of this project is to develop a cost-effective facial recognition system using an ESP32-CAM, LCD, TTL. The system will be designed to perform real-time face detection and recognition with high accuracy, leveraging neural networks for effective classification. By integrating affordable hardware and deep learning techniques, the project aims to provide a practical solution for facial recognition in various applications while addressing challenges related to hardware limitations and real-time processing.

➢ Objectives:

1. **Design and Implement Facial Recognition System:** Develop a real-time facial recognition system using the ESP32-CAM, and OLED display, focusing on cost-effectiveness and ease of use.
2. **Train a Neural Network Model:** Utilize a neural network, specifically a convolutional neural network (CNN) to train the model for accurate face detection and recognition.
3. **Integrate Hardware Components:** Seamlessly integrate the ESP32-CAM for face capture, and OLED for displaying the results in real-time.
4. **Optimize System Performance:** Ensure the system performs face detection and recognition efficiently with limited computational resources, balancing accuracy and processing speed.

5. **Test and Evaluate Accuracy:** Assess the system's performance by testing it on a diverse set of faces and evaluating the recognition accuracy and processing time.

➢ Scope / Applicability:

Scope: The system is applicable to security applications, user authentication systems, and small-scale surveillance solutions, providing a cost-effective and efficient method for real-time facial recognition.

• **Applicability:**

- **Individuals:** Personal security applications such as home automation or user access control.
- **Small Businesses:** Simple security systems for controlling access and monitoring premises.
- **Educational Institutions:** Use for student identification and attendance tracking in classrooms or events.
- **Developers/DIY Enthusiasts:** Accessible platform for building custom facial recognition projects using affordable hardware.

➢ Purpose:

Develop a cost-effective facial recognition system that utilizes affordable hardware components, such as the ESP32-CAM, and an OLED display, to perform real-time face detection and recognition. The system aims to demonstrate the feasibility of implementing facial recognition technology using limited computational resources while maintaining high accuracy and efficiency.

By integrating deep learning techniques with low-cost hardware, the project seeks to provide a practical solution for personal, educational, and small-scale security applications, where budget constraints may limit the use of more expensive systems.

➢ Solution (Proposal):

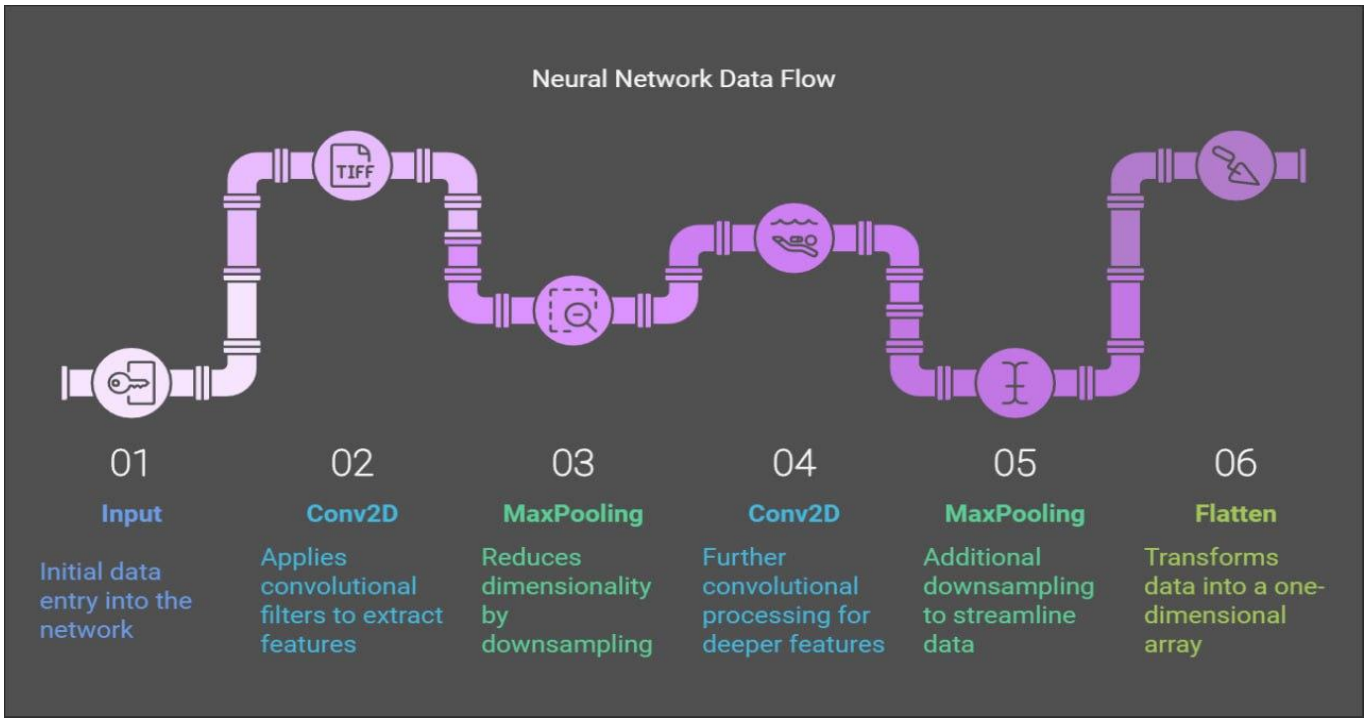
- Data Collection: Capturing facial images using the ESP32-CAM module.
- Preprocessing: Performing face detection to isolate faces from the background and ensure the images are standardized for model input.
- Model Training: Training a convolutional neural network (CNN) on labeled face data to recognize distinct facial features.
- Recognition: Matching the captured face with stored data to identify individuals.
- Display Results: Displaying the recognition results on the connected OLED screen.
- Optimization: Refining the model to balance accuracy and processing time, considering hardware limitations.
- Testing and Evaluation: Testing the system under different conditions and evaluating its accuracy and performance.

➢ Concepts of neural networks:

1. **Convolutional Neural Networks (CNNs):** A deep learning model designed for processing and recognizing visual data, particularly useful in facial recognition tasks.
2. **Face Detection:** The process of identifying and locating human faces in images, which is essential for isolating faces before recognition.
3. **Feature Extraction:** Extracting unique facial features (such as eyes, nose, and mouth) from images to form a representation for identification.
4. **Activation Functions:** Mathematical functions like ReLU (Rectified Linear Unit) used in CNN layers to introduce non-linearity, helping the network learn complex patterns.

5. **Backpropagation:** The process of adjusting the weights of the neural network based on the error between predicted and actual outputs during training.
6. **Pooling:** A technique used in CNNs to reduce the dimensionality of feature maps, which helps speed up processing and reduce computation.
7. **Loss Function:** A function that measures the difference between the predicted output and the actual label, guiding the network's training process.

➢ Implementation (Process):



➢ Resources Used:

- **Programming Language:** Python
- **Libraries:** OpenCV, TensorFlow/Keras, NumPy, Matplotlib.
- **Tools:**ESP32-CAM, OLED display, Jupiter Notebook for initial model development
- **Data Source:** Celebrity Faces dataset for initial testing, followed by pre-made and custom dataset of user faces for real-world evaluation

➢ Dataset Used:

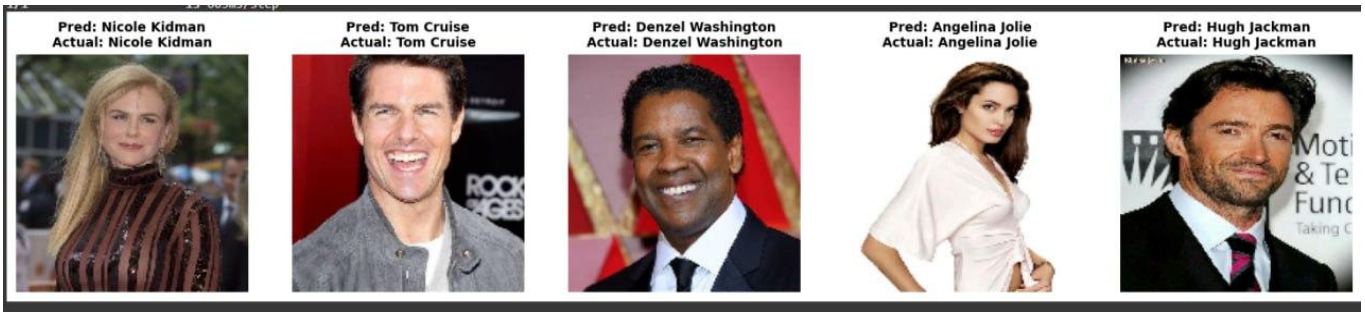
Celebrity Faces Dataset:

- Used for initial testing and evaluation of the facial recognition system.
- Provided a diverse set of facial images for training and assessing the model's performance on known faces.

➢ **Custom Our Images Faces Dataset:**

- Contains real-world facial data of project participants for personalized testing and validation.

Premade dataset :



Our Custom Dataset:

➢ main code:

```
import zipfile

# Path to the downloaded ZIP file
zip_file_path = 'Celebrity Faces Dataset.zip'
extract_folder = '/content/celebrity_faces_dataset/'

# Extract the ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_folder)

print(f"Dataset extracted to: {extract_folder}")
```

```
Dataset extracted to: /content/celebrity_faces_dataset/
```

```
import os
import shutil
import random

# Define paths to the dataset
dataset_dir = '/content/celebrity_faces_dataset/Celebrity Faces Dataset'
train_dir = os.path.join(dataset_dir, 'train')
validation_dir = os.path.join(dataset_dir, 'validation')

# Create the train and validation directories if they don't exist
os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)

# Get the list of all celebrity directories
celebrity_dirs = [os.path.join(dataset_dir, celebrity) for celebrity in os.listdir(dataset_dir)
                  if os.path.isdir(os.path.join(dataset_dir, celebrity))]

# Loop through each celebrity folder to split the images
for celebrity_dir in celebrity_dirs:
    # Create subdirectories in train and validation directories for each celebrity
    os.makedirs(os.path.join(train_dir, os.path.basename(celebrity_dir)), exist_ok=True)
    os.makedirs(os.path.join(validation_dir, os.path.basename(celebrity_dir)), exist_ok=True)

    # Get all images in the celebrity folder
    images = [os.path.join(celebrity_dir, image) for image in os.listdir(celebrity_dir) if image.endswith('.jpg')]

    # Shuffle the images
    random.shuffle(images)

    # Split the images (80% for training and 20% for validation)
    split_index = int(0.8 * len(images))

    train_images = images[:split_index]
    validation_images = images[split_index:]

    # Move the images to the respective directories
    for train_image in train_images:
        shutil.move(train_image, os.path.join(train_dir, os.path.basename(celebrity_dir), os.path.basename(train_image)))

    for validation_image in validation_images:
        shutil.move(validation_image, os.path.join(validation_dir, os.path.basename(celebrity_dir), os.path.basename(validation_image)))

print("Data split into train and validation directories successfully.")
```

```
Data split into train and validation directories successfully.
```

```
import os
import numpy as np
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
```

```
import os

# Check if the train and validation directories exist
train_dir = '/content/celebrity_faces_dataset/Celebrity Faces Dataset/train/'
validation_dir = '/content/celebrity_faces_dataset/Celebrity Faces Dataset/validation/'

print("Train directory exists:", os.path.exists(train_dir))
print("Validation directory exists:", os.path.exists(validation_dir))

# List the contents of the train and validation directories
print("Train directory contents:", os.listdir(train_dir) if os.path.exists(train_dir) else "Train directory not found")
print("Validation directory contents:", os.listdir(validation_dir) if os.path.exists(validation_dir) else "Validation directory not found")
```

```
Train directory exists: True
Validation directory exists: True
Train directory contents: ['train', 'Sandra Bullock', 'Scarlett Johansson', 'Kate Winslet', 'Natalie Portman', 'Johnny Depp', 'Brad Pitt', 'Ni
Validation directory contents: ['train', 'Sandra Bullock', 'Scarlett Johansson', 'Kate Winslet', 'Natalie Portman', 'Johnny Depp', 'Brad Pitt'
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the path for train and validation directories
train_dir = '/content/celebrity_faces_dataset/Celebrity Faces Dataset/train/'
validation_dir = '/content/celebrity_faces_dataset/Celebrity Faces Dataset/validation/'

# Set up ImageDataGenerator for rescaling the images (normalization)
train_datagen = ImageDataGenerator(rescale=1./255)  # Normalize the images to [0,1]
validation_datagen = ImageDataGenerator(rescale=1./255)

# Set up the data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),  # Resize images to the required input size for the model
    batch_size=32,
    class_mode='categorical'  # Since we have multiple classes (celebrities)
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 1791 images belonging to 19 classes.
Found 884 images belonging to 19 classes.
```

```python
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Initialize the model
model = Sequential()

# First Convolutional Layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Second Convolutional Layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Third Convolutional Layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening Layer
model.add(Flatten())

# Fully Connected Layer
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))

# Output Layer
model.add(Dense(19, activation='softmax'))  # 19 classes for the celebrities

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Summarize the model
model.summary()
```

```python
# Calculate steps per epoch and validation steps
steps_per_epoch = (train_generator.samples + train_generator.batch_size - 1) // train_generator.batch_size
validation_steps = (validation_generator.samples + validation_generator.batch_size - 1) // validation_generator.batch_size

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=steps_per_epoch,  # Ensure all samples are included
    epochs=50,  # Adjust the number of epochs as needed
    validation_data=validation_generator,
    validation_steps=validation_steps  # Ensure all validation samples are included
)
```

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

```
plt.figure(figsize=(12, 6))

# Plot training accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()

# Heatmap for Accuracy and Loss Correlation
# Convert accuracy and loss history to numpy arrays for correlation
accuracy = np.array([history.history['accuracy'], history.history['val_accuracy']])
loss = np.array([history.history['loss'], history.history['val_loss']])

# Create a heatmap of the accuracy and loss correlation
plt.figure(figsize=(8, 6))
sns.heatmap(np.corrcoef(accuracy, loss), annot=True, cmap='coolwarm', xticklabels=['Train Accuracy', 'Validation Accuracy', 'Train Loss', 'Validation Loss'], yticklabels=['Train Accuracy', 'Validation Accuracy', 'Train Loss', 'Validation Loss'])
plt.title('Heatmap of Accuracy and Loss Correlation')
plt.show()

# Histogram for Accuracy and Loss distribution
plt.figure(figsize=(12, 6))

# Plot histogram for Training and Validation Accuracy
plt.subplot(1, 2, 1)
plt.hist(history.history['accuracy'], bins=10, alpha=0.7, label='Training Accuracy')
plt.hist(history.history['val_accuracy'], bins=10, alpha=0.7, label='Validation Accuracy')
plt.title('Histogram of Accuracy')
plt.xlabel('Accuracy')
plt.ylabel('Frequency')
plt.legend()

# Plot histogram for Training and Validation Loss
plt.subplot(1, 2, 2)
plt.hist(history.history['loss'], bins=10, alpha=0.7, label='Training Loss')
plt.hist(history.history['val_loss'], bins=10, alpha=0.7, label='Validation Loss')
plt.title('Histogram of Loss')
plt.xlabel('Loss')
plt.ylabel('Frequency')
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()
```

```
from tensorflow.keras.models import load_model

# Load the pre-trained model
model = load_model('trained_cnn_model.h5')

# Verify the model structure
model.summary()
```

```
# Evaluate the model on validation data
validation_loss, validation_accuracy = model.evaluate(validation_generator)
print(f"Validation Loss: {validation_loss}")
print(f"Validation Accuracy: {validation_accuracy}")
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Get a batch of images from the validation generator
validation_images, validation_labels = next(validation_generator)

# Predict on the batch
predictions = model.predict(validation_images)

# Get the mapping of class indices to celebrity names
class_indices = validation_generator.class_indices
class_names = {v: k for k, v in class_indices.items()}  # Reverse the dictionary to map from index to name

# Display predictions and actual labels with improved visualization
fig, axes = plt.subplots(1, 5, figsize=(15, 3))
for i, ax in enumerate(axes):
    ax.imshow(validation_images[i])
    predicted_class_index = np.argmax(predictions[i])
    actual_class_index = np.argmax(validation_labels[i])

    # Get the celebrity name from the index
    predicted_name = class_names[predicted_class_index]
    actual_name = class_names[actual_class_index]

    # Set the title with a cleaner format
    ax.set_title(f'Pred: {predicted_name}\nActual: {actual_name}', fontsize=10, fontweight='bold')
    ax.axis('off')

# Display the plot
plt.tight_layout()  # This adjusts the spacing between subplots for better readability
plt.show()
```

➢ output and visualization:

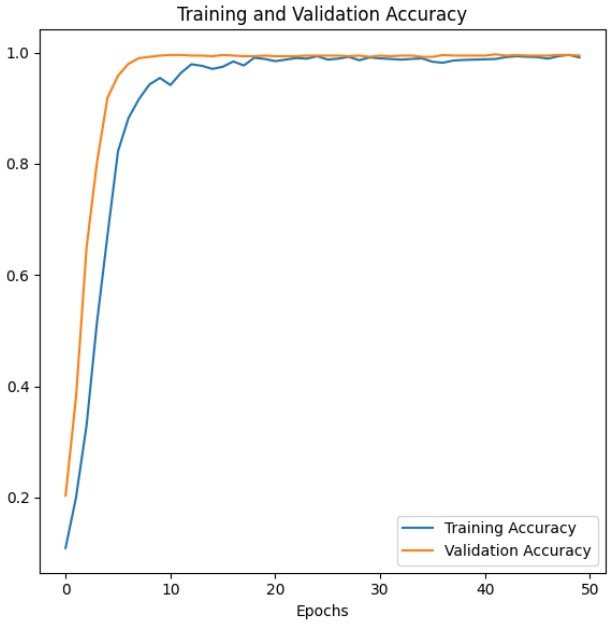
```
Found 1791 images belonging to 19 classes.
Found 884 images belonging to 19 classes.
```

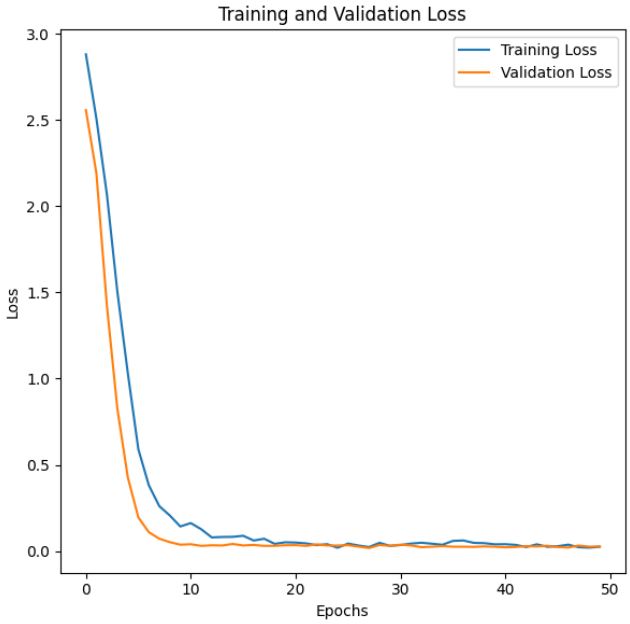
Model: "sequential"

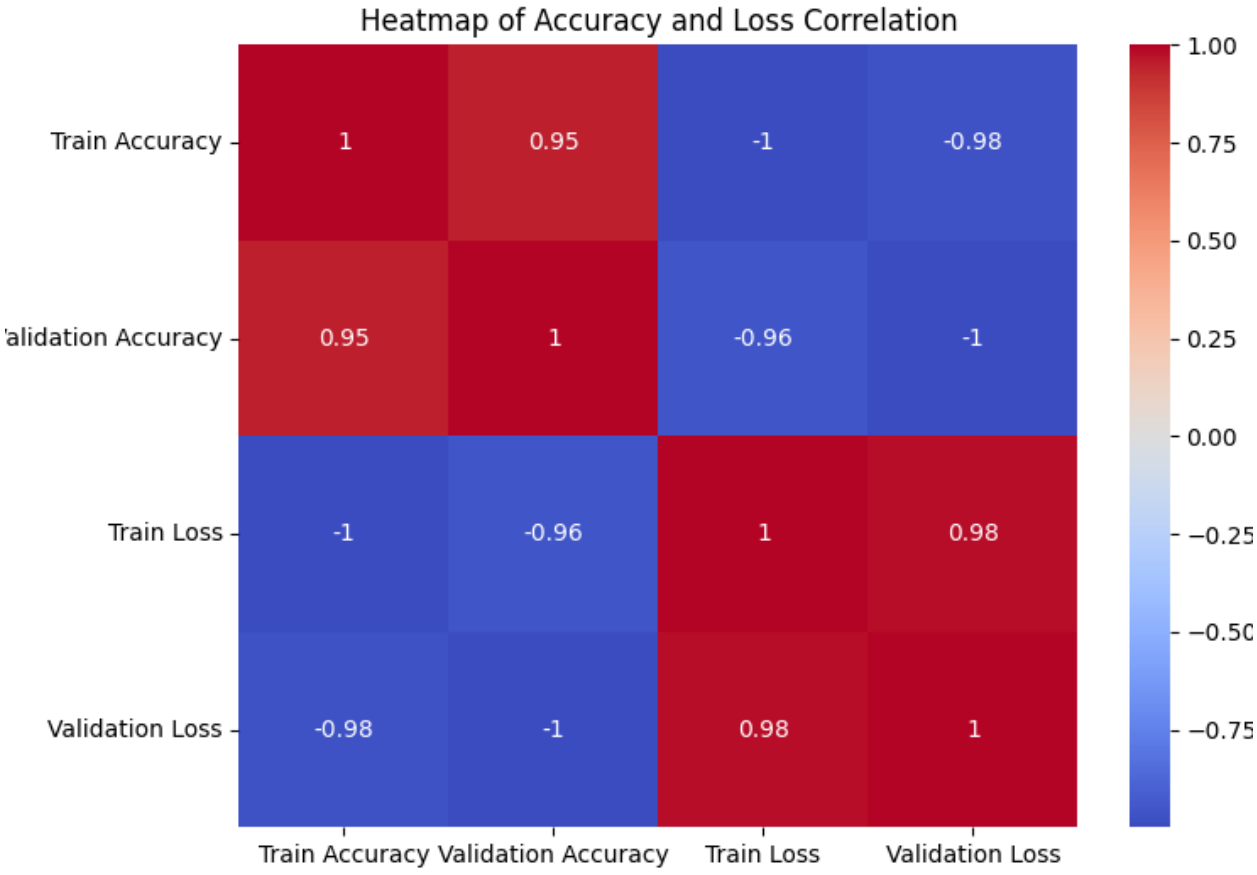
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
flatten (Flatten)	(None, 36992)	0
dense (Dense)	(None, 512)	18,940,416
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 19)	9,747

Total params: 19,043,411 (72.64 MB)
Trainable params: 19,043,411 (72.64 MB)
Non-trainable params: 0 (0.00 B)

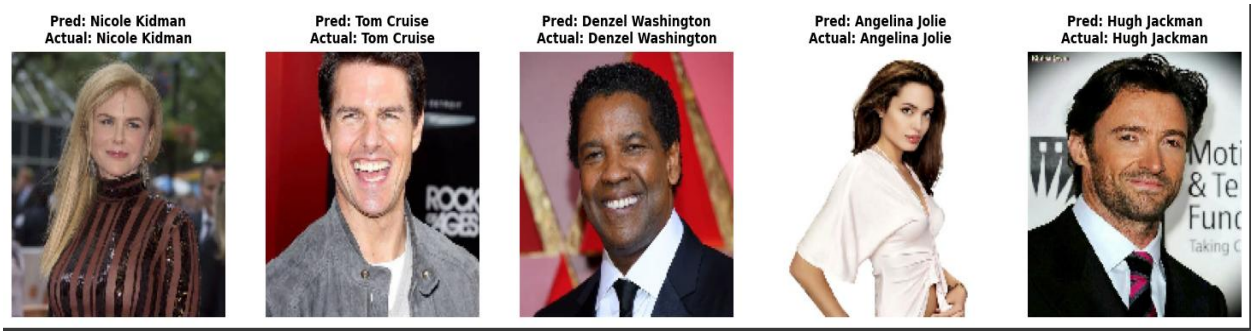
```
Epoch 1/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 137s 2s/step - accuracy: 0.0903 - loss: 3.0618 - val_accuracy: 0.2036 - val_loss: 2.5569
Epoch 2/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 126s 2s/step - accuracy: 0.1951 - loss: 2.5579 - val_accuracy: 0.3812 - val_loss: 2.1904
Epoch 3/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 152s 2s/step - accuracy: 0.3199 - loss: 2.1402 - val_accuracy: 0.6493 - val_loss: 1.4240
Epoch 4/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.5117 - loss: 1.5427 - val_accuracy: 0.8032 - val_loss: 0.8242
Epoch 5/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 143s 2s/step - accuracy: 0.6812 - loss: 0.9963 - val_accuracy: 0.9186 - val_loss: 0.4264
Epoch 6/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.8079 - loss: 0.6428 - val_accuracy: 0.9581 - val_loss: 0.1953
Epoch 7/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 125s 2s/step - accuracy: 0.8748 - loss: 0.3886 - val_accuracy: 0.9796 - val_loss: 0.1094
Epoch 8/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.9010 - loss: 0.3018 - val_accuracy: 0.9898 - val_loss: 0.0719
Epoch 9/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 144s 2s/step - accuracy: 0.9522 - loss: 0.1977 - val_accuracy: 0.9921 - val_loss: 0.0518
Epoch 10/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.9485 - loss: 0.1678 - val_accuracy: 0.9943 - val_loss: 0.0372
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Epochs


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epochs

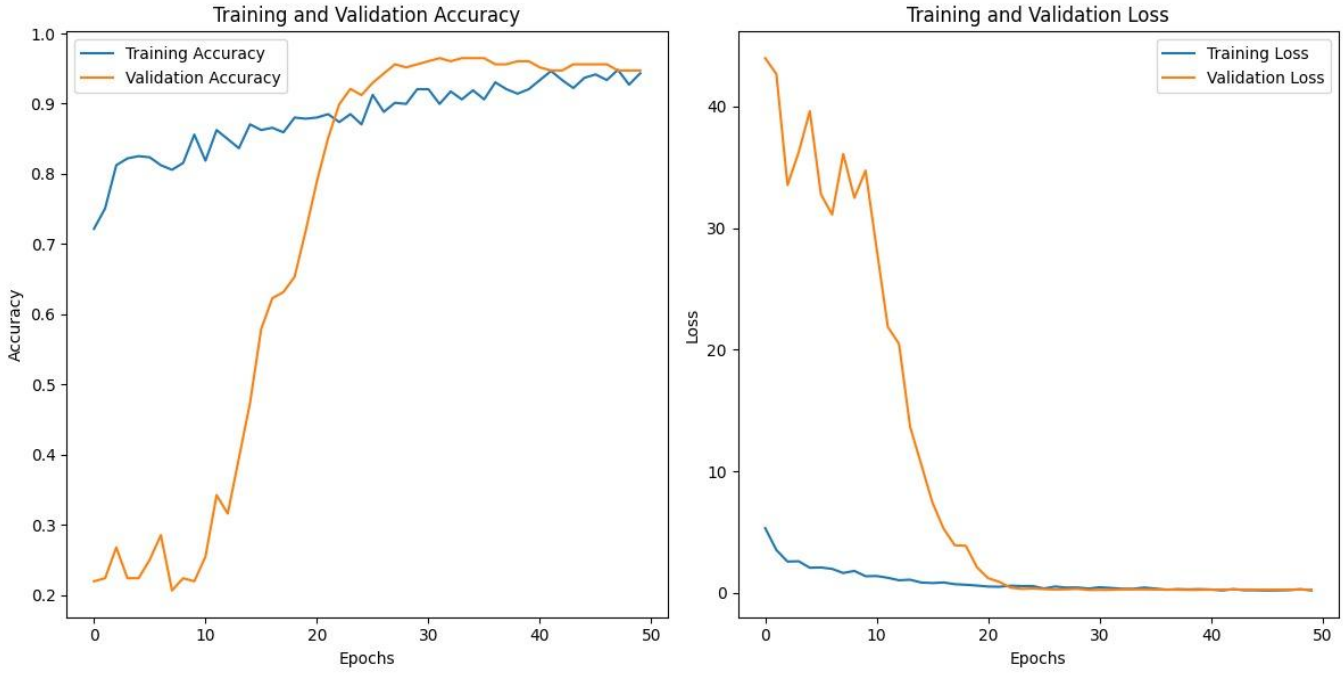

Heatmap of Accuracy and Loss Correlation
Train Accuracy
alidation Accuracy
Train Loss
Validation Loss
1
0.95
-1
-0.98
0.95
1
-0.96
-1
-1
-0.96
1
0.98
-0.98
-1
0.98
1
Train Accuracy
Validation Accuracy
Train Loss
Validation Loss

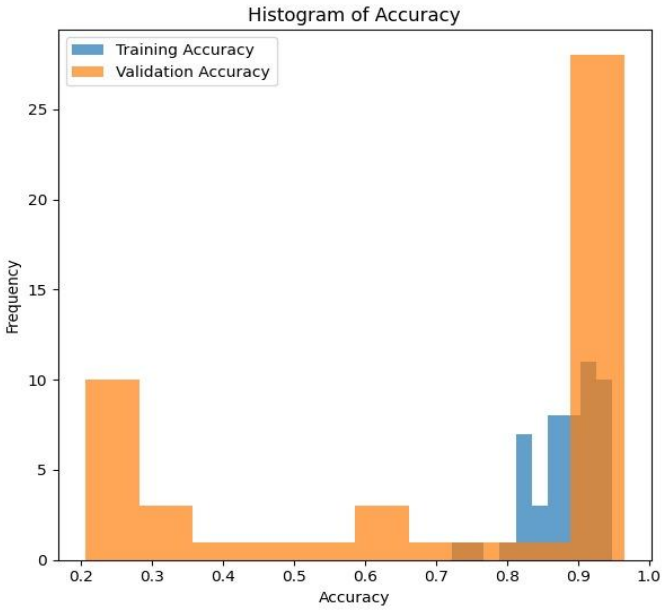
```
28/28 ━━━━━━━━━━━━━━━━━━━━ 24s 822ms/step - accuracy: 0.9972 - loss: 0.0123
Validation Loss: 0.026977149769663 81
Validation Accuracy: 0.9943438768386841
```

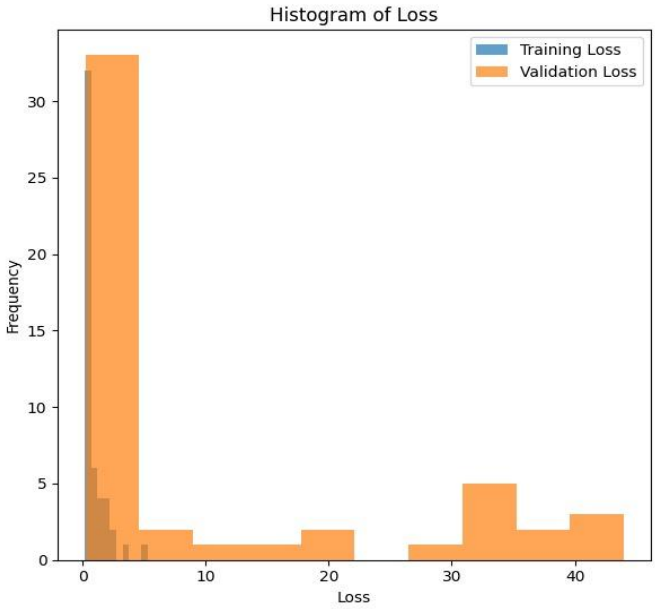


➢ Output & Visualization for the custom dataset:

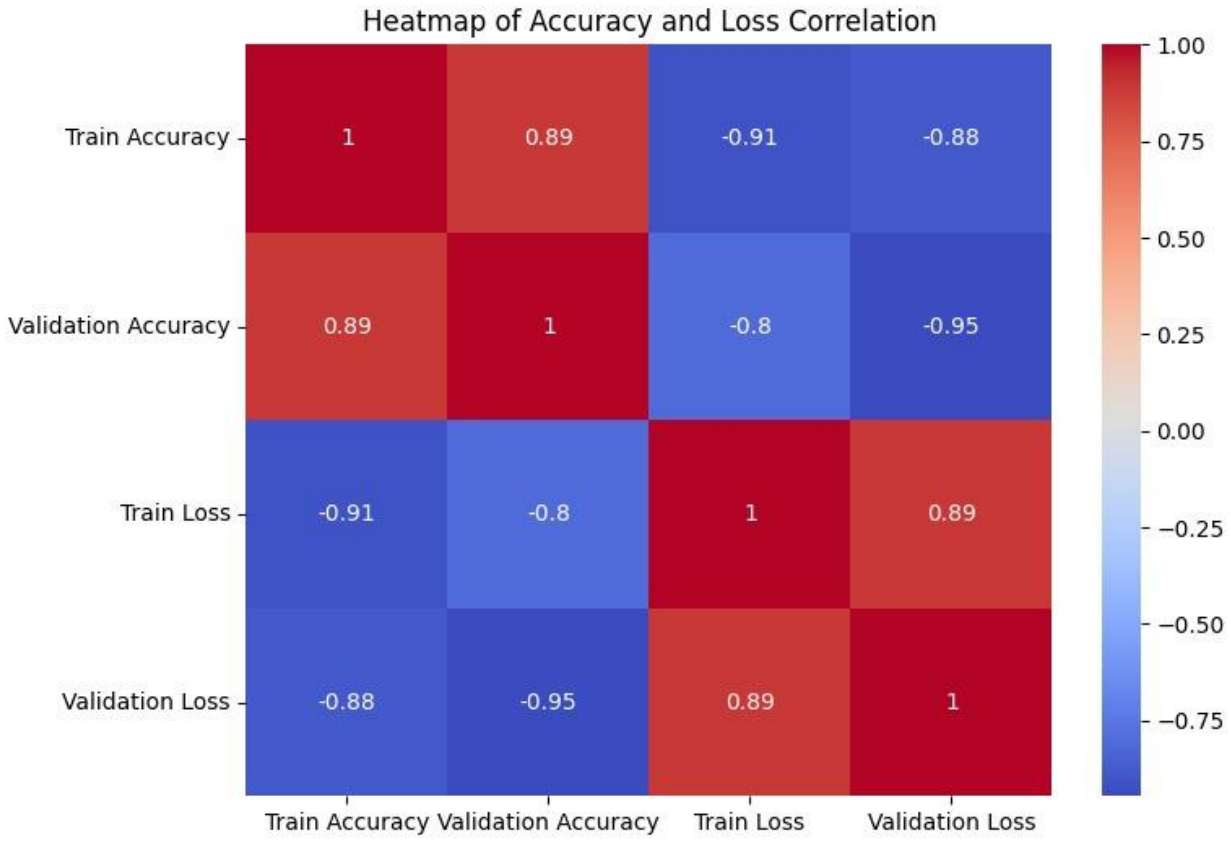
Training and Validation curves:




Histogram of Accuracy
Training Accuracy
Validation Accuracy
Frequency
Accuracy
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0
5
10
15
20
25


Histogram of Loss
Training Loss
Validation Loss
Frequency
Loss
0
10
20
30
40
0
5
10
15
20
25
30

Heatmap:


Heatmap of Accuracy and Loss Correlation
Train Accuracy
Validation Accuracy
Train Loss
Validation Loss
1
0.89
-0.91
-0.88
0.89
1
-0.8
-0.95
-0.91
-0.8
1
0.89
-0.88
-0.95
0.89
1
Train Accuracy
Validation Accuracy
Train Loss
Validation Loss
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

➢ conclusion :

The project demonstrates the potential of using cost-effective hardware like ESP32-CAM for facial recognition tasks. By integrating neural networks, we successfully classified faces and laid the groundwork for practical applications. Moving forward, our vision of creating a drone-based smart attendance system highlights the innovative possibilities of combining AI and IoT with further development, this solution can transform how attendance is managed, making it more efficient, scalable.

➢ References :

1. **KoreaScience. (2022). IoT system based on face recognition technology using ESP32-CAM. Retrieved from https://koreascience.kr/article/JAKO202223862715662.page**
2. **ELA Journal. (n.d.). Face recognition and monitoring using ESP32-CAM. Retrieved from https://ela.kpi.ua/items/a41c04d1-f3be-47a2-9a61-225d04253f33**
3. **IEEE Xplore. (2022). Design of an IoT system based on face recognition technology using ESP32-CAM. Retrieved from https://ieeexplore.ieee.org/abstract/document/10303506/**
4. **ResearchGate. (2022). Design of an IoT system based on face recognition technology using ESP32-CAM. Retrieved from https://www.researchgate.net/publication/361877428_Design_of_an_IOT_System_based_on_Face_Recognition_Technology_using_ESP32-CAM**
5. **ResearchGate. (n.d.). Face recognition using ESP32-CAM for real-time tracking and monitoring. Retrieved from https://www.researchgate.net/publication/380024424_Face_Recognition_Using_ESP32-Cam_for_Real-Time_Tracking_and_Monitoring**
6. **ICICELB. (2022). Face recognition using ESP32-CAM. Retrieved from http://www.icicelb.org/ellb/contents/2022/3/elb-13-03-12.pdf**
7. **Garuda. (2022). A performance evaluation of ESP32 camera face recognition for various projects. Retrieved from http://download.garuda.kemdikbud.go.id/article.php?article=3202067&val=28194&title=A%20Performance%20evaluation%20of%20ESP32%20Camera%20Face%20Recognition%20for%20various%20projects**