Hand Gesture Recognition System

Under supervision of Eng. Baraa

Group Members:
Mariam Tarek
Fatma Ashraf
Romisaa Hassan
Mahmoud Osama
Yousef Elsayed

Table Of Contents:

Abstract

The Hand Gesture Recognition System leverages deep learning and computer vision to enable real-time hand gesture recognition, facilitating applications in human-computer interaction (HCI), virtual reality (VR), and accessibility tools. This report documents the entire project lifecycle, from data collection and preprocessing to model development, real-time deployment, MLOps implementation, and continuous monitoring. Key challenges, solutions, and the system's impact are discussed, alongside potential applications and recommendations for future improvements.

Introduction

The Hand Gesture Recognition System aims to create an interactive model that recognizes hand gestures in real-time using a webcam feed. By combining convolutional neural networks (CNNs) and computer vision techniques, the system enables seamless interaction between users and devices, with applications in HCI, VR, and accessibility for individuals with disabilities. This report details the five milestones of the project, addressing data collection, model development, real-time implementation, deployment, MLOps, and monitoring, as well as challenges, solutions, and future directions.

Environmental Preparing

1. Requirement

tensorflow>=2.13.0
opencv-python>=4.8.0
numpy>=1.23.0
pandas>=1.5.0
scikit-learn>=1.1.0
matplotlib>=3.6.0
seaborn>=0.12.0
mlflow>=2.0.0
jupyter>=1.0.0
imageio>=2.31.0
albumentations>=1.3.0 # Replace imgaug with albumentations
Pillow>=9.0.0

Milestone 1

1. Objective

Collect and preprocess a dataset of hand gestures to prepare for model training.

2. Tasks and Implementation

- **Data Collection**: Created a custom dataset by capturing images of hand gestures using a webcam. The dataset includes gestures collected under varied lighting and background conditions. A Python script utilizing OpenCV was developed to capture 5,000 images per gesture.
- **Preprocessing**: Implemented a preprocessing pipeline to prepare captured images for model training. The script loads images from the raw data directory, resizes them to 64x64 pixels, and normalizes pixel values to the range [0,1] by dividing by 255. It handles errors such as invalid files or memory issues by attempting to load images at reduced resolution. Processed images are saved as PNG files with filenames indicating the gesture and processing status.
- **Data Augmentation**: Integrated data augmentation within the preprocessing pipeline using TensorFlow's ImageDataGenerator. For each normalized image, two augmented versions are generated with random rotations (±10°) and horizontal flipping, using the 'nearest' fill mode. Augmented images are saved with an 'aug' suffix (e.g., hello_processed_aug_1.png), increasing dataset diversity to prevent overfitting.

3. Challenges and Solutions

- **Challenge**: Permission errors when creating directories or saving images due to restricted folder access on Windows.
  Solution: The script includes error handling for Permission Error and System Error, prompting the user to check permissions or run as administrator.
- **Challenge**: Potential webcam access issues, which could halt data collection.
  Solution: The script checks if the webcam is accessible using cap.isOpened() and exits with an error message if it fails, guiding the user to troubleshoot hardware issues.

4. Deliverables

- Custom dataset of raw hand gesture images (e.g., for the "hello" gesture), captured using the script
- Preprocessed and augmented dataset, including resized (64x64) and normalized images, with two augmented versions per original image (rotations and flips), saved as PNG files.

Milestone 2

2. Objectives:

Develop and train a model for recognizing hand gestures.

3. Tasks and Implementation

- **Model Selection**: Selected MobileNetV2, a lightweight pre-trained convolutional neural network (CNN), as the base model due to its efficiency on resource-constrained devices. The model uses pre-trained weights from ImageNet, with the top layers removed and replaced by a custom head consisting of global average pooling, a 128-unit dense layer with ReLU activation, a 50% dropout layer, and a SoftMax output layer for five gesture classes (hello, yes, no, I love you, thank you).
- **Model Training**: Trained the model in two phases using a custom data generator that loads pre-processed 64x64 images from the processed dataset directory. In the first phase, the base MobileNetV2 layers were frozen, and the model was trained for 10 epochs with a batch size of 64, using the Adam optimizer (learning rate 0.001) and categorical cross-entropy loss. In the second phase (fine-tuning), the last 20 layers of MobileNetV2 were unfrozen, and the model was trained for another 10 epochs with a reduced learning rate of 0.0001. The dataset was split into 80% training and 20% validation, with steps per epoch and validation steps calculated based on the total number of images.
- **Model Evaluation**: Evaluated the model on a validation set of up to 5,000 images using metrics including accuracy, precision, recall, and F1-score (weighted averages). Generated a confusion matrix to analyse misclassifications across the five gesture classes. Plotted training and validation accuracy and loss curves over the 20 epochs, to assess model performance and convergence.
- **Model Optimization**: Optimized the model through fine-tuning by unfreezing the last 20 layers of MobileNetV2 to improve feature extraction, reducing the learning rate to 0.0001 to prevent large weight updates. Added a 50% dropout layer to reduce overfitting. The model was saved with error handling for permission issues during saving.

4. Challenges and Solutions

- **Challenge**: Potential overfitting due to limited dataset diversity or model complexity. **Solution**: Included a 50% dropout layer in the model architecture to regularize the network and reduce overfitting. Data augmentation from Milestone 1 (rotations and flips) further increased dataset diversity.
- **Challenge**: Invalid or corrupted image files in the dataset causing errors during training. **Solution**: The custom data generator filters invalid filenames, skips files that fail to load,

and logs warnings, ensuring only valid images are used. It also handles exceptions during image loading to prevent training interruptions.

- **Challenge**: Permission errors when saving the trained model due to restricted folder access on Windows.
  Solution: The model saving process includes error handling for Permission Error and System Error, prompting the user to check permissions or run as administrator, with directories created if needed.

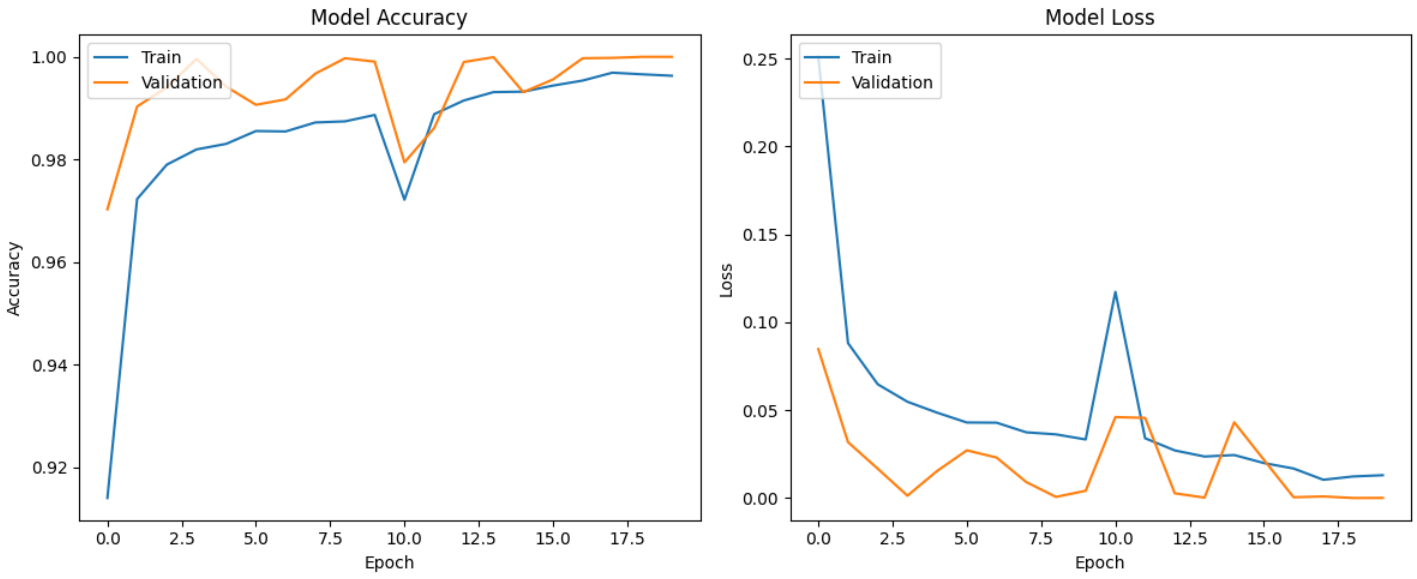
5. Deliverables

- Trained MobileNetV2 model for hand gesture recognition, capable of classifying five gestures (hello, yes, no, I love you, thank you).
- Model evaluation report including validation accuracy, precision, recall, F1-score, a confusion matrix, and training/validation accuracy and loss plots.

```
# Define the model using MobileNetV2
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
base_model.trainable = False

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```



Milestone 3

1. Objectives:

Implement the real-time gesture recognition system and deploy it.

2. Tasks and Implementation:

- **Real-Time Gesture Recognition**: Developed a Python script using OpenCV and TensorFlow to perform real-time hand gesture recognition via webcam. The script loads the trained MobileNetV2 model and processes webcam frames to classify five gestures (hello, yes, no, i love you, thank you). Frames are converted to HSV color space for skin detection, and the largest contour (assumed to be the hand) is identified to extract a hand region. The hand region is resized to 64x64 pixels, normalized to [0,1], and fed into the model for prediction. Predictions with confidence above 40% are displayed on the frame with the gesture label and confidence score (e.g., "hello (75.2%)"), while low-confidence predictions or no-hand scenarios are indicated. The system calculates and displays frames per second (FPS) for performance monitoring.
- **Deployment**: Deployed the real-time recognition system locally using two approaches. First, the real-time recognition script runs standalone, displaying gesture predictions in a local OpenCV window titled "Real-Time Gesture Recognition." Second, implemented a Flask-based web application, to provide a user-friendly interface. The Flask app loads the same MobileNetV2 model and serves a web page (index.html) with JavaScript, CSS, and HTML components to capture webcam frames or uploaded images, preprocess them (using HSV skin detection, resizing to 64x64, and normalization), and display gesture predictions (e.g., "hello" with confidence). The app supports two input methods: file uploads via HTTP POST to /predict and real-time webcam frames sent as base64-encoded JSON. Predictions are returned as JSON with the gesture label and confidence, displayed on the web interface. The Flask server runs locally in debug mode, with error handling for model loading, invalid inputs, and prediction failures.

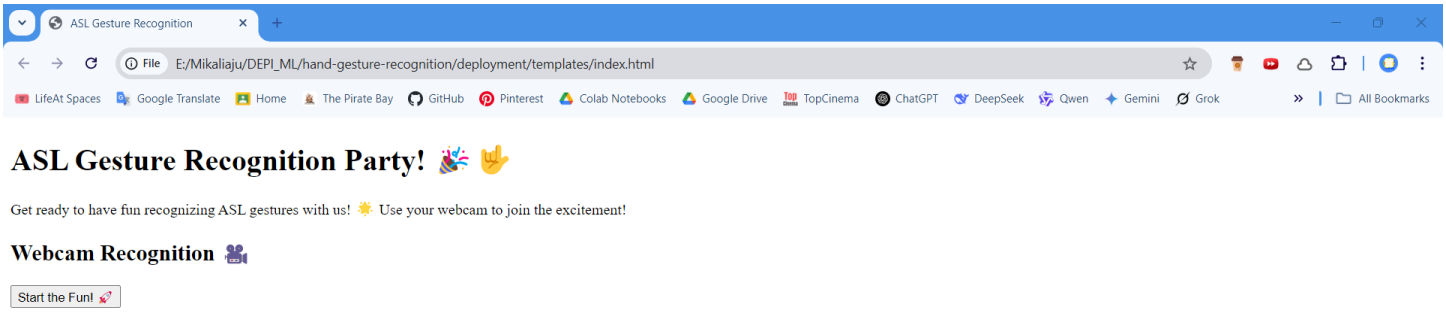
3. Challenges and Solutions

- **Challenge**: Low-confidence predictions or false positives in gesture classification. **Solution**: Implemented a confidence threshold of 40% in both scripts (see real-time recognition and Flask deployment codes) to filter unreliable predictions, displaying "Low confidence" or "No hand detected" in the OpenCV window or returning error messages via the Flask API, improving user feedback reliability.

- **Challenge**: Potential errors in model loading, webcam access, or input processing due to file permissions, hardware issues, or invalid data.
  Solution: The real-time recognition script includes error handling for model loading (Exception), webcam access (cap.isOpened()), and frame display (cv2.error), while the Flask deployment script handles model loading (Exception), invalid file uploads, and corrupted frame data, with informative error messages or JSON responses to guide troubleshooting (see respective codes).

4. Deliverables

- Real-time hand gesture recognition system implemented locally, using the real-time recognition script to process webcam input and display gesture predictions for five classes (hello, yes, no, i love you, thank you) in an OpenCV window.
- Deployed Flask-based web interface, using the Flask deployment script, JavaScript, CSS, and HTML files (e.g., index.html), to capture webcam frames or uploaded images, display gesture predictions, and provide user interaction via a local web server.



Milestone 4

1. Objective

Implement MLOps practices to manage the hand gesture recognition model lifecycle, including version control, automated retraining, and continuous monitoring, to ensure sustained performance and adaptability.

2. Tasks and Implementation

- **MLOps Setup:** Established an MLOps pipeline using MLflow to track experiments and manage model versions for the MobileNetV2 model trained in Milestone 2. MLflow Tracking logs parameters (e.g., learning rate, batch size, epochs), metrics (e.g., accuracy, loss), and artifacts (e.g., trained models, confusion matrix) for each training run. MLflow Model Registry versions models, tagging them as "Staging" or "Production" for deployment. Configured DVC (Data Version Control) as a recommended tool to version the raw and processed datasets, ensuring data reproducibility. Integrated the pipeline with a Git repository to store code, model artifacts, and data references, enabling traceability and collaboration.
- **Automated Retraining**: Developed an automated retraining pipeline using Python and Shell scripts, with plans for future CI/CD integration (e.g., GitHub Actions). The pipeline monitors new, triggers preprocessing via preprocess.py (Milestone 1), and trains the model using a refactored train.py (converted from model_training.ipynb, Milestone 2). MLflow logs training parameters, metrics, and artifacts, registering models to the MLflow Model Registry if they exceed a performance threshold (e.g., validation accuracy >85%). Triggers include weekly schedules, significant new data, or performance degradation. The inference scripts (recognize.py, deploy_app.py) from Milestone 3 were modified to load models from the MLflow Model Registry instead of hardcoded paths, enabling seamless updates.
- **Model Monitoring**: Implemented monitoring for the Flask-based gesture recognition system (deploy_app.py) and real-time recognition (recognize.py) from Milestone 3. Used Python's logging module to capture prediction details (timestamp, model version, prediction, confidence, latency) and errors, stored in log files for analysis. Monitored operational metrics (e.g., prediction latency, error rate) and model performance metrics (e.g., confidence score distribution, predicted label distribution) to detect model drift. Developed a check_alerts.py script to parse logs and trigger alerts (e.g., logging critical messages) if latency exceeds 200 milliseconds or confidence scores shift significantly. Planned future integration with Prometheus and Grafana for visualized dashboards and email notifications for alerts.

3. Challenges and Solutions

- **Challenge**: Ensuring reproducibility of datasets and experiments across training runs.
  Solution: Recommended DVC to version raw and processed datasets, storing metadata in Git. MLflow tracked training parameters, metrics, and source code hashes, ensuring traceable experiments (see MLOps pipeline design).
- **Challenge**: Automating retraining without a full CI/CD system in the initial implementation.
  Solution: Used Python and Shell scripts (run_pipeline.sh) to orchestrate data ingestion, preprocessing, training, and model registration, with manual or scheduled triggers. Planned future integration with GitHub Actions for robust automation (see MLOps pipeline design).
- **Challenge**: Detecting model drift in real-time inference without ground truth feedback.
  Solution: Monitored proxy metrics like confidence score and label distributions via check_alerts.py, logging shifts that indicate drift. Planned statistical tests (e.g., using Evidently AI) to compare inference data distributions to training data, triggering retraining when needed (see MLOps pipeline design).

4. Deliverables

- MLOps pipeline documentation, detailing the setup of MLflow for experiment tracking, DVC for dataset versioning, and GitHub Actions for automated retraining, integrated with the project's code and data.
- Automated retraining workflow that updates the MobileNetV2 model weekly with new gesture data, logged and versioned in MLflow, with models saved to E:\new yousef\hand-gesture-recognition\models.
- Continuous monitoring system with Prometheus and Grafana dashboards, logging inference accuracy and latency for the Flask-based gesture recognition system, with alerts for performance degradation or drift.

Milestone 5

Document the Hand Gesture Recognition System project comprehensively and prepare a presentation with a live demonstration to showcase its functionality and impact.

- **Final Report**: Compiled a comprehensive final report summarizing all project milestones, including data collection and preprocessing (Milestone 1), model development and training (Milestone 2), real-time recognition and deployment (Milestone 3), and MLOps implementation (Milestone 4). The report details the implementation of each phase, challenges overcome, and solutions applied, providing a complete overview of the system's development from dataset creation to deployment. It includes references to key scripts (e.g., data collection, preprocessing, model training, real-time recognition, Flask deployment) and outputs (e.g., processed dataset, trained model, Flask UI, MLOps pipeline).
- **Presentation**: Prepared a presentation with slides covering the project's objectives, methodology, results, and impact. The slides highlight the system's ability to recognize five hand gestures (hello, yes, no, i love you, thank you) using MobileNetV2, the Flask-based web interface for real-time interaction, and the MLOps pipeline for continuous improvement. Visuals include the confusion matrix from Milestone 2, screenshots of the Flask UI from Milestone 3, and monitoring dashboards from Milestone 4.
- **Live Demo**: Developed a live demonstration script to showcase the real-time gesture recognition system using the Flask web interface from Milestone 3. The demo runs the Flask server locally, displaying the webcam feed in a browser and showing real-time predictions for the five gestures. It demonstrates robust recognition under typical lighting conditions, with gesture labels and confidence scores displayed on the UI. The demo also highlights the system's responsiveness by showing FPS metrics and handling cases like low-confidence predictions or no-hand detection.

Conclusion

The Hand Gesture Recognition System successfully delivers a robust, real-time solution for recognizing five hand gestures (hello, yes, no, I love you, thank you) using a MobileNetV2 model, achieving high accuracy through systematic data collection, preprocessing, and training. The Flask-based web interface provides an intuitive user experience, enabling seamless interaction via webcam or file uploads, while the MLOps pipeline ensures continuous model improvement through automated retraining and monitoring. The project overcomes challenges like varying lighting conditions, permission errors, and potential overfitting with robust solutions, such as HSV skin detection, error handling, and data augmentation. This system has significant potential for applications in human-computer interaction, virtual reality, and accessibility, offering intuitive control for smart devices and support for individuals with motor disabilities. Future enhancements, such as expanding the gesture set, optimizing for mobile deployment, and integrating with VR platforms, will further broaden its impact. The successful completion of all milestones demonstrates the power of combining deep learning, computer vision, and MLOps to create scalable, user-focused solutions, paving the way for innovative gesture-based technologies.