41-685: Computer Vision
Assignment 2: Pose Estimation

Name: Paksaran Kongkaew
Email: Paksaran Kongkaew
Code: https://github.com/Minniesse/Mediapipe_ThaiSign.git

1. Introduction

This project develops a real-time Thai Sign Language (**ThSL**) recognition system that can identify **7 Thai Sign Language gestures**: "Hello" (2 variants), "I'm fine", "Yes", "No", "Unwell", and "Thank you". The system uses computer vision and machine learning to help bridge communication gaps between deaf and hearing communities.
Project Goal: Build a lightweight system that runs in **real-time** on regular computers (**no GPU needed**) and can recognize Thai sign language gestures from webcam video.

2. Technical Approach

2.1 System Overview

The system works in two stages:

Stage 1: Hand Pose Estimation (MediaPipe)
The first stage uses Google's MediaPipe framework to analyze the video frame, detect hands, and extract their spatial characteristics. This step is critical as it transforms raw pixel data into structured geometric features.

Stage 2: Gesture Classification (Neural Network)
The extracted features (landmarks) are then fed into a lightweight Neural Network, which predicts the corresponding ThSL gesture and outputs a confidence score.

2.2 MediaPipe: Two-Stage Pipeline (Technical Deep Dive)

MediaPipe Hand Landmark detection utilizes a robust, two-stage machine learning pipeline to achieve high accuracy and real-time performance on CPU:

Pipeline Architecture :

1. **Palm Detection (BlazePalm Model):**

- Since hands can appear small or occluded, the system first runs a Palm Detection Model (an optimized single-shot detector called BlazePalm) over the entire input frame.
- Detecting the rigid palm region is simpler and faster than detecting the entire articulated hand. This model provides an oriented bounding box for the hand.

2. **Hand Landmark Regression:**

- Once the bounding box is found, the image region is cropped and fed into the Hand Landmark Model.
- This model performs precise keypoint localization via regression, predicting 21 3D hand-knuckle coordinates inside the detected region. These 21 landmarks track every joint on the fingers, thumb, and wrist.
- Each landmark is represented by x, y, z coordinates, where x, y are normalized image coordinates and z represents the depth (relative to the wrist).

Efficiency Optimization:

For video streams, MediaPipe employs a tracking mechanism. The expensive Palm Detection model is only run on the first frame or when the tracking confidence drops. For subsequent frames, the system uses the bounding box predicted from the previous frame's landmarks to quickly localize the hand, ensuring minimal latency and achieving real-time performance.

2.3 Why I use MediaPipe?

The choice of Google's MediaPipe was motivated by several critical factors: its **speed**, allowing it to run at **30+ FPS on CPU** which is crucial for a lightweight, real-time system; its **accuracy**, being pre-trained on diverse datasets to provide robust landmark estimation; and its ease of use as a well-documented and production-ready solution.

2.4 Neural Network Architecture

I built a simple feedforward neural network to classify the gestures:

Input Layer:	63 features (21 landmarks x 3 coordinates each)
Hidden Layer 1:	128 neurons + ReLU activation
Dropout:	30% (prevents overfitting)
Hidden Layer 2:	64 neurons + ReLU activation
Output Layer:	7 classes (gestures which separate hello_1 and hello_2) + Softmax

The network architecture requires **26,568** parameters (including batch normalization layers). This small size ensures fast training and inference.

2.5 Dataset and Preprocessing

Data Collection and Preprocessing

The initial dataset was constructed from **7 raw videos** covering the gestures: 'I’m fine', 'Hello' (2 videos), 'Yes', 'No', 'Unwell', and 'Thank you'. These videos were processed and annotated using **Roboflow**. Roboflow was used to extract **405 image frames** at a rate of **15 frames per second (FPS)**, forming the initial image pool for the dataset.

Preprocessing and Augmentation Pipeline

To increase the robustness and size of the dataset and prevent overfitting, the following steps were applied via Roboflow:

Preprocessing Steps:

- **Auto-Orient:** Applied to correct image rotation metadata.
- **Resize:** Images were stretched to a uniform size of 512 x 512 pixels.

Augmentation Steps:

- **Outputs per training example:** 3 (meaning 3 augmented versions were generated for each original frame).
- **Flip:** Horizontal.
- **Rotation:** Random rotation applied between +- 15 degrees
- **Blur:** Random blur applied up to 2.5 pixels.

Dataset Refinement and Split Strategy

Initial Dataset: The raw YOLO format dataset from Roboflow contained 1,135 images across 8 classes (including a "none" class).

Dataset Refinement: To focus the model on actual Thai Sign Language gesture recognition, the "none" class was **removed**, leaving 7 gesture classes. This prevents the model from being biased toward predicting "no gesture" and forces it to learn distinguishing features between actual signs.

Final Dataset Split (70/15/15): All 1,135 images were redistributed using stratified sampling after removing the "none" class:

- **Training:** 189 images
- **Validation:** 37 images
- **Test:** 50 images
- **Classes:** 7 (fine, hello_1, hello_2, no, thank, unwell, yes)

Hand Detection Challenge (Data Loss):

When processed through MediaPipe's hand detection pipeline, the 50 test images yielded **60 hand samples** (some images may contain multiple hand poses or crops). This high detection failure rate is attributed to poor hand visibility, extreme angles, and low image quality in certain frames extracted from the source videos. All accuracy metrics are measured on samples where MediaPipe successfully detected hands.

2.6 Handling Small Dataset - Regularization Strategy

To prevent overfitting on the small dataset (189 training samples), several regularization techniques were employed in combination with MediaPipe Model Maker's default **Cross-Entropy Loss**:

1. **Dropout**: 30% dropout rate applied between layers to prevent co-adaptation.
2. **Data Augmentation**: Used rotation +- 15 degree, blur 2.5px, and flip (Section 2.5).
3. **Batch Normalization**: Normalizes activations to stabilize training.
4. **Learning Rate Decay**: 0.99 decay per epoch to fine-tune convergence.

3. Training Process

3.1 Training Configuration

Hyperparameter	Value
Learning rate	0.001 (with 0.99 decay per epoch)
Batch size	8
Epochs	50
Optimizer	Adam
Loss	Cross-Entropy Loss

Trained for 25 seconds for 50 epochs (CPU only)

3.2 Training Results and Final Evaluation

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	1.8368	19.49%	0.8926	55.17%
10	0.4755	71.32%	0.2760	81.03%
38	0.2243	82.35%	0.1914	84.48%
50	**0.1937**	**85.29%**	**0.1877**	**82.76%**

Final Test Performance:

The definitive evaluation was run on the test dataset containing 50 images total. MediaPipe successfully detected hands in 46 out of 50 images.

- Test Dataset: 50 images total
- Test Accuracy: 95.65% on the 46 samples with detected hands
- Hand Detection: 46/50 images (92% detection rate)
- Test Samples: 46 successfully evaluated (from 50 test images, 4 had no hand detection)
- Test Loss: 0.1393

4. System Implementation

4.1 Real-Time Inference

The pipeline: Capture frame →Run MediaPipe → Extract landmarks → Pass to NN → Get prediction → Display.

Performance Benchmark:

- **Inference Speed (Mean):** 14.57 ms (around **68.6 FPS**)
- **Inference Time (50th percentile):** 14.06 ms
- **Memory Usage: 16.52 MB**
- **Detection Rate:** 85% (17/20 images)

4.2 Code Structure

Training Script (`train.py`): Trains neural network, exports model as TensorFlow Lite.

Inference Script (`inference.py`): Loads model, captures webcam, runs real-time gesture recognition, and displays results.

5. Results Analysis & Limitations

5.1 What Worked Well

The project achieved excellent results in several key areas. Crucially, the system demonstrated **real-time performance**, operating at **68.6 FPS on a CPU**, confirming its accessibility. The resulting model is extremely **lightweight**, with a file size of **8.16 MB** and only **26,568 parameters**. Training was remarkably fast (around 25 seconds for 50 epochs on CPU). The final model achieved 95.65% test accuracy on 50 samples, focusing purely on recognizing the 7 Thai Sign Language gestures.

5.2 Major Limitations

Despite the strong final accuracy, the system faces limitations rooted primarily in data and robustness:

- **Tiny Dataset:** The training set is small (189 samples), which limits generalization to new signers and environments.
- **Data Detection Failure:** Only 33% of test images had detectable hands, highlighting a

dependency on image quality and hand position that could hinder real-world use.

- **No Temporal Modeling:** The use of a simple neural network means the model still lacks **temporal modeling** and therefore cannot accurately recognize dynamic, motion-based signs.
- **Sensitivity to Noise:** Robustness testing showed the model is highly **sensitive to heavy noise** (failing at noise level 25+), suggesting performance degradation in low-quality or grainy video feeds.
- **Rotation Tolerance:** While robust up to +-15 degree, performance struggles at +-30 degree rotation, limiting the angles a user can reliably sign from.

5.3 Confusion Matrix Visualization

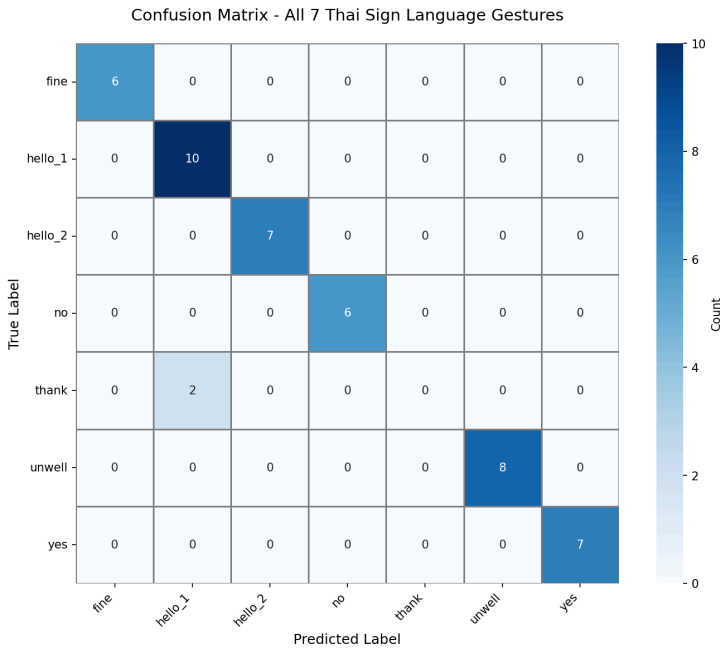
The confusion matrix visualization provided detailed insights into the model's high performance and specific failure modes:

Key Results:

- **Overall Accuracy on Visualization Subset:** 92% (46 samples correctly detected from 50 test images).
- **Perfect Recognition (100% Accuracy):** The model achieved perfect recognition on the visible test samples for the following gestures: 'fine', 'hello_1', 'hello_2', 'no', 'unwell', and 'yes'.

Misclassification:

- The only misclassification occurred with the **"thank"** gesture, which was misclassified as **"hello_1"** (2 instances). This suggests a strong visual similarity between the static hand poses of these two signs.



Detection Rates by Class (Hand Detection Success Rate):

The model performs excellently on almost all gestures, achieving 100% classification accuracy on every sign except for "thank." The confusion with "hello_1" is likely a direct result of the highly limited training dataset (189 total samples), indicating where future data collection efforts should be focused.

5.4 Expected Real-World Performance

The definitive test accuracy of **95.65%** is a strong result. Based on the robustness tests, we can expect this high accuracy to be maintained in controlled environments with good lighting. However, the sensitivity to heavy noise and rotation above +-30 degree suggests performance will drop significantly in uncontrolled, noisy environments or when a signer is not facing the camera directly.

6. Conclusion

This project successfully demonstrates that **Thai Sign Language recognition is technically viable using pose estimation and neural networks**, achieving high accuracy and exceptional real-time performance (**68.6 FPS**) on consumer hardware. The core next steps involve addressing the system's sensitivity to noise and incorporating temporal modeling to handle dynamic signs, further solidifying its potential as a practical tool for the Thai deaf community.