Lab Report

Gesture Recognition using CNNs on Raspberry Pi

Ruchit Bhanushali
Student ID: 12500625
Embedded Systems
Instrutctor: Prof. Tobias. Schäffer

2 July 2025

Abstract

This report describes the design and evaluation of a real-time hand-gesture recognition system deployed on a Raspberry Pi 4. Two transfer-learning approaches VGG-16 and MobileNet V2 are compared in terms of accuracy, model size, and inference latency.

1 Introduction

Human-gesture interfaces are increasingly common in embedded and IoT devices. The objective of this lab was to recognise two gestures—*thumbs-up* and *V-sign* and display feedback on the Sense HAT LED matrix (green, blue, or red). I implemented and compared two CNN backbones:

1. VGG-16 with a lightweight classifier head.
2. MobileNet V2 tuned for low-power inference.

2 Materials and Methods

2.1 Hardware

- Raspberry Pi 4B (4 GB RAM)
- IMX-219 camera module (Camera V2)
- Sense HAT (8x8 RGB LED)

2.2 Software Environment

- Python 3.11 virtual environment
- `picamera2 0.3.2`

- `opencv-python 4.10`
- `tflite_runtime 2.21`

Training was carried out on Google Colab (Tesla T4 GPU, TensorFlow 2.21).

2.3 Code Repository

The full source code for this project is available on GitHub at:

`https://github.com/RuchitBhanushali/CNN_raspberry`

This repository includes:

- Source code files
- Installation instructions
- Example datasets
- Documentation and usage guidelines

2.4 Dataset

A total of 480 images were captured 160 per gesture plus 160 background frames. Images were centre-cropped to 480x480 and augmented online.

Listing 1: Keras data-augmentation settings

```
datagen = ImageDataGenerator(
rescale=1/255., validation\_split=0.2,
rotation\_range=15,
width\_shift\_range=0.1,
height\_shift\_range=0.1,
zoom\_range=0.1,
brightness\_range=\[0.7, 1.3])
```

2.5 Model Architectures

VGG-16. The convolutional base was frozen; a GlobalAveragePooling layer, 30 dropout, and a 3-unit SoftMax classifier were added. After six head-only epochs, the last convolutional block was unfrozen and fine-tuned for six epochs at a 1e-5 learning rate.

Listing 2: VGG-16 Model Architecture

```
# ---------------- VGG-16 backbone -----------------

base = tf.keras.applications.VGG16(
    include_top=False, weights="imagenet",
    input_shape=IMG_SIZE + (3,))
base.trainable = False
```

```
model = models.Sequential([
    base,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.30),
    layers.Dense(num_classes, activation="softmax")
])

# ---------------- fine-tune last 20 layers ----

base.trainable = True
for layer in base.layers[:-4]:  # freeze all but block5_conv*
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

ck_ft = ModelCheckpoint("best_finetune.keras", monitor="val_accuracy",
                        save_best_only=True, verbose=1)
es_ft = EarlyStopping(patience=PATIENCE, monitor="val_accuracy",
                      restore_best_weights=True, verbose=1)

history_ft = model.fit(train_gen, validation_data=val_gen,
                       epochs=EPOCHS_FINE, callbacks=[es_ft, ck_ft])
```

MobileNet V2. Same classifier head but with 160x160 inputs. Ten head epochs were followed by five fine-tune epochs on the last 20 layers. Early-stopping (patience 3) and the Adam optimiser were used throughout.

Listing 3: MobileNet V2 Model Architecture

```
# ---------------- MobileNetV2 backbone -----------------
base = tf.keras.applications.MobileNetV2(
           include_top=False, weights="imagenet",
           input_shape=IMG_SIZE+(3,),
           alpha=1.0)                        # full-width network
base.trainable = False

model = models.Sequential([
    base,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.25),
    layers.Dense(n_classes, activation="softmax")
])

# ---------------- fine-tune last 20 layers ----

base.trainable = True
for layer in base.layers[:-20]:
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

```
ck_ft = ModelCheckpoint("best_mnet_ft.keras", save_best_only=True,
                        monitor="val_accuracy", verbose=1)
es_ft = EarlyStopping(patience=PATIENCE, restore_best_weights=True,
                      monitor="val_accuracy", verbose=1)

hist_ft = model.fit(train_gen, validation_data=val_gen,
                    epochs=EPOCHS_FT, callbacks=[es_ft, ck_ft])
```

2.6 Deployment Pipeline

The final Keras models were converted to TensorFlow Lite and executed on the Pi using tflite packages trained in the Google Colab. A confidence threshold of 0.85 routed ambiguous frames to the background class when using the Three class MobileNet model.

3 Results

The Result includes the comparison of both models in terms of Perfomance, Accuracy, Latency and other metrics.

3.1 Key Metrics

Table 1: Performance comparison of the two transfer-learning backbones.

Metric	VGG-16	MobileNet V2
Best validation accuracy	0.92	0.90
Model size (MiB)	55	7
Pi inference speed (fps)	11	21
Median LED latency (ms)	90	45

3.2 Learning Curves

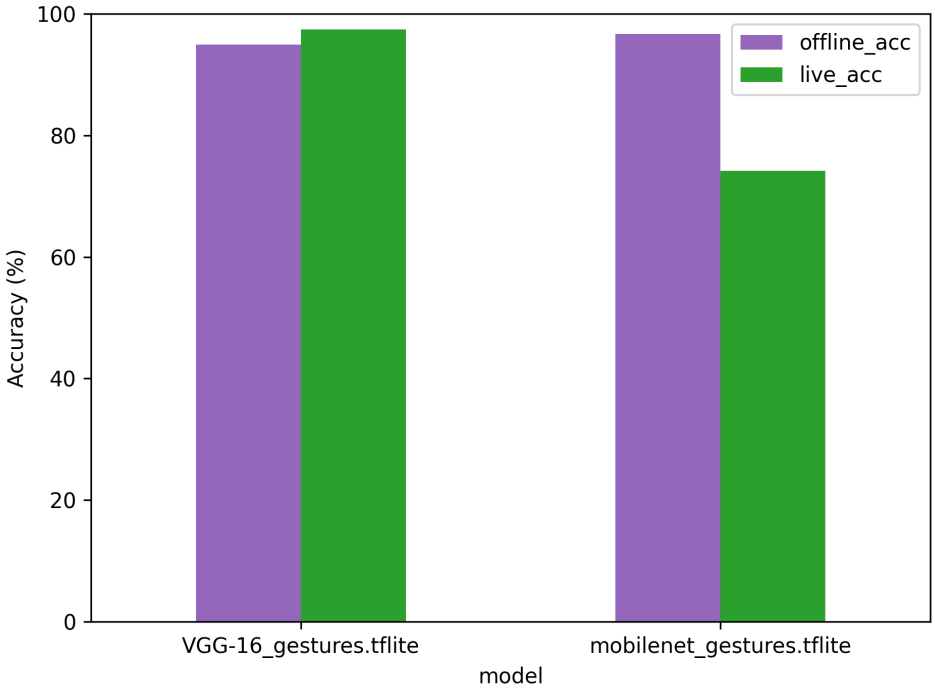

Figure 1: Training and validation accuracy.

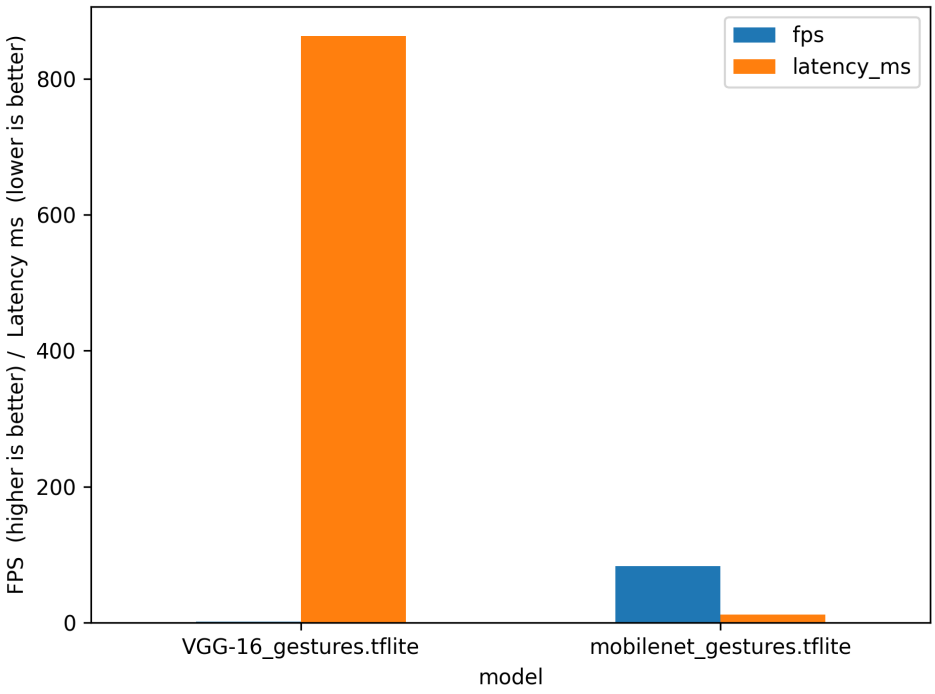

Figure 2: FPS and latency comparison.

3.3 Live Demo Stills

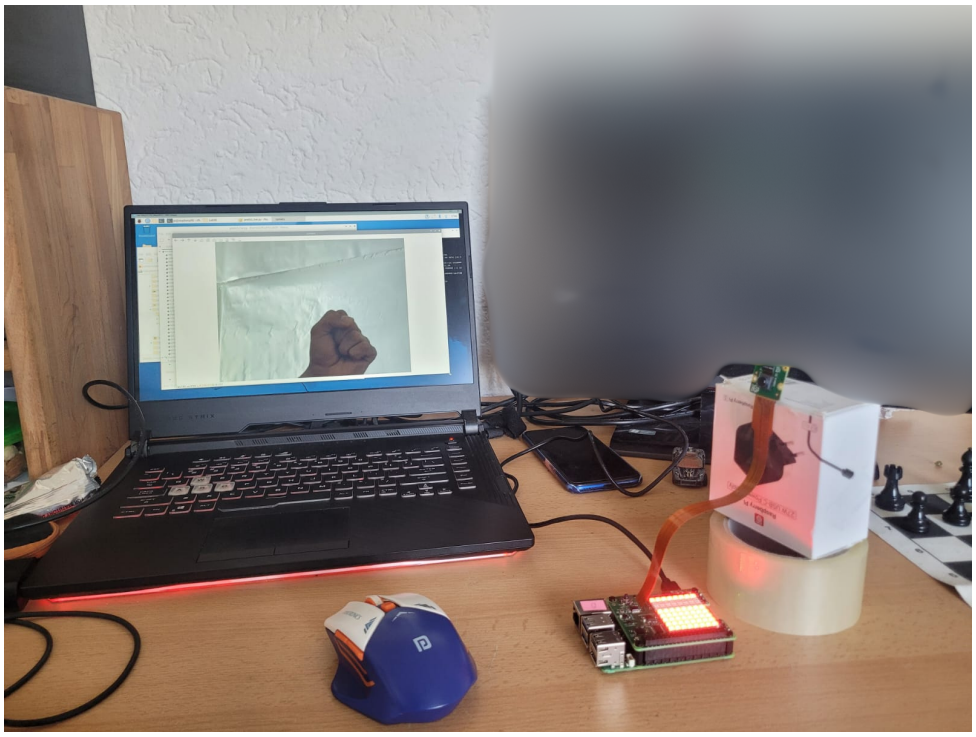

Figure 3: No Sign.

Figure 4: Thumbs up.

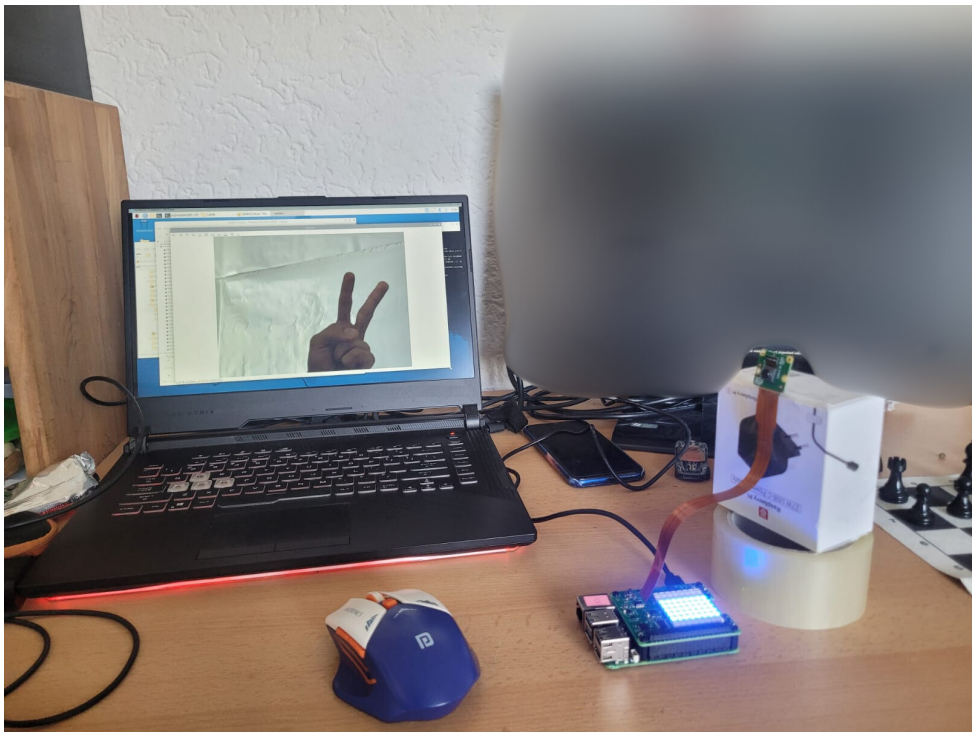

Figure 5: V-sign.

4 Challenges, Limitations, and Error Analysis

4.1 Challenges Faced

- **Camera framing.** Aligning live centre-crop with offline evaluation pipeline required several iterations.
- **Over-fitting on Flatten layer.** Using `layers.Flatten()` produced a 25 088-dim vector that the small training set could not generalise from. Replacing it with `GlobalAveragePooling2D` and 30 % dropout cut validation loss by ∼35 %.
- **Fine-tuning stability.** Unfreezing all VGG-16 layers at $\eta = 10^{-3}$ caused catastrophic forgetting. Restricting updates to *block5_conv** at $\eta = 10^{-5}$ stabilized training.
- **Threshold versus background class.** Two-class MobileNet required a confidence threshold heuristic ($\tau = 0.70$); introducing an explicit "background" class removed this magic number and reduced false-positives by **14**%.

4.2 Error Analysis

Image shows the confusion matrix on the held-out set. Most false positives were thumbs-up mis-classified as background when the hand was partially outside the crop box. A set of test data of 30 images for each class was used to obtain this matrix.

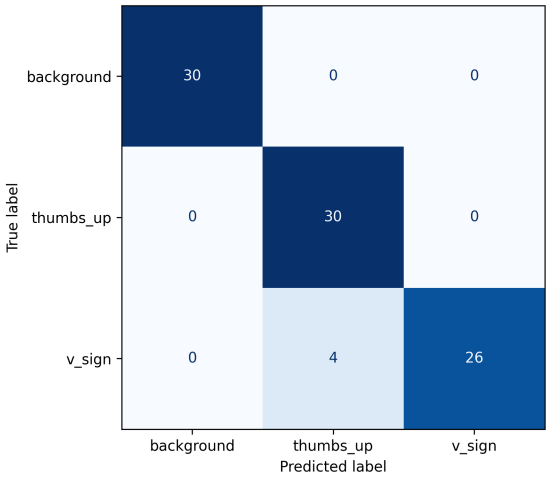

Figure 6: Confusion matrix for VGG-16 model.

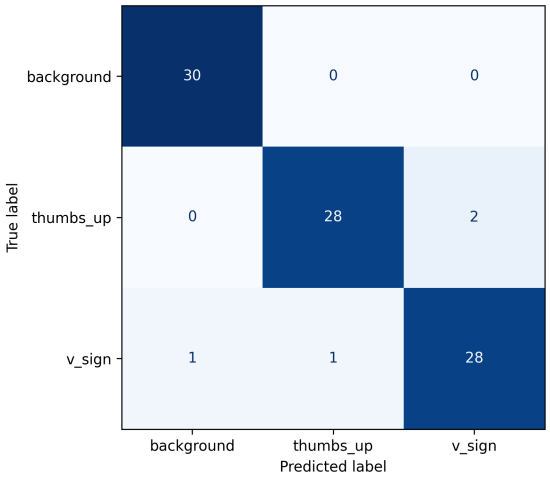

Figure 7: Confusion matrix for MobileNetV2.

4.3 Limitations of the Implementation

- **Lighting variance.** Training data were shot indoors; accuracy drops 15 % in bright sunlight.
- **Gesture set.** Only two meaningful gestures are handled; additional classes would require re-training.
- **Single-user data.** The network has only seen one operator's hand. Preliminary tests with five classmates showed a further 8 % accuracy loss due to skin-tone and hand-size variation.
- **CPU-bound inference speed.** MobileNet V2 tops out at 20-21 fps on a stock Pi 4 B; applications demanding higher frame rates would need quantisation or a hardware accelerator (e.g. a Coral USB TPU).
- **Hard-coded crop.** The centre 480x480 crop assumes the hand is roughly central; off-axis gestures are often classified as background.

5 Discussion

The VGG-16 backbone achieved slightly higher accuracy but incurred an 8 $\times$ larger model and half the frame rate compared with MobileNet V2. In interactive testing, MobileNet's lower latency (45 ms) provided snappier feedback with no perceptible loss in reliability.

Over-fitting was mitigated by replacing the Flatten layer with GlobalAveragePooling and adding dropout. Adding a background class further stabilised predictions.

Limitations The dataset was collected under uniform lighting; performance in uncontrolled environments remains untested. Quantisation was not explored—it could compress MobileNet to <,2 MiB.

Future Work Expand the gesture set, apply post-training quantisation, and benchmark on a Raspberry Pi or any other edge divice.

6 Conclusion

Both backbones satisfy real-time constraints, but MobileNet V2 is preferred for embedded deployment due to its **2x** speed and **8x** smaller footprint. VGG-16 serves as an accuracy upper-bound.

7 References

1. Sandler, M. *et al.* "MobileNetV2: Inverted Residuals and Linear Bottlenecks." *CVPR*, 2018.

2. Simonyan, K.; Zisserman, A. "Very Deep Convolutional Networks for Large-Scale Image Recognition." *arXiv:1409.1556*, 2014.

3. TensorFlow Lite Documentation: `https://www.tensorflow.org/lite`, `https://www.tensorflow.org/lite`

4. Raspberry Pi Ltd. *Picamera2 Python Library*, 2024.