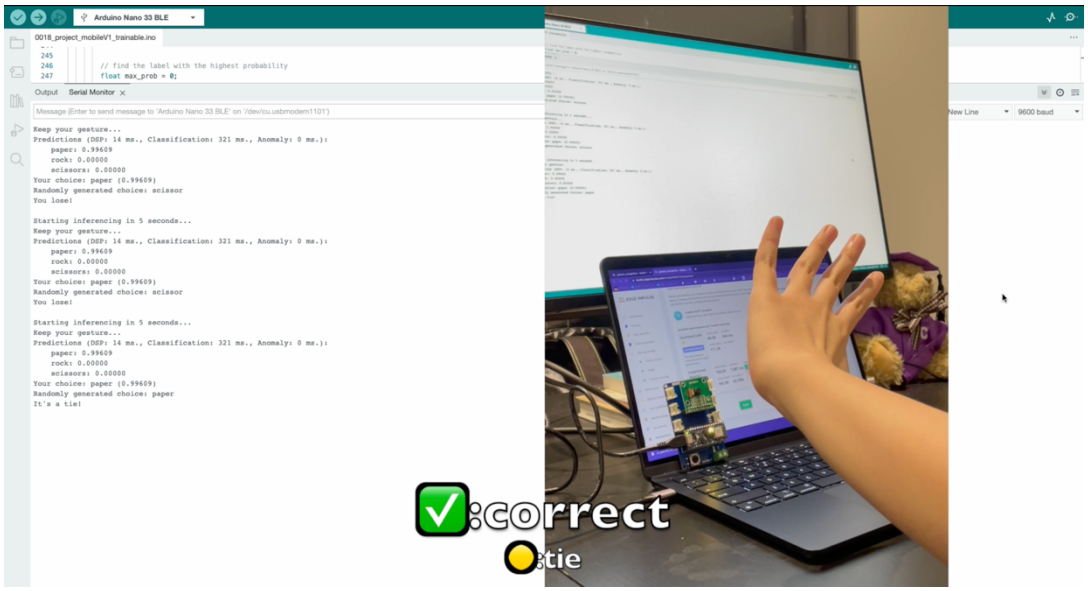
Playing Rock, Paper, and Scissors with Computer



CEGE 0018 Coursework Project

Author: Yiqi Huang

GitHub Link: https://github.com/sophuang/CASA0018CourseworkProject

Edge Impulse Link: https://studio.edgeimpulse.com/public/196617/latest

1. Introduction

1.1 Background of Gesture Recognition

Gesture recognition is a significant field of computer vision technique since it is a way of human communicating with computers and machines. Most important and vital applications of gesture recognition nowadays include giving commands or talking to computers without using mouse to click, interacting with medical instruments during medical operation and surgery, controlling vehicles functions during driving, controlling smart home devices and etc. All these applications allow the use to control the device or system without touching a button or screen physically, which provide the convenience, safety and accuracy when the user need a multitasking process.

Another popular appliable aspect of gesture recognition is the controlling in gesture-based games. Computer games are a particularly technologically promising and commercially rewarding arena for innovative interfaces due to the entertaining nature of the interaction. (Admin, 2022) This provides more interactive, immersive, and fast-response gaming experiences and interfaces for the user. And this is also the field this study will be mainly focused on.

1.2 Research Question

The inspiration of this project is a Switch game called 'How to win paper rock and scissors in Dr. Kawashima's Brain Training'. In this game, the infrared (IR) sensors are used to detect user's gesture, which have advantages in clearly detecting how many fingers the user is holding up and whether the user is holding a fist or not. (Jones, n.d.)

To have a better understanding about computer vision, object detection and classification lectures taught in this module, this study aims to create a simple gaming program in Arduino IDE that is deployed on the microcontroller, Arduino Nano BLE33 to complete the following tasks:

- Recognize which of the gestures, Rock, Paper and Scissors the user gives, using the built-in camera on Arduino Nano BLE33.
- Output a randomly generated gesture choice by the computer program.
- Compare the recognized and generated choices to determine whether the user wins or loses. Also use the built-in LED light as an output signal of winning and losing status.
- The program can recognize the user's gesture up to 80%

1.3 Methodology and Application Overview

This study primarily utilizes Edge Impulse, a comprehensive platform that seamlessly facilitates data collection, labeling, train-test set splitting, model building, testing, and program deployment in a consistent and streamlined manner.

In more details, the transfer learning technique will be employed in the model building progress. This is a deep learning approach that enables the use of pre-trained models as a foundation for training customized models with a limited dataset to accomplish new objectives. This method not only facilitates a more efficient training process but also often results in improved performance. (Brownlee, 2019)

Multiple pre-trained base models are available for object detection, with popular and powerful options such as Faster R-CNN, YOLO, and MobileNet. However, when deploying a program on a microcontroller, it is essential to consider factors such as model size and computational complexity to ensure optimal performance on resource-constrained devices. (Admin, 2023) (Shinya, et al., 2019)

MobileNet is a family of lightweight neural network architectures designed specifically for mobile and embedded vision applications. It uses depthwise separable convolutions, which greatly reduce the number of parameters and computations compared to traditional convolutional layers. This makes MobileNet a suitable choice for running on resource-constrained devices while still delivering reasonable accuracy. (Howard, 1027)

2. Data

2.1 Data Collection

In this study, Edge Impulse was utilized to collect, label, and split the dataset, with images captured using the OV7675 camera module provided in the Arduino Nano 33 BLE KIT. The camera module offers a resolution of 640 x 480 VGA. The final dataset consists of 525 images, divided into 450 training and 75 testing samples, following an 85%:15% split. The dataset comprises four labels:

- Rock: 165 training / 25 testing
- Paper: 120 training / 25 testing
- Scissors: 165 training / 25 testing

Initially, images of gestures such as thumbs-up or the "OK" sign were collected under a separate label called "Other." However, after several experimental observations, it was determined that including these additional gestures adversely affected the model's performance in classifying rock, paper, and scissors, resulting in decreased accuracy. Given that the project's primary objective is to enable users to play rock, paper, and scissors with a computer, users are not expected to use other gestures during gameplay. As a result, all data in the "Other" label was excluded from the analysis.

As depicted in the figures above, the dataset sizes for the different labels vary. The experimental results in a later section of this study demonstrate that the model has lower

accuracy in distinguishing between "Scissors" and "Rock" gestures. Consequently, the dataset sizes for these two classes are slightly larger than that of the "Paper" class. This adjustment allows the model to better learn the specific features of the "Scissors" and "Rock" classes, ultimately improving its overall performance.

2.2 Dataset Description

- **Control Factor**

During the data collection process, three primary factors were considered to ensure the robustness of the dataset. The first one is the background variability. To mitigate the impact of varying backgrounds on the sampled images, data was collected across different locations, featuring diverse backgrounds, and at different times of day, encompassing a range of daylight conditions. This approach helps the model generalize better across varied environments.

The second one is hand orientation and position. To account for the differences between left and right hands, as well as the front (palm) and back of hands, the dataset was carefully curated to include samples covering these variations. This consideration ensures the model can accurately detect the target gestures, irrespective of hand orientation and position.

The last one is gesture variations across individuals. Different people may exhibit distinct gesture habits when forming rock, paper, or scissors signs. To address this issue, images were collected with the assistance of friends, thereby introducing a variety of individual styles into the dataset. This diverse data helps the model become more adept at recognizing the gestures, regardless of individual variations. Here are the examples of images input, showing the data in different background, daylight conditions and by different people.

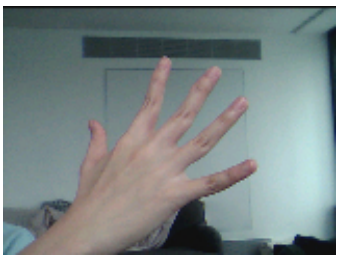


Fig.1 Sample of “paper” facing in back

Fig.2 Sample of “paper” facing in front by my friend

Fig.3 Sample of “scissors” facing in front

Fig.4 Sample of “scissors” facing in back

Fig.5 Sample of “rock” facing in front

Fig.6 Sample of “rock” by my friend

2.3 Features Generation

The feature generation process began by setting the image size to 96x96 pixels, using the "squash" resize mode and “Transfer Learning” as the learning box.

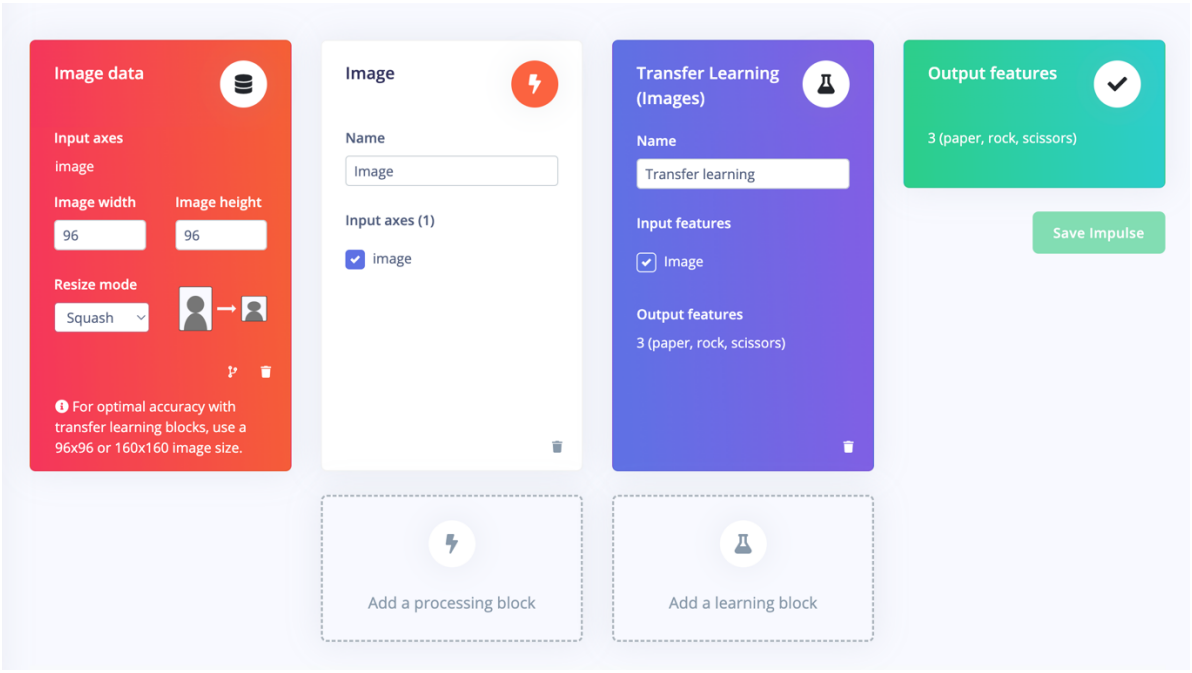


Fig.7 Impulse Design

The "squash" resize mode was chosen to ensure the model retains as much information as possible from the original images. Additionally, the color depth was set to RGB to provide the model with a richer representation of the input data. Upon completion, the generated features were visualized as shown below.

As depicted in the Feature Explorer, the features contained within each label are not easily distinguishable or separable. The sample points are clustered into three prominent groups, with each group containing more than one label. Notably, the groups at the bottom left and bottom right primarily consist of rock and scissors labels. This observation indicates that the rock and scissors labels share similar features, which also aligns with the experimental results presented

later in this study. Therefore, we can infer that differentiating between rock and scissor poses a significant challenge for this project.

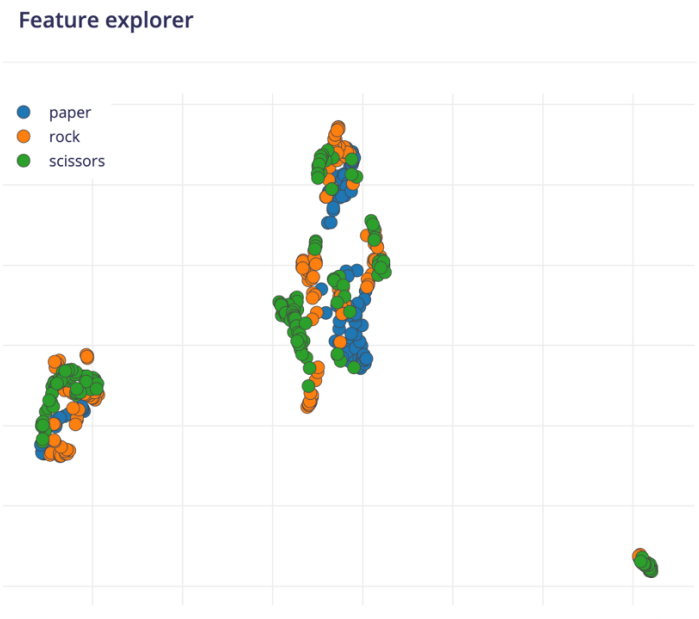


Fig.8 Feature explorer

3. Model Building and Training

3.1 Initialization

In this section, we focus on the process of building and training the machine learning model using the prepared dataset and feature extraction techniques. To initialize the first model, the convolutional neural network MobileNetV2 is employed, with 20 epochs, a 0.005 learning rate, and a 20% validation set. To minimize the overfitting issue, an auto-balanced dataset and data augmentation techniques are also implemented.

The trained model achieves a 91.8% accuracy in the validation set and 86.49% in the testing set, as illustrated in the figures below. This performance demonstrates that the model is relatively robust.

However, it is important to note that the PEAK RAM USE is 346.9KB. Even when using the EON compiler, the RAM usage still surpasses the maximum allowable size of 256KB on the Arduino Nano board. Consequently, the EON tuner is employed to experiment with various parameters in order to obtain an optimized model that fits within the 256KB RAM constraint.

Fig.9 Validation accuracy and Testing accuracy

3.2 Experiment

- **EON tuner**

Utilizing the EON tuner, various models with different parameters were experimented with, as illustrated in the following table:

	Learning type	Image input	color depth	learning rate	epochs	Accuracy (validation)	Accuracy (test)
1	Classification	32 x 32	grayscale	0.001	50	89%	55%
2	Classification	32 x 32	RGB	0.001	50	87%	45%
3	Transfer Learning	96 x 96	RGB	0.0005	50	84%	62%
4	Transfer Learning	96 x 96	grayscale	0.005	50	75%	62%
5	Transfer Learning	96 x 96	RGB	0.005	50	73%	76%

As indicated in the table, models employing classification learning type (non-pretrained models, standard convolutional neural networks with three 2D convolutional layers and one dropout

layer) exhibit strong performance on the validation set but show a significant decline in performance on the unseen testing set.

In pursuit of the best performance on unseen data, the transfer learning approach using MobileNetV1 (Model No. 5) was chosen. This model features a 96x96 image input size, RGB color depth, 0.005 learning rate, and 50 epochs. These results align with the assumptions and discussions presented in sections 1.3 and 2.3.

To elaborate further, a learning rate of 0.005 strikes an optimal balance between faster convergence and preventing the overshooting of optimal weights during the training process, as compared to learning rates of 0.001 and 0.0005. A higher learning rate might cause the model to overshoot the optimal weights, resulting in unstable training, while a lower learning rate could prolong convergence and potentially cause the model to become stuck in local minima. Incorporating RGB color depth enables the model to distinguish between different gestures more effectively. In contrast, grayscale images would limit the model's learning capacity to brightness variations alone, potentially impairing its ability to differentiate between the classes, thus validating the choices made in sections 1.3 and 2.3.

```python
import math
from pathlib import Path
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Dense, InputLayer, Dropout, Conv1D, Flatten, Reshape, MaxPooling1D, BatchNormalization,
    Conv2D, GlobalMaxPooling2D, Lambda, GlobalAveragePooling2D)
from tensorflow.keras.optimizers import Adam, Adadelta
from tensorflow.keras.losses import categorical_crossentropy

sys.path.append('./resources/libraries')
import ei_tensorflow.training

WEIGHTS_PATH = './transfer-learning-weights/edgeimpulse/MobileNetV1.0_1.96x96.color.bsize_96.lr_0_05.epoch_66.val_accuracy_0.14.hdf5'

# Download the model weights
root_url = 'https://cdn.edgeimpulse.com/'
p = Path(WEIGHTS_PATH)
if not p.exists():
    print(f"Pretrained weights {WEIGHTS_PATH} unavailable; downloading...")
    if not p.parent.exists():
        p.parent.mkdir(parents=True)
    weights_data = requests.get(root_url + WEIGHTS_PATH[2:]).content
    with open(WEIGHTS_PATH, 'wb') as f:
        f.write(weights_data)
    print(f"Pretrained weights {WEIGHTS_PATH} unavailable; downloading OK")
    print("")
```

```python
INPUT_SHAPE = (96, 96, 3)


base_model = tf.keras.applications.MobileNet(
    input_shape = INPUT_SHAPE,
    weights = WEIGHTS_PATH,
    alpha = 0.1
)

base_model.trainable = True

model = Sequential()
model.add(InputLayer(input_shape=INPUT_SHAPE, name='x_input'))
# Don't include the base model's top layers
last_layer_index = -5
model.add(Model(inputs=base_model.inputs, outputs=base_model.layers[last_layer_index].output))
model.add(Reshape((-1, model.layers[-1].output.shape[3])))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(classes, activation='softmax'))



BATCH_SIZE = 32
EPOCHS = args.epochs or 50
LEARNING_RATE = args.learning_rate or 0.005
train_dataset = train_dataset.batch(BATCH_SIZE, drop_remainder=False)
validation_dataset = validation_dataset.batch(BATCH_SIZE, drop_remainder=False)
callbacks.append(BatchLoggerCallback(BATCH_SIZE, train_sample_count, epochs=EPOCHS))
```

```python
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_dataset, validation_data=validation_dataset, epochs=EPOCHS, verbose=2,
    callbacks=callbacks)

print('')
print('Initial training done.', flush=True)

# How many epochs we will fine tune the model
FINE_TUNE_EPOCHS = 10
# What percentage of the base model's layers we will fine tune
FINE_TUNE_PERCENTAGE = 65

print('Fine-tuning best model for {} epochs...'.format(FINE_TUNE_EPOCHS), flush=True)

# Load best model from initial training
model = ei_tensorflow.training.load_best_model(BEST_MODEL_PATH)

# Determine which layer to begin fine tuning at
model_layer_count = len(model.layers)
fine_tune_from = math.ceil(model_layer_count * ((100 - FINE_TUNE_PERCENTAGE) / 100))

# Allow the entire base model to be trained
model.trainable = True
# Freeze all the layers before the 'fine_tune_from' layer
for layer in model.layers[:fine_tune_from]:
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.000045),
              loss='categorical_crossentropy',
```

```python
FINE_TUNE_EPOCHS = 10
# What percentage of the base model's layers we will fine tune
FINE_TUNE_PERCENTAGE = 65

print('Fine-tuning best model for {} epochs...'.format(FINE_TUNE_EPOCHS), flush=True)

# Load best model from initial training
model = ei_tensorflow.training.load_best_model(BEST_MODEL_PATH)

# Determine which layer to begin fine tuning at
model_layer_count = len(model.layers)
fine_tune_from = math.ceil(model_layer_count * ((100 - FINE_TUNE_PERCENTAGE) / 100))

# Allow the entire base model to be trained
model.trainable = True
# Freeze all the layers before the 'fine_tune_from' layer
for layer in model.layers[:fine_tune_from]:
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.000045),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_dataset,
              epochs=FINE_TUNE_EPOCHS,
              verbose=2,
              validation_data=validation_dataset,
              callbacks=callbacks,
              class_weight=None
          )
```

Fig.10 Keras mode of MobileNetV1 0.1

- **Keras mode**

To achieve better performance, I modified the base model MobileNetV1 to be trainable in Keras, setting the FINE_TUNE_EPOCHS to 10 and FINE_TUNE_PERCENTAGE to 65.

This means that during the fine-tuning stage of the pre-trained model, the model will iterate 10 times over the dataset to update its pre-trained weights with our gesture dataset. Only 65% of the base model's layers will be fine-tuned, while the remaining 35% will be kept frozen. This approach is designed to retain the valuable features learned by the base, pre-trained model during its initial training while allowing the model to adapt to the new task.

- **Training and testing results**

As a result, the model achieved 97.8% accuracy in the validation set and 90.67% accuracy in the testing set. This significant improvement in model performance can be attributed to allowing the base model to be trainable, enabling it to adapt its pre-trained weights to be more customized and better suited to our dataset.

Fig.11 Quntized validation set accuracy of trainable MobileNetV1 0.1.

Fig.12Unoptimized testing set accuracy of trainable MobileNetV1 0.1

Fig.13 Unoptimized testing set accuracy of trainable MobileNetV1 0.1.

3.3 Model Deployment

Next, the model is deployed in the Arduino IDE. Although the quantized version provided by the EON compiler has lower RAM usage, the unoptimized version offers higher accuracy. As a result, the unoptimized version is chosen for deployment. Afterward, the model is exported as a library, allowing it to be easily integrated into the Arduino IDE for use.

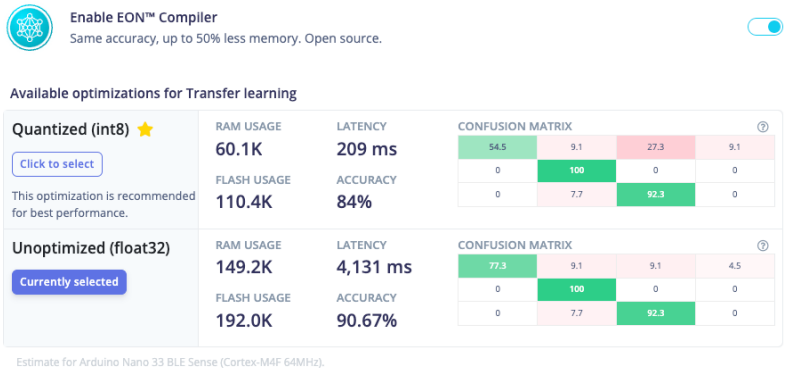


Fig.14 Anlysis of Quantized and Unoptimized versions of model by EON complier

5. Program Building

To incorporate the desired functionality, custom code has been added to the deployed script as follows:

- Initialize the built-in LED colors:

```
30
31  /* Initialize the LED color*/
32  #define RED 22
33  #define BLUE 24
34  #define GREEN 23
35
36  int option;
37
```

- Define a function to gerente a string out of a list of options randomly:

```
// Generate a random string from a list of options
const char* random_string(const char** options, size_t count) {
    return options[rand() % count];
}
```

- Change the delay time in the program to 5 seconds so that game rounds are not coming too quickly. Turn the image classification delay in the program as a preparing time for the user:

```
165  void loop()
166  {
167      bool stop_inferencing = false;
168
169      while(stop_inferencing == false) {
170          ei_printf("\nStarting inferencing in 5 seconds...\n");
171
172          // instead of wait_ms, we'll wait on the signal, this allows threads to cancel us...
173          if (ei_sleep(5000) != EI_IMPULSE_OK) {
174              break;
175          }
176
177          ei_printf("Keep your gesture...\n");
178
179          if (ei_camera_init() == false) {
180              ei_printf("ERR: Failed to initialize image sensor\r\n");
181              break;
182          }
183
```

- In the printing prediction stages, add the additional command to output the most confidence recognition as user's choice

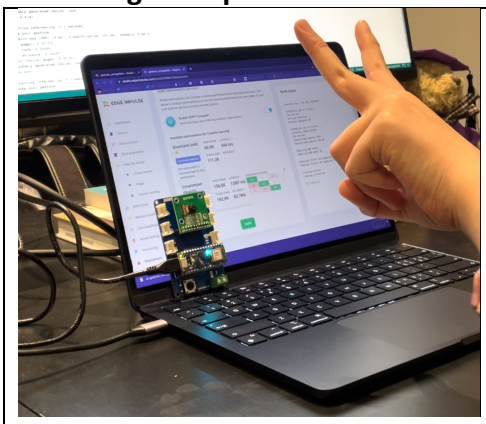
```
// find the label with the highest probability
float max_prob = 0;
const char* max_label = "";
for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
    if (result.classification[ix].value > max_prob) {
        max_prob = result.classification[ix].value;
        max_label = result.classification[ix].label;
    }
}

ei_printf("Your choice: %s (%.5f)\n", max_label, max_prob);
```

- Let the computer randomly generates choices out of {"rock", "paper", "scissors"} using the function defined before.
- Compare the generated choice with user's choice using strcmp() function. The strcmp() function is used to compare two strings character by character and determine their equality or the lexicographic order.
- Output the gaming round result and turn on the LED light as a signal
    - Blue indicates lose
    - Green indicates ties
    - Red indicates win

```
// generate a random choice and compare it with the max_label
const char* options[] = {"rock", "paper", "scissor"};
const char* random_choice = random_string(options, 3);
ei_printf("Randomly generated choice: %s\n", random_choice);
if (strcmp(max_label, random_choice) == 0) {
    ei_printf("It's a tie!\n");
    digitalWrite(GREEN, HIGH);
    delay(1000);
    digitalWrite(BLUE, LOW);
    digitalWrite(RED, LOW);
    digitalWrite(GREEN, LOW);

} else if (strcmp(max_label, "rock") == 0 && strcmp(random_choice, "scissor") == 0 ||
           strcmp(max_label, "paper") == 0 && strcmp(random_choice, "rock") == 0 ||
           strcmp(max_label, "scissor") == 0 && strcmp(random_choice, "paper") == 0) {
    ei_printf("You win!\n");
    digitalWrite(RED, HIGH);
    delay(1000);
    digitalWrite(GREEN, LOW);
    digitalWrite(RED, LOW);
    digitalWrite(BLUE, LOW);

} else {
    ei_printf("You lose!\n");
    digitalWrite(BLUE, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000);
    digitalWrite(RED, LOW);
    digitalWrite(GREEN, LOW);
    digitalWrite(BLUE, LOW);


}
```

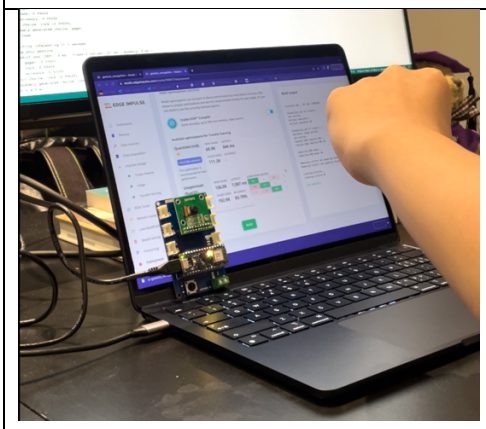
6. Program Testing

6.1 testing examples



```
Starting inferencing in 5 seconds...
Keep your gesture...
Predictions (DSP: 13 ms., Classification: 323 ms., Anomaly: 0 ms.):
    paper: 0.00391
    rock: 0.00391
    scissors: 0.99609
Your choice: scissors (0.99609)
Randomly generated choice: scissor
You lose!
```

- Experiment Environment: Night
- Prediction: Correct
- Facing direction: Palm
- Tester: Myself

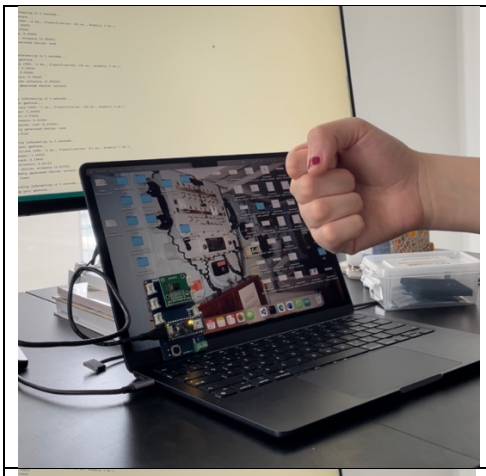


```
Starting inferencing in 5 seconds...
Keep your gesture...
Predictions (DSP: 13 ms., Classification: 322 ms., Anomaly: 0 ms.):
    paper: 0.00000
    rock: 0.99609
    scissors: 0.00000
Your choice: rock (0.99609)
Randomly generated choice: rock
It's a tie!
```

- Experiment Environment: Night
- Prediction: Correct
- Facing direction: Palm
- Tester: Myself

```
Starting inferencing in 5 seconds...
Keep your gesture...
Predictions (DSP: 13 ms., Classification: 322 ms., Anomaly: 0 ms.):
    paper: 0.98438
    rock: 0.00000
    scissors: 0.01562
Your choice: paper (0.98438)
Randomly generated choice: scissor
You lose!
```

- Experiment Environment: Night
- Prediction: Correct
- Facing direction: Palm
- Tester: Myself

	Starting inferencing in 5 seconds... Keep your gesture... Predictions (DSP: 13 ms., Classification: 324 ms., Anomaly: 0 ms.): paper: 0.00000 rock: 0.99609 scissors: 0.00000 Your choice: rock (0.99609) Randomly generated choice: rock It's a tie! • Experiment Environment: Day • Prediction: Correct • Facing direction: Back • Tester: My roommate
	Starting inferencing in 5 seconds... Keep your gesture... Predictions (DSP: 13 ms., Classification: 324 ms., Anomaly: 0 ms.): paper: 0.00000 rock: 0.01172 scissors: 0.98828 Your choice: scissors (0.98828) Randomly generated choice: scissor You lose! • Experiment Environment: Day • Prediction: Correct • Facing direction: Back • Tester: My roommate
	Starting inferencing in 5 seconds... Keep your gesture... Predictions (DSP: 13 ms., Classification: 324 ms., Anomaly: 0 ms.): paper: 0.00000 rock: 0.00000 scissors: 0.99609 Your choice: scissors (0.99609) Randomly generated choice: rock You lose! • Experiment Environment: Day • Prediction: Wrong; Recognized paper as scissors • Facing direction: Back • Tester: My roommate

7. Discussion and Conclusion

To conclude, transfer learning applied on MobileNetV1 performs robustly in this project. However, it still has limitations. The program testing results indicate a case where the paper gesture is mis-recognized as scissors. This might be attributed to the unbalanced dataset size. In

earlier experiments (conducted during the presentation process), the primary uncertainty was observed between the rock and scissors gestures, leading to an increased dataset size for rock and scissors. However, doing so inadvertently increased the uncertainty for the paper gesture, as demonstrated in Fig.14. For future work, utilizing a balanced dataset might be a more reasonable approach.

Currently, the program is unable to recognize other gestures and provide appropriate alerts to the user. As previously mentioned, including the "Other" label resulted in decreased model performance. This issue may be due to the limited data collection for the "Other" label. Owing to the time constraints of this project, only 525 samples were collected. For future work, it is recommended to include data for the "Other" label and increase the dataset size to enable this additional functionality.

Bibliography

Shinya, Y., Simo-Serra, E. & Suzuki, T., 2019. Understanding the effects of pre-training for object detectors via eigenspectrum.

Howard, A. G., 1027. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.

Admin, 2022. *Gesture recognition and its application in machine learning.* [Online]
Available at: https://how2electronics.com/gesture-recognition-application-machine-learning/

Jones, R., n.d. *Dr Kawashima's brain training for Nintendo Switch.* [Online]
Available at: https://www.trustedreviews.com/reviews/dr-kawashimas-brain-training-for-nintendo-switch

Brownlee, J., 2019. *A gentle introduction to transfer learning for Deep learning.* [Online]
Available at: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

Admin, 2023. *ESP32 cam object detection & identification with opencv.* [Online]
Available at: https://iotprojectsideas.com/esp32-cam-object-detection-identification-with-opencv/

Declaration of Authorship

I, AUTHORS NAME HERE, confirm that the work presented in this assessment is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Yiqi Huang

ASSESSMENT DATE: 27th, April, 2023