TensorFlow Model for Embedded Object Detection

ECEN 5060 DEEP LEARNING

BRADY BARLOW

Problem Statement

Advances in machine learning have made it possible to implement TensorFlow models on microcontrollers, paving the way for real-time object detection and decision-making in compact, resource-constrained environments. The goal of this project is to develop a modular architecture to scale a large TensorFlow model down to TensorFlow Lite model for deployment on a SparkFun Thing Plus RP2040 microcontroller with an attached ARDUCAM and OLED.

Software Selections

The dataset used for this project is COCO 2017, containing 118,287 images with multilabel annotations. To align with road object detection goals, only bicycle, car, motorcycle, bus, truck, traffic light, and stop sign classes were used.

MobileNetV2 was selected for its efficiency and trained using a two-stage approach to optimize for these specific classes.

Due to current limitations with LiteRT on the RP2040 platform, and with pico-tflmicro being the officially supported TensorFlow Lite Micro implementation for RP2040, TensorFlow was chosen over PyTorch to allow direct model conversion to TFLite and C header files for embedded deployment.

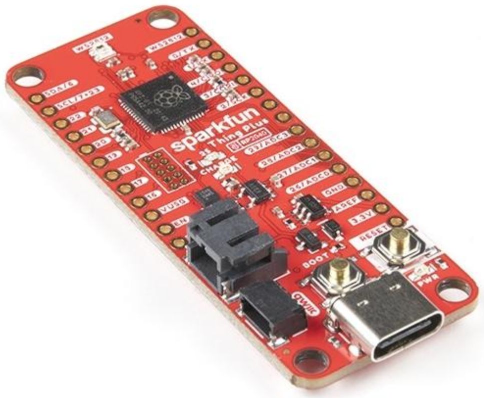
Hardware Selections

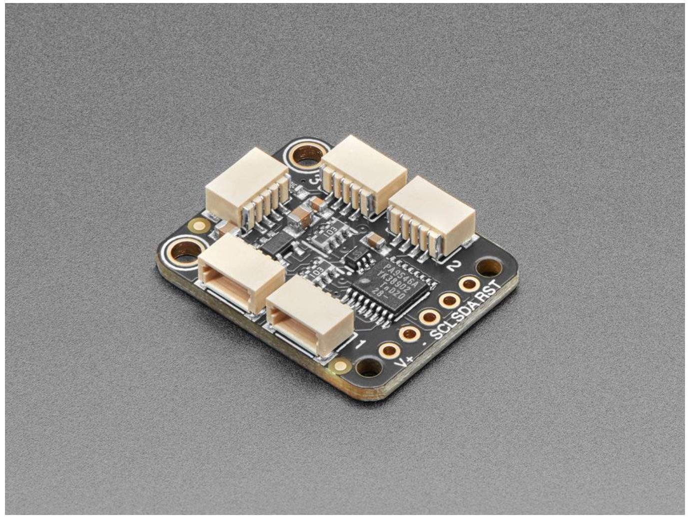
SparkFun Thing Plus - RP2040

Arducam 5MP Plus OV5642 Mini Camera Module

https://www.sparkfun.com/sparkfun-qwiic-oled-display-0-91-in-128x32-lcd-24606.html

Adafruit PCA9546 4-Channel STEMMA QT / Qwiic I2C Multiplexer - TCA9546A Compatible





Configuration

```python
# —— CONFIGURATION ——————————————————————————————
BATCH_SIZE = 32
FROZEN_EPOCHS = 10
FINE_TUNE_EPOCHS = 30
INITIAL_LR = 1e-3
IMG_HEIGHT, IMG_WIDTH = 64, 64
IMG_SIZE = (IMG_HEIGHT, IMG_WIDTH)
SEED = 42
NICKNAME = 'RoadLiteMobileNetV2'
EXPORT_DIR = './export'
DATA_DIR = './Data'
EXCEL_DIR = './excel'
PREDICTION_DIR = './predictions'

# Set seeds for reproducibility
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

# Road object classes to detect
ROAD_CLASSES = ['bicycle', 'car', 'motorcycle', 'bus', 'truck', 'traffic light', 'stop sign']
NUM_CLASSES = len(ROAD_CLASSES)

# Create necessary directories
for directory in [EXPORT_DIR, DATA_DIR, EXCEL_DIR, PREDICTION_DIR]:
    os.makedirs(directory, exist_ok = True)
```

Model Building

```python
# —— MODEL BUILDING ————————————————————————————————
def build_model(trainable_base = False, fine_tuning = False):
    """Build the MobileNetV2-based model"""
    # Load MobileNetV2 with pre-trained weights, using alpha=0.35 for a smaller model
    base_model = MobileNetV2(
        input_shape = (*IMG_SIZE, 3),
        include_top = False,
        weights = 'imagenet',
        alpha = 0.35  # Lighter model
    )

    # Set base model trainable status
    base_model.trainable = trainable_base

    # If fine-tuning, only make the last few layers trainable
    if fine_tuning:
        for layer in base_model.layers[:-20]:
            layer.trainable = False

    # Build model
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dropout(0.5),  # Prevent overfitting
        layers.Dense(128, activation = 'relu'),
        layers.Dropout(0.3),  # Additional dropout
        layers.Dense(NUM_CLASSES, activation = 'sigmoid')
    ])

    # Compile with appropriate learning rate
    model.compile(
        optimizer = tf.keras.optimizers.Adam(
            INITIAL_LR if not fine_tuning else INITIAL_LR * 0.1,
            clipnorm = 1.0  # Gradient clipping for stability
        ),
        loss = 'binary_crossentropy',
        metrics = [
            tf.keras.metrics.BinaryAccuracy(name = 'binary_accuracy'),
            tf.keras.metrics.Precision(name = 'precision'),
            tf.keras.metrics.Recall(name = 'recall'),
            tf.keras.metrics.AUC(name = 'auc')
        ]
    )

    return model
```

Stage 1 Training

```
# ── STAGE 1: FROZEN BASE TRAINING ──────────────────────────────────────────
print("\n=== Stage 1: Training with Frozen Base Model ===")
stage1_model = build_model(trainable_base = False)

# Callbacks
metrics_callback = MetricsCallback(test_ds, ROAD_CLASSES, df_test)
early_stop = EarlyStopping(
    monitor = 'val_f1_macro',
    mode = 'max',
    patience = 7,
    restore_best_weights = True,
    verbose = 1
)
checkpoint = ModelCheckpoint(
    os.path.join(EXPORT_DIR, 'best_model_stage1.keras'),
    monitor = 'val_f1_macro',
    mode = 'max',
    save_best_only = True,
    verbose = 1
)
reduce_lr = ReduceLROnPlateau(
    monitor = 'val_f1_macro',
    mode = 'max',
    factor = 0.5,
    patience = 3,
    min_lr = 1e-6,
    verbose = 1
)

# Train stage 1
history_stage1 = stage1_model.fit(
    train_ds,
    epochs = FROZEN_EPOCHS,
    validation_data = test_ds,
    callbacks = [metrics_callback, early_stop, checkpoint, reduce_lr],
    verbose = 1
)
```

Stage 2 Training

```
# ── STAGE 2: FINE TUNING ──────────────────────────────
print("\n=== Stage 2: Fine-tuning Model ===")
# Load best stage 1 model
stage1_model = tf.keras.models.load_model(
    os.path.join(EXPORT_DIR, 'best_model_stage1.keras')
)

# Create fine-tuning model
stage2_model = build_model(trainable_base = True, fine_tuning = True)

# Copy weights from stage 1
stage2_model.set_weights(stage1_model.get_weights())

# Callbacks for stage 2
metrics_callback_stage2 = MetricsCallback(test_ds, ROAD_CLASSES, df_test)
early_stop_stage2 = EarlyStopping(
    monitor = 'val_f1_macro',
    mode = 'max',
    patience = 10,
    restore_best_weights = True,
    verbose = 1
)
checkpoint_stage2 = ModelCheckpoint(
    os.path.join(EXPORT_DIR, 'best_model_final.keras'),
    monitor = 'val_f1_macro',
    mode = 'max',
    save_best_only = True,
    verbose = 1
)
reduce_lr_stage2 = ReduceLROnPlateau(
    monitor = 'val_f1_macro',
    mode = 'max',
    factor = 0.2,
    patience = 5,
    min_lr = 1e-7,
    verbose = 1
)

# Train stage 2
history_stage2 = stage2_model.fit(
    train_ds,
    epochs = FINE_TUNE_EPOCHS,
    validation_data = test_ds,
    callbacks = [metrics_callback_stage2, early_stop_stage2, checkpoint_stage2, reduce_lr_stage2],
    verbose = 1
)
```

Exporting TFLite

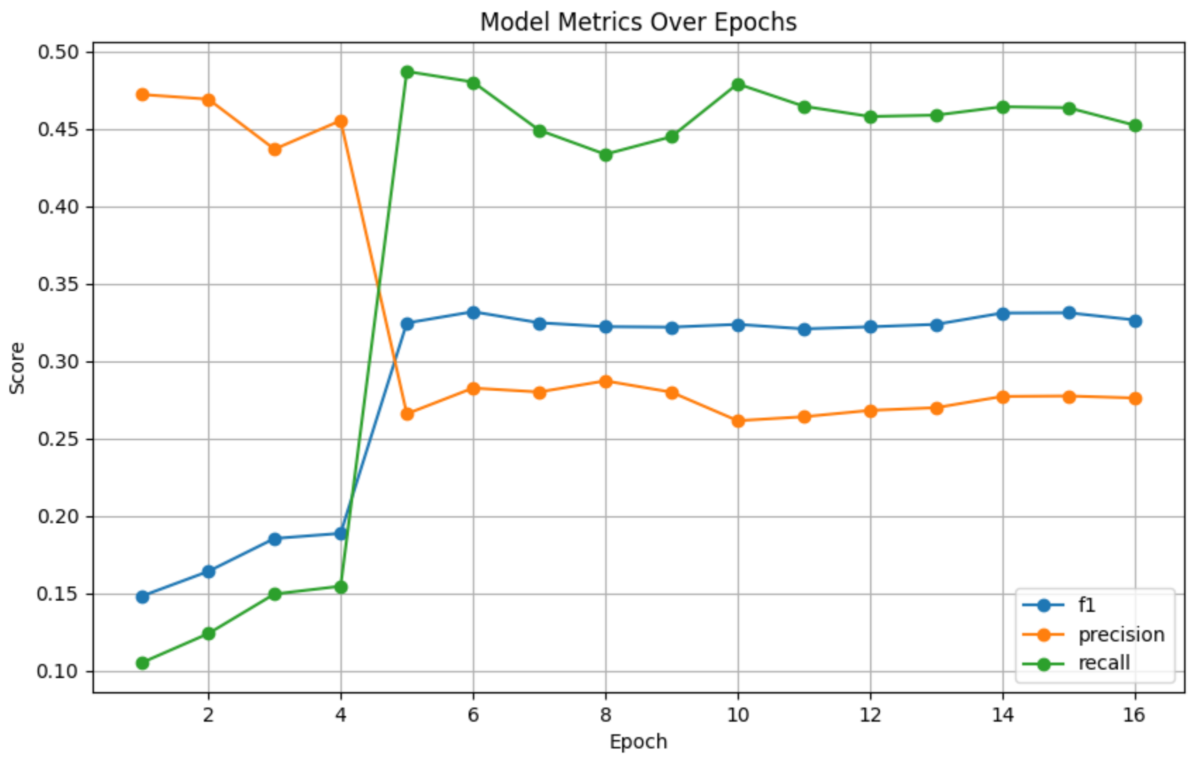
Total params: 1,402,967 (5.35 MB)
Trainable params: 413,943 (1.58 MB)
Non-trainable params: 161,136 (629.44 KB)
Optimizer params: 827,888 (3.16 MB)

TFLite model size: 771.81 KB

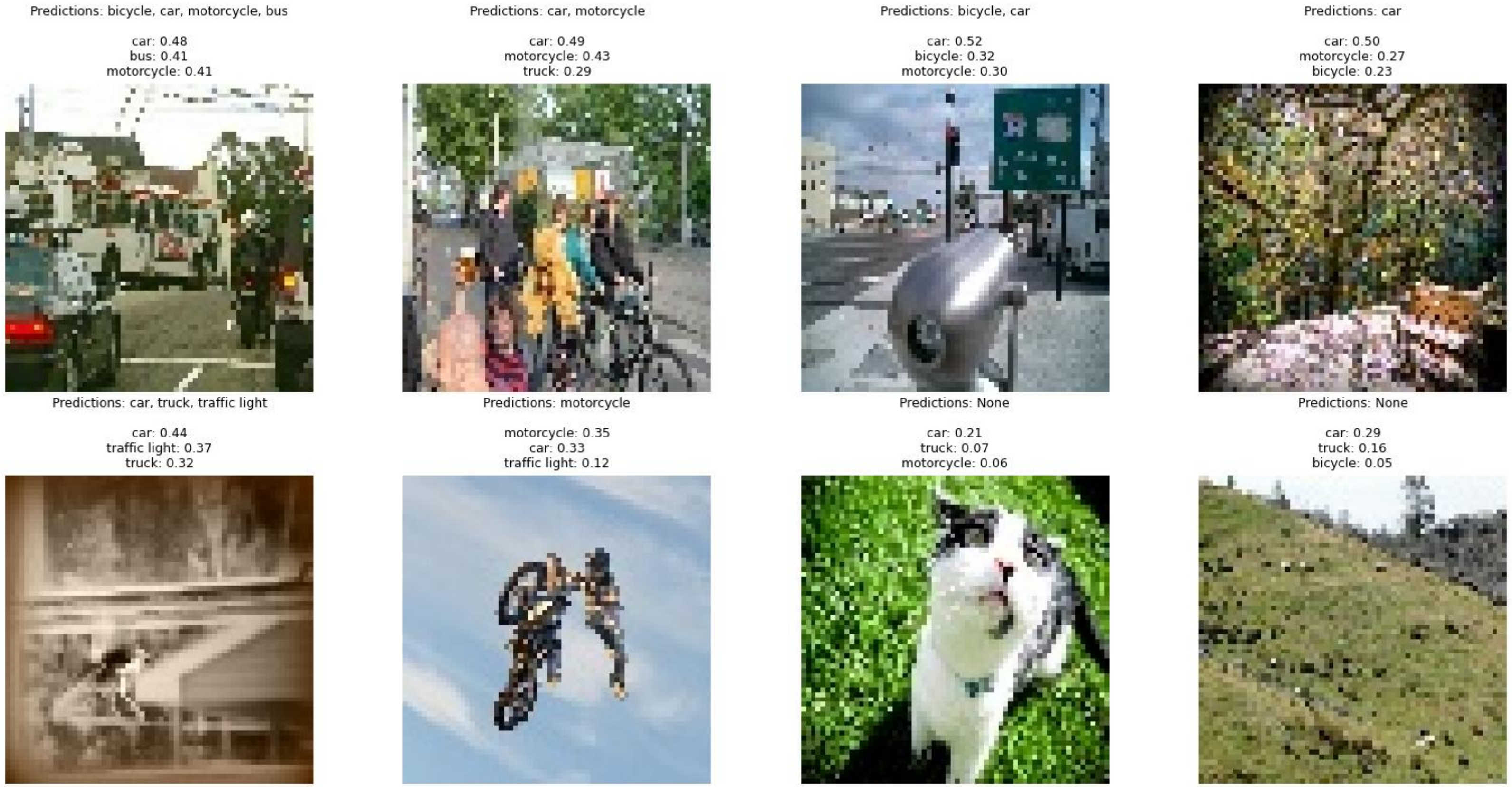
```python
# —— EXPORT TO TFLITE

print("\n=== Converting to TFLite for RP2040 Deployment ===")

# Load the best model
final_model = tf.keras.models.load_model(
    os.path.join(EXPORT_DIR, 'best_model_final.keras')
)

# Summary of the model
final_model.summary()


# Define a representative dataset for quantization
def representative_dataset():
    """Generate representative dataset for quantization"""
    for i in range(100):
        row = df_train.sample(1).iloc[0]
        img_path = os.path.join(DATA_DIR, row['filename'])
        img = cv2.imread(img_path)
        img = cv2.resize(img, IMG_SIZE)
        img = img.astype(np.float32) / 255.0
        yield [np.expand_dims(img, axis = 0)]


# Create TFLite converter
converter = tf.lite.TFLiteConverter.from_keras_model(final_model)

# Apply optimizations for RP2040
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset
converter.target_spec.supported_ops =
[tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.float32
converter.inference_output_type = tf.uint8

# Convert model
tflite_model = converter.convert()

# Save TFLite model
tflite_path = os.path.join(EXPORT_DIR, f"{NICKNAME}.tflite")
with open(tflite_path, 'wb') as f:
    f.write(tflite_model)

print(f"TFLite model saved: {tflite_path}")
print(f"TFLite model size: {len(tflite_model) / 1024:.2f} KB")
```

Keras Prediction

Results From Training

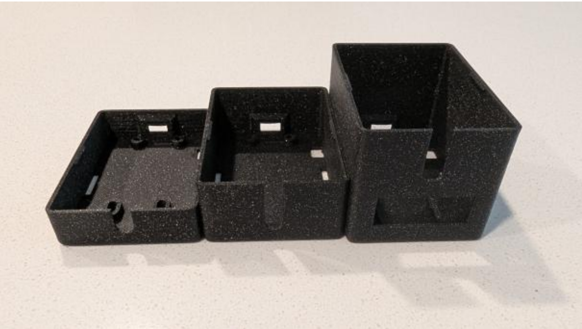


TFLite Prediction



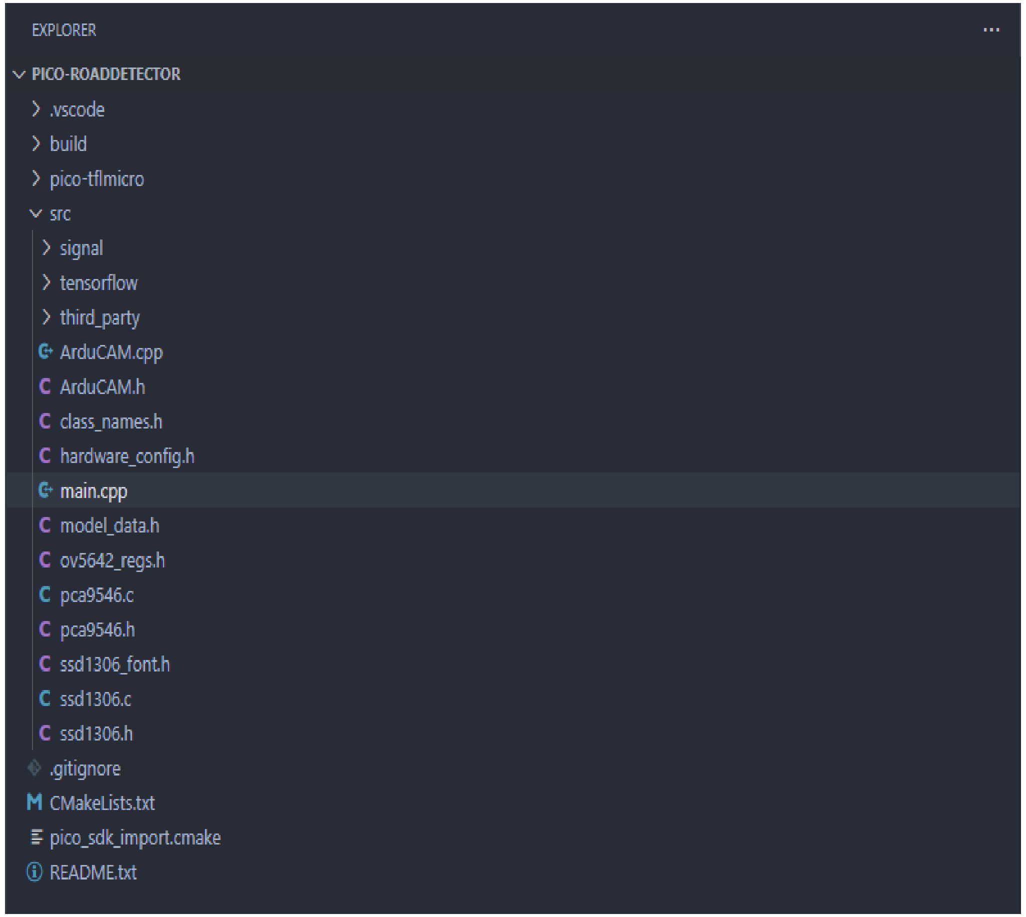
Header Generation And Initial Design Of Enclosure

```python
with open('export/RoadLiteMobileNetV2.tflite', 'rb') as f:
    data = f.read()

with open('export/model_data.h', 'w') as f:
    f.write('const unsigned char model_data[] = {\n')
    for i, b in enumerate(data):
        if i % 12 == 0:
            f.write('\n ')
        f.write(f' 0x{b:02x},')
    f.write('\n};\nconst int model_data_len = sizeof(model_data);\n')
```

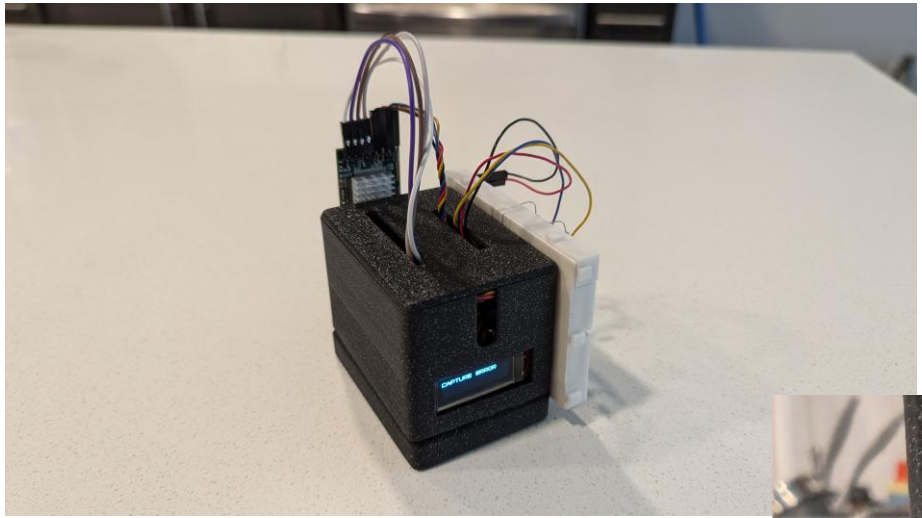


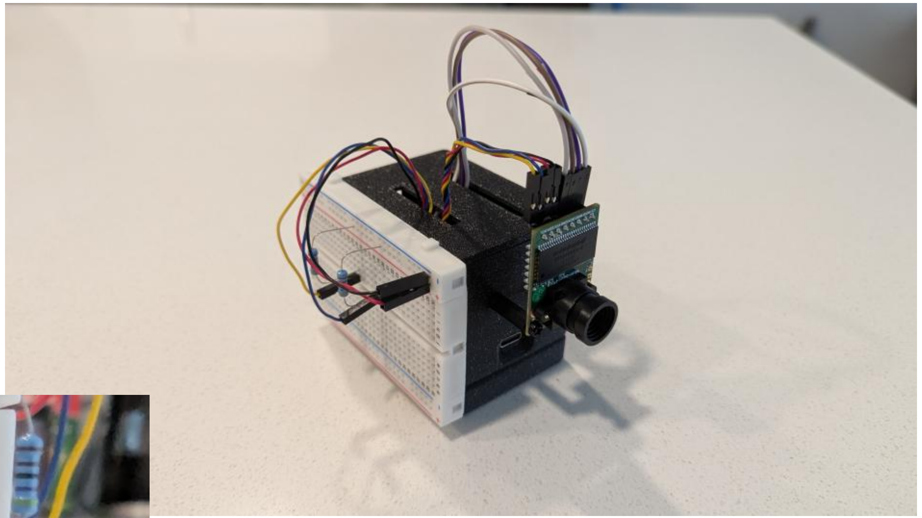
Pico Deployment and results

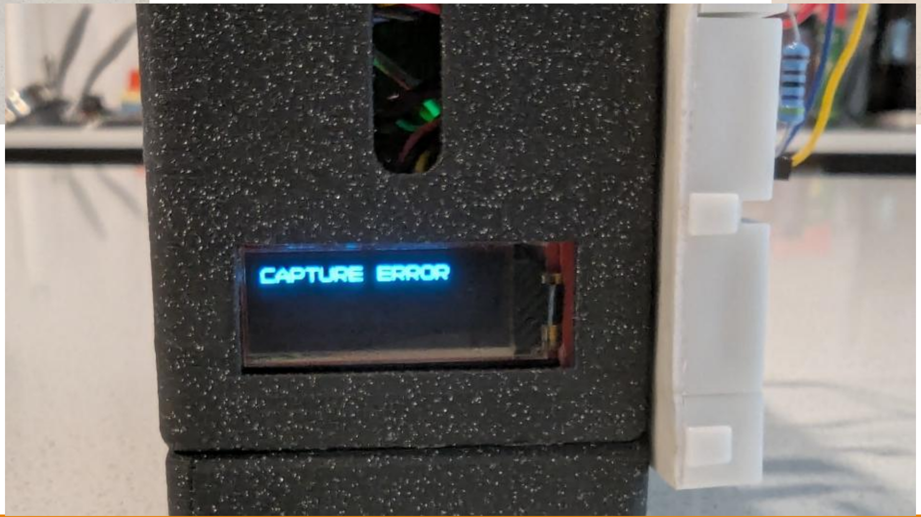
Setting up TensorFlow Lite...
Tensor arena size: 160 KB
Getting model from model_data (790336 bytes)...
Model version: 3, Schema version: 3
Creating op resolver...
Registering operations...
Building interpreter...
Allocating tensors...
Allocation status: 0 (0 is success)
Input tensor type: 1 (kTfLiteInt8=9, kTfLiteUInt8=3, kTfLiteFloat32=1)
Output tensor type: 3 (kTfLiteInt8=9, kTfLiteUInt8=3, kTfLiteFloat32=1)
Input tensor dims: 1 x 64 x 64 x 3
Output tensor dims: 1 x 7
Input tensor details - zero_point: 0, scale: 0.000000
Output tensor details - zero_point: 0, scale: 0.003906
Arena size: 127512 bytes used, 163840 bytes available
TensorFlow Lite ready



Conclusion







References

https://github.com/tensorflow/tflite-micro

https://github.com/raspberrypi/pico-tflmicro

https://cocodataset.org/#home

https://www.sparkfun.com/sparkfun-thing-plus-rp2040.html

https://www.sparkfun.com/arducam-5mp-plus-ov5642-mini-camera-module.html

https://www.sparkfun.com/sparkfun-qwiic-oled-display-0-91-in-128x32-lcd-24606.html

https://www.adafruit.com/product/5664

MORE REFERNCES WILL BE INCLUDED WITHIN FINAL REPORT.