ASL_detection_project

This document contains a complete, runnable ASL detection project you can download and run locally. It assumes your dataset follows this structure:

data/

train/

A/...

B/...

... (all 29 classes)

test/

A/...

B/...

...

Project structure

ASL_detection_project/

├── data/ # (your dataset - not included)

├── models/

├── src/

| ├── train.py

| ├── evaluate.py

| └── predict.py

├── requirements.txt

└── README.md

---

requirements.txt

tensorflow>=2.10

numpy

matplotlib

scikit-learn

opencv-python

pillow

README.md

ASL Detection

Setup

1. Create a Python 3.8+ virtual environment and install requirements:

```bash
python -m venv venv
source venv/bin/activate   # or .\venv\Scripts\activate on Windows
pip install -r requirements.txt
```

2. Put your dataset in data/train and data/test with one subfolder per class (29 classes).
3. Train the model:

```
python src/train.py --data_dir data --img_size 128 --batch_size 32 --epochs 30
```

4. Evaluate:

```
python src/evaluate.py --data_dir data --img_size 128 --batch_size 32 --model_path models/asl_mobilenetv2.h5
```

5. (Optional) Run webcam demo:

```
python src/predict.py --model_path models/asl_mobilenetv2.h5 --img_size 128
```

---

src/train.py

```python
"""Train a transfer-learning ASL classifier using MobileNetV2.

Saves model to models/asl_mobilenetv2.h5 and a saved_model folder.
"""
import argparse
import os
```

```
from pathlib import Path
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau


def get_datasets(data_dir, img_size, batch_size):
    train_dir = os.path.join(data_dir, 'train')
    val_dir = os.path.join(data_dir, 'test')

    train_ds = tf.keras.preprocessing.image_dataset_from_directory(
        train_dir,
        labels='inferred',
        label_mode='categorical',
        batch_size=batch_size,
        image_size=(img_size, img_size),
        shuffle=True
    )

    val_ds = tf.keras.preprocessing.image_dataset_from_directory(
        val_dir,
        labels='inferred',
        label_mode='categorical',
        batch_size=batch_size,
        image_size=(img_size, img_size),
        shuffle=False
    )

    class_names = train_ds.class_names
```

```
    # Prefetch for performance
    AUTOTUNE = tf.data.AUTOTUNE
    train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)
    val_ds = val_ds.prefetch(buffer_size=AUTOTUNE)

    return train_ds, val_ds, class_names


def build_model(num_classes, img_size, fine_tune_at=100):
    base_model = MobileNetV2(input_shape=(img_size, img_size, 3), include_top=False,
weights='imagenet')
    base_model.trainable = True

    # Freeze early layers
    for layer in base_model.layers[:fine_tune_at]:
        layer.trainable = False

    inputs = layers.Input(shape=(img_size, img_size, 3))
    x = layers.Rescaling(1./127.5, offset=-1)(inputs)  # MobileNetV2 preprocessing

    # Data augmentation (simple)
    data_augmentation = tf.keras.Sequential([
        layers.RandomFlip('horizontal'),
        layers.RandomRotation(0.08),
        layers.RandomZoom(0.08),
    ])
    x = data_augmentation(x)

    x = base_model(x, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.3)(x)
```

```
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(inputs, outputs)

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model


def main(args):
    os.makedirs('models', exist_ok=True)

    train_ds, val_ds, class_names = get_datasets(args.data_dir, args.img_size, args.batch_size)
    num_classes = len(class_names)

    model = build_model(num_classes, args.img_size, fine_tune_at=100)
    model.summary()

    checkpoint = ModelCheckpoint('models/asl_mobilenetv2.h5', monitor='val_accuracy',
save_best_only=True, verbose=1)
    early = EarlyStopping(monitor='val_loss', patience=6, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-7)

    history = model.fit(
        train_ds,
        validation_data=val_ds,
        epochs=args.epochs,
        callbacks=[checkpoint, early, reduce_lr]
    )
```

```
    # Save final model (also saved_model format)
    model.save('models/asl_mobilenetv2.h5')
    model.save('models/asl_mobilenetv2_saved')


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir', type=str, default='data')
    parser.add_argument('--img_size', type=int, default=128)
    parser.add_argument('--batch_size', type=int, default=32)
    parser.add_argument('--epochs', type=int, default=30)
    args = parser.parse_args()
    main(args)
```

---

src/evaluate.py

```
"""Evaluate model on test set and show confusion matrix + per-class accuracy."""
import argparse
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import itertools
import tensorflow as tf


def get_dataset(data_dir, img_size, batch_size):
    test_dir = os.path.join(data_dir, 'test')
    ds = tf.keras.preprocessing.image_dataset_from_directory(
        test_dir,
        labels='inferred',
        label_mode='categorical',
```

```
        batch_size=batch_size,
        image_size=(img_size, img_size),
        shuffle=False
    )
    return ds


def plot_confusion_matrix(cm, class_names, out_path='confusion_matrix.png'):
    fig = plt.figure(figsize=(12, 10))
    plt.imshow(cm, interpolation='nearest')
    plt.title('Confusion matrix')
    plt.colorbar()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names, rotation=90)
    plt.yticks(tick_marks, class_names)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    fig.savefig(out_path)
    print(f"Saved confusion matrix to {out_path}")


def main(args):
    ds = get_dataset(args.data_dir, args.img_size, args.batch_size)
```

```
    class_names = ds.class_names

    model = tf.keras.models.load_model(args.model_path)

    y_true = []
    y_pred = []

    for images, labels in ds:
        preds = model.predict(images)
        y_true.extend(np.argmax(labels.numpy(), axis=1).tolist())
        y_pred.extend(np.argmax(preds, axis=1).tolist())

    cm = confusion_matrix(y_true, y_pred)
    plot_confusion_matrix(cm, class_names, out_path='models/confusion_matrix.png')

    print('\nClassification report:\n')
    print(classification_report(y_true, y_pred, target_names=class_names))

    # Per-class accuracy
    class_acc = cm.diagonal() / cm.sum(axis=1)
    for name, acc in zip(class_names, class_acc):
        print(f"{name}: {acc:.3f}")


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir', type=str, default='data')
    parser.add_argument('--img_size', type=int, default=128)
    parser.add_argument('--batch_size', type=int, default=32)
    parser.add_argument('--model_path', type=str, default='models/asl_mobilenetv2.h5')
    args = parser.parse_args()
```

```
    main(args)
```

---

src/predict.py

```
"""Simple webcam demo that predicts ASL signs in real-time.

Press 'q' to quit.
"""
import argparse
import cv2
import numpy as np
import tensorflow as tf



def main(args):
    model = tf.keras.models.load_model(args.model_path)
    img_size = args.img_size

    # We need class names. If the model was trained with Keras Dataset, we will read from data/train
    import os
    classes = sorted(os.listdir(os.path.join(args.data_dir, 'train')))

    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print('Cannot open webcam')
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            break
```

```
        # Flip and resize
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        img = cv2.resize(frame_rgb, (img_size, img_size))
        img = img.astype('float32') / 127.5 - 1.0  # MobileNetV2 preprocessing
        input_arr = np.expand_dims(img, axis=0)

        preds = model.predict(input_arr)
        idx = np.argmax(preds[0])
        label = classes[idx]
        prob = preds[0][idx]

        # Overlay
        cv2.putText(frame, f"{label} ({prob:.2f})", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)
        cv2.imshow('ASL demo', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--model_path', type=str, default='models/asl_mobilenetv2.h5')
    parser.add_argument('--img_size', type=int, default=128)
    parser.add_argument('--data_dir', type=str, default='data')
    args = parser.parse_args()
    main(args)
```

Notes & Tips

- If your dataset is small, consider heavier augmentation or K-fold cross validation.
- To reduce model size for deployment, try tf.keras.models.save_model(..., include_optimizer=False) and/or convert to TFLite.
- If training on CPU is slow, use a GPU runtime (local GPU or Colab).

If you'd like, I can also:

- Convert this into a downloadable ZIP.
- Add a Colab notebook that mounts your Google Drive dataset and trains on GPU.
- Replace MobileNetV2 with a custom CNN if you prefer a smaller-from-scratch model.

Tell me which of the above you'd like next.