COMPARISON_MobileNetV2

June 21, 2025

[1]:
```
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
import os
from PIL import Image, UnidentifiedImageError
import shutil

# Configuration
IMG_SIZE = (96, 96)
BATCH_SIZE = 32
VALIDATION_SPLIT = 0.4
SEED = 42
ROOT_PATH = ''
DATASET_PATH = os.path.join(ROOT_PATH,"raw_data")
CORRUPT_PATH = os.path.join(ROOT_PATH,"corrupt_images")
os.makedirs(CORRUPT_PATH, exist_ok=True)

for root, dirs, files in os.walk(DATASET_PATH):
    for file in files:
        ext = os.path.splitext(file)[1].lower()
        if ext in [".jpg", ".jpeg", ".png", ".bmp", ".gif"]:
            path = os.path.join(root, file)
            try:
                with Image.open(path) as img:
                    img.verify()  # Check integrity
            except (UnidentifiedImageError, OSError, IOError) as e:
                # Move the corrupt image
                print(f"Corrupt image found: {path} - moving to {CORRUPT_PATH}")
                dest_path = os.path.join(CORRUPT_PATH, os.path.relpath(path,␣
 ↪DATASET_PATH))
                os.makedirs(os.path.dirname(dest_path), exist_ok=True)
                shutil.move(path, dest_path)

LANDMARK_DIR = os.path.join(ROOT_PATH,"data")
RAW_IMAGE_DIR = os.path.join(ROOT_PATH,"raw_data")
FILTERED_IMAGE_DIR = os.path.join(ROOT_PATH,"filtered_raw_data")
DATASET_PATH = FILTERED_IMAGE_DIR
# Supported image extensions
```

```
IMAGE_EXTENSIONS = ['.jpg', '.jpeg', '.png', '.bmp']

# Create filtered output structure
os.makedirs(FILTERED_IMAGE_DIR, exist_ok=True)

for class_name in os.listdir(LANDMARK_DIR):
    if class_name == 'debug':
        continue
    landmark_class_dir = os.path.join(LANDMARK_DIR, class_name)
    raw_class_dir = os.path.join(RAW_IMAGE_DIR, class_name)
    filtered_class_dir = os.path.join(FILTERED_IMAGE_DIR, class_name)
    os.makedirs(filtered_class_dir, exist_ok=True)

    for file in os.listdir(landmark_class_dir):
        if not file.endswith("_landmarks.json"):
            continue

        # Get base filename without "_landmarks.json"
        base_name = file.replace("_landmarks.json", "")

        # Look for corresponding image in raw directory
        for ext in IMAGE_EXTENSIONS:
            image_file = os.path.join(raw_class_dir, base_name + ext)
            if os.path.exists(image_file):
                # Copy to filtered folder
                shutil.copy(image_file, os.path.join(filtered_class_dir, os.
 ↪path.basename(image_file)))
                break

# Load training dataset with validation split
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    DATASET_PATH,
    validation_split=VALIDATION_SPLIT,
    subset="training",
    seed=SEED,
    color_mode="rgb",
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE
)
num_classes = len(train_ds.class_names)
label_map = train_ds.class_names

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    DATASET_PATH,
    validation_split=VALIDATION_SPLIT,
    subset="validation",
    seed=SEED,
```

```
    color_mode="rgb",
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE
)

test_ds = val_ds.shard(2,0)
val_ds = val_ds.shard(2,1)
# Normalize pixel values to [0, 1]
normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds = test_ds.map(lambda x, y: (normalization_layer(x), y))
# Cache and prefetch for performance
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
Found 3488 files belonging to 26 classes.
Using 2093 files for training.
Found 3488 files belonging to 26 classes.
Using 1395 files for validation.
```

[2]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout,␣
 ↪BatchNormalization
from tensorflow.keras.layers import Flatten, Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import MobileNetV2
base_model = MobileNetV2(input_shape=(96, 96, 3), include_top=False,␣
 ↪weights='imagenet')
base_model.trainable = False

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
])

model.compile(optimizer=Adam(1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[3]:
```
history = model.fit(train_ds, validation_data=val_ds, epochs=50)
```

```
Epoch 1/50
66/66             25s 176ms/step -
accuracy: 0.0594 - loss: 3.6202 - val_accuracy: 0.1650 - val_loss: 2.9301
Epoch 2/50
66/66             4s 62ms/step -
accuracy: 0.1798 - loss: 2.8242 - val_accuracy: 0.2547 - val_loss: 2.4942
Epoch 3/50
66/66             4s 62ms/step -
accuracy: 0.2942 - loss: 2.3955 - val_accuracy: 0.3357 - val_loss: 2.1721
Epoch 4/50
66/66             4s 63ms/step -
accuracy: 0.3880 - loss: 2.0473 - val_accuracy: 0.3835 - val_loss: 1.9820
Epoch 5/50
66/66             4s 62ms/step -
accuracy: 0.4759 - loss: 1.7452 - val_accuracy: 0.3994 - val_loss: 1.9093
Epoch 6/50
66/66             4s 62ms/step -
accuracy: 0.5066 - loss: 1.5541 - val_accuracy: 0.4197 - val_loss: 1.8164
Epoch 7/50
66/66             4s 61ms/step -
accuracy: 0.5838 - loss: 1.3988 - val_accuracy: 0.4327 - val_loss: 1.7915
Epoch 8/50
66/66             4s 63ms/step -
accuracy: 0.5724 - loss: 1.3272 - val_accuracy: 0.4573 - val_loss: 1.7481
Epoch 9/50
66/66             4s 62ms/step -
accuracy: 0.6124 - loss: 1.2087 - val_accuracy: 0.4616 - val_loss: 1.7505
Epoch 10/50
66/66             4s 61ms/step -
accuracy: 0.6212 - loss: 1.1270 - val_accuracy: 0.4515 - val_loss: 1.6986
Epoch 11/50
66/66             4s 61ms/step -
accuracy: 0.6875 - loss: 1.0018 - val_accuracy: 0.4544 - val_loss: 1.7240
Epoch 12/50
66/66             4s 63ms/step -
accuracy: 0.6933 - loss: 0.9697 - val_accuracy: 0.4602 - val_loss: 1.7199
Epoch 13/50
66/66             4s 63ms/step -
accuracy: 0.6931 - loss: 0.9005 - val_accuracy: 0.4776 - val_loss: 1.6800
Epoch 14/50
66/66             4s 66ms/step -
accuracy: 0.7368 - loss: 0.8183 - val_accuracy: 0.4978 - val_loss: 1.6865
Epoch 15/50
66/66             4s 66ms/step -
accuracy: 0.7322 - loss: 0.7962 - val_accuracy: 0.4660 - val_loss: 1.6973
Epoch 16/50
66/66             4s 66ms/step -
accuracy: 0.7792 - loss: 0.7181 - val_accuracy: 0.4863 - val_loss: 1.6713
```

```
Epoch 17/50
66/66             4s 66ms/step -
accuracy: 0.7809 - loss: 0.6868 - val_accuracy: 0.4949 - val_loss: 1.6599
Epoch 18/50
66/66             4s 66ms/step -
accuracy: 0.7694 - loss: 0.6880 - val_accuracy: 0.4863 - val_loss: 1.6965
Epoch 19/50
66/66             4s 65ms/step -
accuracy: 0.7796 - loss: 0.6509 - val_accuracy: 0.4805 - val_loss: 1.7247
Epoch 20/50
66/66             5s 70ms/step -
accuracy: 0.8041 - loss: 0.5981 - val_accuracy: 0.4848 - val_loss: 1.7109
Epoch 21/50
66/66             5s 73ms/step -
accuracy: 0.8091 - loss: 0.5804 - val_accuracy: 0.4920 - val_loss: 1.7134
Epoch 22/50
66/66             5s 75ms/step -
accuracy: 0.8164 - loss: 0.5750 - val_accuracy: 0.4689 - val_loss: 1.7871
Epoch 23/50
66/66             5s 72ms/step -
accuracy: 0.8162 - loss: 0.5566 - val_accuracy: 0.4660 - val_loss: 1.7657
Epoch 24/50
66/66             4s 68ms/step -
accuracy: 0.8206 - loss: 0.5284 - val_accuracy: 0.4834 - val_loss: 1.7703
Epoch 25/50
66/66             4s 65ms/step -
accuracy: 0.8607 - loss: 0.4466 - val_accuracy: 0.4819 - val_loss: 1.7550
Epoch 26/50
66/66             4s 65ms/step -
accuracy: 0.8357 - loss: 0.4684 - val_accuracy: 0.4848 - val_loss: 1.7584
Epoch 27/50
66/66             4s 65ms/step -
accuracy: 0.8261 - loss: 0.5242 - val_accuracy: 0.4790 - val_loss: 1.8045
Epoch 28/50
66/66             4s 65ms/step -
accuracy: 0.8398 - loss: 0.4429 - val_accuracy: 0.4472 - val_loss: 1.8468
Epoch 29/50
66/66             4s 67ms/step -
accuracy: 0.8634 - loss: 0.4220 - val_accuracy: 0.4674 - val_loss: 1.8495
Epoch 30/50
66/66             4s 67ms/step -
accuracy: 0.8637 - loss: 0.4270 - val_accuracy: 0.4761 - val_loss: 1.8739
Epoch 31/50
66/66             4s 65ms/step -
accuracy: 0.8698 - loss: 0.4159 - val_accuracy: 0.4776 - val_loss: 1.8587
Epoch 32/50
66/66             4s 67ms/step -
accuracy: 0.8747 - loss: 0.3784 - val_accuracy: 0.4703 - val_loss: 1.9077
```

```
Epoch 33/50
66/66             4s 67ms/step -
accuracy: 0.8695 - loss: 0.3635 - val_accuracy: 0.4790 - val_loss: 1.8692
Epoch 34/50
66/66             4s 66ms/step -
accuracy: 0.8615 - loss: 0.4325 - val_accuracy: 0.4703 - val_loss: 1.8776
Epoch 35/50
66/66             4s 66ms/step -
accuracy: 0.8644 - loss: 0.3907 - val_accuracy: 0.4993 - val_loss: 1.8267
Epoch 36/50
66/66             4s 67ms/step -
accuracy: 0.8939 - loss: 0.3498 - val_accuracy: 0.4863 - val_loss: 1.9108
Epoch 37/50
66/66             4s 66ms/step -
accuracy: 0.8873 - loss: 0.3446 - val_accuracy: 0.4805 - val_loss: 1.8563
Epoch 38/50
66/66             4s 65ms/step -
accuracy: 0.8877 - loss: 0.3212 - val_accuracy: 0.4863 - val_loss: 1.9237
Epoch 39/50
66/66             4s 66ms/step -
accuracy: 0.8871 - loss: 0.3336 - val_accuracy: 0.4978 - val_loss: 1.9210
Epoch 40/50
66/66             4s 66ms/step -
accuracy: 0.9095 - loss: 0.3106 - val_accuracy: 0.4747 - val_loss: 1.9156
Epoch 41/50
66/66             4s 66ms/step -
accuracy: 0.9038 - loss: 0.3031 - val_accuracy: 0.4834 - val_loss: 1.8933
Epoch 42/50
66/66             4s 67ms/step -
accuracy: 0.9071 - loss: 0.2997 - val_accuracy: 0.4848 - val_loss: 1.9062
Epoch 43/50
66/66             4s 66ms/step -
accuracy: 0.9003 - loss: 0.3089 - val_accuracy: 0.4920 - val_loss: 1.8998
Epoch 44/50
66/66             4s 67ms/step -
accuracy: 0.9061 - loss: 0.2978 - val_accuracy: 0.4906 - val_loss: 1.8853
Epoch 45/50
66/66             4s 66ms/step -
accuracy: 0.8956 - loss: 0.3134 - val_accuracy: 0.4747 - val_loss: 1.9886
Epoch 46/50
66/66             4s 66ms/step -
accuracy: 0.9021 - loss: 0.2785 - val_accuracy: 0.4776 - val_loss: 1.9685
Epoch 47/50
66/66             4s 67ms/step -
accuracy: 0.9131 - loss: 0.2804 - val_accuracy: 0.4848 - val_loss: 1.9265
Epoch 48/50
66/66             4s 66ms/step -
accuracy: 0.8990 - loss: 0.3112 - val_accuracy: 0.4776 - val_loss: 1.9597
```

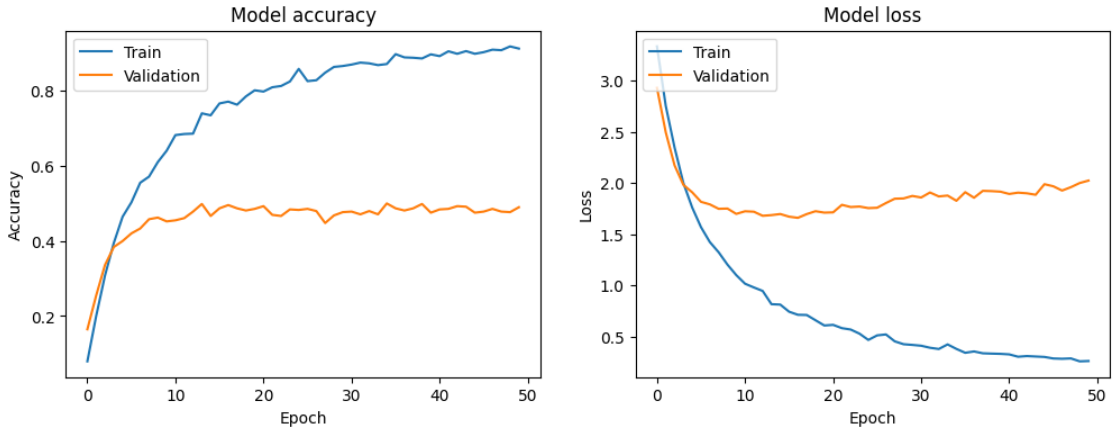
```
Epoch 49/50
66/66             4s 66ms/step -
accuracy: 0.9157 - loss: 0.2496 - val_accuracy: 0.4761 - val_loss: 2.0000
Epoch 50/50
66/66             4s 66ms/step -
accuracy: 0.9124 - loss: 0.2465 - val_accuracy: 0.4891 - val_loss: 2.0238
```

[4]:
```python
test_loss, test_accuracy = model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
22/22             7s 328ms/step -
accuracy: 0.4467 - loss: 2.2660
Test Accuracy: 0.4702
Test Loss: 2.2137
```

[5]:
```python
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import numpy as np
```

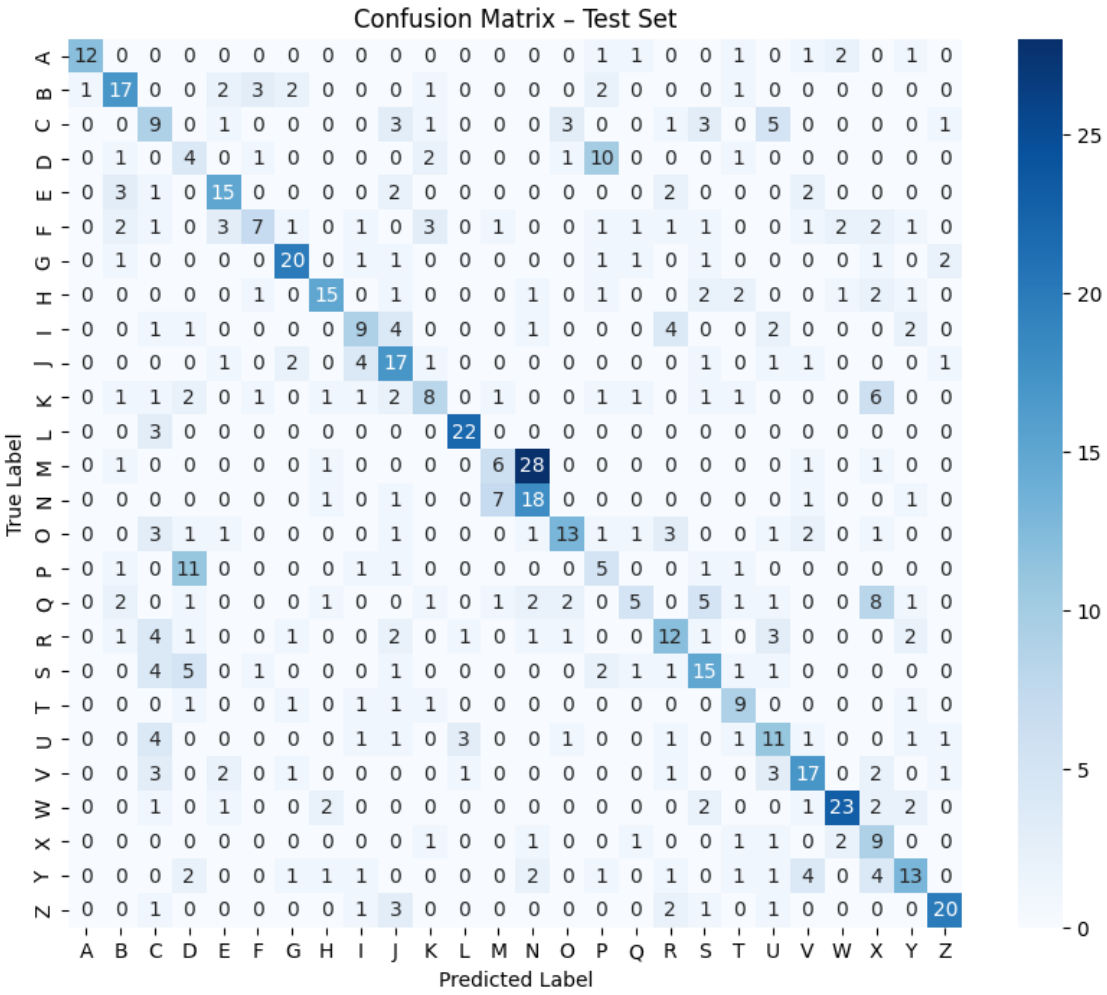
[6]:
```python
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



[7]:
```
y_true, y_pred = [], []
target_names = [label_map[i] for i in range(len(label_map))]
for X_batch, y_batch in test_ds:
    y_true.append(y_batch.numpy())

    batch_pred = model.predict(X_batch, verbose=0)
    y_pred.append(np.argmax(batch_pred, axis=1))

y_true = np.concatenate(y_true)
y_pred = np.concatenate(y_pred)

print(classification_report(
    y_true, y_pred,
    digits=3,
    target_names=target_names
))

cm = confusion_matrix(y_true, y_pred, labels=range(len(label_map)))
labels = [label_map[i] for i in range(len(label_map))]

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Test Set")
plt.show()
```

```
              precision    recall  f1-score   support

           A      0.923     0.632     0.750        19
           B      0.567     0.586     0.576        29
```

C	0.250	0.333	0.286	27
D	0.138	0.200	0.163	20
E	0.577	0.600	0.588	25
F	0.500	0.241	0.326	29
G	0.690	0.690	0.690	29
H	0.682	0.556	0.612	27
I	0.429	0.375	0.400	24
J	0.415	0.586	0.486	29
K	0.421	0.286	0.340	28
L	0.815	0.880	0.846	25
M	0.375	0.158	0.222	38
N	0.327	0.621	0.429	29
O	0.619	0.448	0.520	29
P	0.192	0.238	0.213	21
Q	0.417	0.161	0.233	31
R	0.414	0.400	0.407	30
S	0.441	0.469	0.455	32
T	0.429	0.600	0.500	15
U	0.355	0.423	0.386	26
V	0.531	0.548	0.540	31
W	0.767	0.676	0.719	34
X	0.237	0.562	0.333	16
Y	0.500	0.406	0.448	32
Z	0.769	0.690	0.727	29
accuracy			0.470	704
macro avg	0.491	0.476	0.469	704
weighted avg	0.498	0.470	0.470	704


Confusion Matrix – Test Set
True Label
Predicted Label
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
25
20
15
10
5
0