COMPARISON_MobileNetV2

June 21, 2025

[1]:
```python
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 1691 files belonging to 26 classes.
Using 1015 files for training.
Found 1691 files belonging to 26 classes.
Using 676 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
32/32           15s 198ms/step -
accuracy: 0.0547 - loss: 3.7721 - val_accuracy: 0.1636 - val_loss: 3.0626
Epoch 2/50
32/32           2s 61ms/step -
accuracy: 0.1725 - loss: 2.9055 - val_accuracy: 0.2438 - val_loss: 2.7551
Epoch 3/50
32/32           2s 65ms/step -
accuracy: 0.2634 - loss: 2.5440 - val_accuracy: 0.2932 - val_loss: 2.4769
Epoch 4/50
32/32           2s 67ms/step -
accuracy: 0.3473 - loss: 2.2006 - val_accuracy: 0.3302 - val_loss: 2.3144
Epoch 5/50
32/32           2s 68ms/step -
accuracy: 0.4187 - loss: 1.9388 - val_accuracy: 0.3704 - val_loss: 2.1759
Epoch 6/50
32/32           2s 64ms/step -
accuracy: 0.4847 - loss: 1.6540 - val_accuracy: 0.3951 - val_loss: 2.0827
Epoch 7/50
32/32           2s 63ms/step -
accuracy: 0.5521 - loss: 1.4912 - val_accuracy: 0.4074 - val_loss: 1.9792
Epoch 8/50
32/32           2s 64ms/step -
accuracy: 0.5599 - loss: 1.3213 - val_accuracy: 0.3920 - val_loss: 1.9830
Epoch 9/50
32/32           2s 62ms/step -
accuracy: 0.6516 - loss: 1.0817 - val_accuracy: 0.3981 - val_loss: 1.9734
Epoch 10/50
32/32           2s 62ms/step -
accuracy: 0.6559 - loss: 1.0572 - val_accuracy: 0.3827 - val_loss: 1.9619
Epoch 11/50
32/32           2s 62ms/step -
accuracy: 0.7437 - loss: 0.8752 - val_accuracy: 0.3765 - val_loss: 1.9411
Epoch 12/50
32/32           2s 65ms/step -
accuracy: 0.7457 - loss: 0.8193 - val_accuracy: 0.4198 - val_loss: 1.9009
Epoch 13/50
32/32           2s 63ms/step -
accuracy: 0.7947 - loss: 0.7212 - val_accuracy: 0.4198 - val_loss: 1.9385
Epoch 14/50
32/32           2s 62ms/step -
accuracy: 0.7742 - loss: 0.6841 - val_accuracy: 0.4290 - val_loss: 1.9604
Epoch 15/50
32/32           2s 63ms/step -
accuracy: 0.8037 - loss: 0.6217 - val_accuracy: 0.4414 - val_loss: 1.9576
Epoch 16/50
32/32           2s 71ms/step -
accuracy: 0.8315 - loss: 0.5755 - val_accuracy: 0.4167 - val_loss: 1.9722
```

```
Epoch 17/50
32/32            2s 66ms/step -
accuracy: 0.8370 - loss: 0.5185 - val_accuracy: 0.4414 - val_loss: 2.0844
Epoch 18/50
32/32            2s 71ms/step -
accuracy: 0.8533 - loss: 0.4822 - val_accuracy: 0.4321 - val_loss: 1.9409
Epoch 19/50
32/32            2s 70ms/step -
accuracy: 0.8436 - loss: 0.4679 - val_accuracy: 0.4228 - val_loss: 2.0139
Epoch 20/50
32/32            2s 69ms/step -
accuracy: 0.8762 - loss: 0.4021 - val_accuracy: 0.4321 - val_loss: 2.0611
Epoch 21/50
32/32            2s 66ms/step -
accuracy: 0.8855 - loss: 0.4012 - val_accuracy: 0.4136 - val_loss: 2.0505
Epoch 22/50
32/32            2s 72ms/step -
accuracy: 0.8861 - loss: 0.3923 - val_accuracy: 0.4290 - val_loss: 2.0719
Epoch 23/50
32/32            2s 69ms/step -
accuracy: 0.8995 - loss: 0.3476 - val_accuracy: 0.4228 - val_loss: 2.0445
Epoch 24/50
32/32            2s 66ms/step -
accuracy: 0.9040 - loss: 0.3169 - val_accuracy: 0.4259 - val_loss: 2.0573
Epoch 25/50
32/32            2s 64ms/step -
accuracy: 0.9135 - loss: 0.3170 - val_accuracy: 0.4228 - val_loss: 2.0572
Epoch 26/50
32/32            2s 65ms/step -
accuracy: 0.9103 - loss: 0.2807 - val_accuracy: 0.4228 - val_loss: 2.0662
Epoch 27/50
32/32            2s 63ms/step -
accuracy: 0.9165 - loss: 0.3002 - val_accuracy: 0.4136 - val_loss: 2.0864
Epoch 28/50
32/32            2s 65ms/step -
accuracy: 0.9165 - loss: 0.2881 - val_accuracy: 0.4259 - val_loss: 2.1015
Epoch 29/50
32/32            2s 67ms/step -
accuracy: 0.9221 - loss: 0.2491 - val_accuracy: 0.4259 - val_loss: 2.1427
Epoch 30/50
32/32            2s 75ms/step -
accuracy: 0.9359 - loss: 0.2189 - val_accuracy: 0.3951 - val_loss: 2.2017
Epoch 31/50
32/32            2s 77ms/step -
accuracy: 0.9301 - loss: 0.2204 - val_accuracy: 0.4198 - val_loss: 2.1645
Epoch 32/50
32/32            3s 80ms/step -
accuracy: 0.9436 - loss: 0.1867 - val_accuracy: 0.4198 - val_loss: 2.2177
```

```
Epoch 33/50
32/32           3s 91ms/step -
accuracy: 0.9228 - loss: 0.2342 - val_accuracy: 0.4074 - val_loss: 2.1982
Epoch 34/50
32/32           3s 84ms/step -
accuracy: 0.9474 - loss: 0.2038 - val_accuracy: 0.4383 - val_loss: 2.2785
Epoch 35/50
32/32           3s 83ms/step -
accuracy: 0.9377 - loss: 0.2192 - val_accuracy: 0.4167 - val_loss: 2.2164
Epoch 36/50
32/32           3s 81ms/step -
accuracy: 0.9299 - loss: 0.2273 - val_accuracy: 0.4259 - val_loss: 2.2419
Epoch 37/50
32/32           3s 80ms/step -
accuracy: 0.9321 - loss: 0.2320 - val_accuracy: 0.4352 - val_loss: 2.1573
Epoch 38/50
32/32           2s 72ms/step -
accuracy: 0.9457 - loss: 0.2051 - val_accuracy: 0.4228 - val_loss: 2.2485
Epoch 39/50
32/32           2s 76ms/step -
accuracy: 0.9604 - loss: 0.1662 - val_accuracy: 0.4043 - val_loss: 2.3764
Epoch 40/50
32/32           2s 69ms/step -
accuracy: 0.9451 - loss: 0.1824 - val_accuracy: 0.4136 - val_loss: 2.3058
Epoch 41/50
32/32           2s 71ms/step -
accuracy: 0.9551 - loss: 0.1511 - val_accuracy: 0.4290 - val_loss: 2.2601
Epoch 42/50
32/32           2s 72ms/step -
accuracy: 0.9296 - loss: 0.1918 - val_accuracy: 0.4012 - val_loss: 2.2889
Epoch 43/50
32/32           2s 75ms/step -
accuracy: 0.9638 - loss: 0.1422 - val_accuracy: 0.4198 - val_loss: 2.2849
Epoch 44/50
32/32           2s 73ms/step -
accuracy: 0.9598 - loss: 0.1563 - val_accuracy: 0.4105 - val_loss: 2.2788
Epoch 45/50
32/32           2s 72ms/step -
accuracy: 0.9573 - loss: 0.1448 - val_accuracy: 0.4475 - val_loss: 2.3064
Epoch 46/50
32/32           2s 72ms/step -
accuracy: 0.9610 - loss: 0.1311 - val_accuracy: 0.4259 - val_loss: 2.3123
Epoch 47/50
32/32           2s 69ms/step -
accuracy: 0.9569 - loss: 0.1466 - val_accuracy: 0.4383 - val_loss: 2.3183
Epoch 48/50
32/32           2s 69ms/step -
accuracy: 0.9772 - loss: 0.1211 - val_accuracy: 0.4167 - val_loss: 2.4513
```

```
Epoch 49/50
32/32               2s 68ms/step -
accuracy: 0.9519 - loss: 0.1423 - val_accuracy: 0.4259 - val_loss: 2.4228
Epoch 50/50
32/32               2s 70ms/step -
accuracy: 0.9503 - loss: 0.1569 - val_accuracy: 0.4290 - val_loss: 2.3494
```

[4]:

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

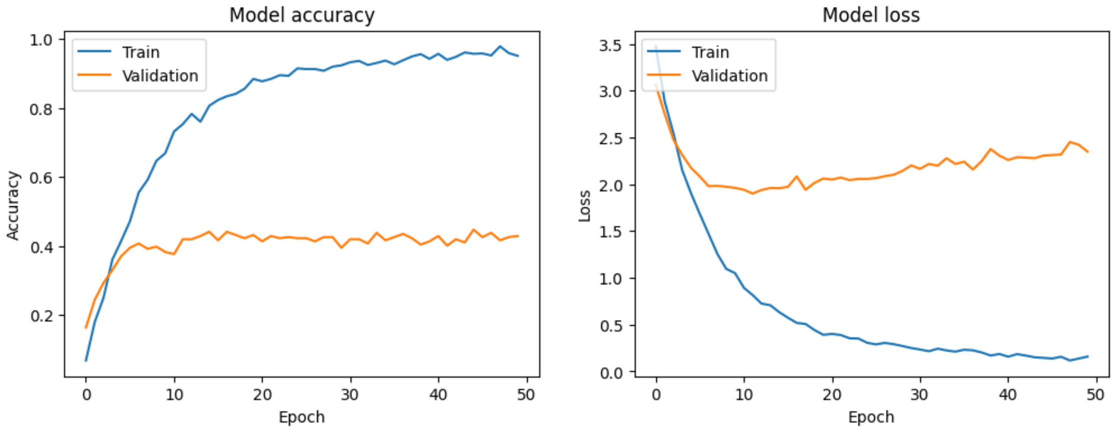
```
11/11               4s 334ms/step -
accuracy: 0.4532 - loss: 2.3851
Test Accuracy: 0.4574
Test Loss: 2.3385
```

[5]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import numpy as np
```

[6]:

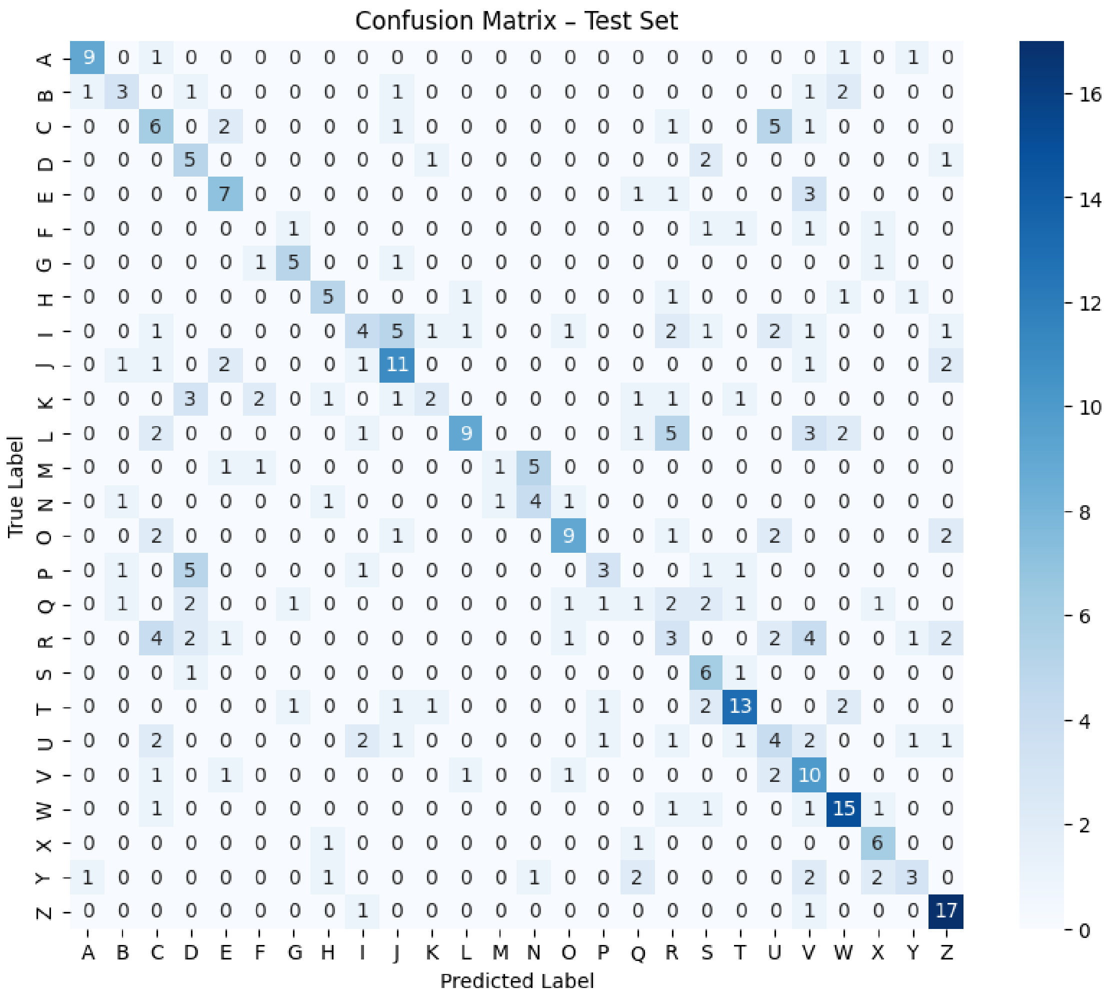
```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
plt.subplot(1, 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      0.818     0.750     0.783        12
           B      0.429     0.333     0.375         9
```

C	0.286	0.375	0.324	16
D	0.263	0.556	0.357	9
E	0.500	0.583	0.538	12
F	0.000	0.000	0.000	5
G	0.625	0.625	0.625	8
H	0.556	0.556	0.556	9
I	0.400	0.200	0.267	20
J	0.478	0.579	0.524	19
K	0.400	0.167	0.235	12
L	0.750	0.391	0.514	23
M	0.500	0.125	0.200	8
N	0.400	0.500	0.444	8
O	0.643	0.529	0.581	17
P	0.500	0.250	0.333	12
Q	0.143	0.077	0.100	13
R	0.158	0.150	0.154	20
S	0.375	0.750	0.500	8
T	0.684	0.619	0.650	21
U	0.235	0.250	0.242	16
V	0.323	0.625	0.426	16
W	0.652	0.750	0.698	20
X	0.500	0.750	0.600	8
Y	0.429	0.250	0.316	12
Z	0.654	0.895	0.756	19
accuracy			0.457	352
macro avg	0.450	0.447	0.427	352
weighted avg	0.471	0.457	0.444	352


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
0 2 4 6 8 10 12 14 16