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
```

```python
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 1722 files belonging to 25 classes.
Using 1034 files for training.
Found 1722 files belonging to 25 classes.
Using 688 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
33/33            9s 120ms/step -
accuracy: 0.1183 - loss: 3.4523 - val_accuracy: 0.5030 - val_loss: 1.9964
Epoch 2/50
33/33            2s 60ms/step -
accuracy: 0.4331 - loss: 1.8864 - val_accuracy: 0.7351 - val_loss: 1.1025
Epoch 3/50
33/33            2s 58ms/step -
accuracy: 0.6330 - loss: 1.2320 - val_accuracy: 0.8065 - val_loss: 0.7908
Epoch 4/50
33/33            2s 58ms/step -
accuracy: 0.7745 - loss: 0.8143 - val_accuracy: 0.8542 - val_loss: 0.6272
Epoch 5/50
33/33            2s 57ms/step -
accuracy: 0.8013 - loss: 0.6647 - val_accuracy: 0.8482 - val_loss: 0.5455
Epoch 6/50
33/33            2s 57ms/step -
accuracy: 0.8354 - loss: 0.5660 - val_accuracy: 0.8810 - val_loss: 0.4308
Epoch 7/50
33/33            2s 57ms/step -
accuracy: 0.8531 - loss: 0.5053 - val_accuracy: 0.8958 - val_loss: 0.3935
Epoch 8/50
33/33            2s 58ms/step -
accuracy: 0.8897 - loss: 0.3778 - val_accuracy: 0.8988 - val_loss: 0.3880
Epoch 9/50
33/33            2s 58ms/step -
accuracy: 0.9136 - loss: 0.2950 - val_accuracy: 0.9077 - val_loss: 0.3186
Epoch 10/50
33/33            2s 57ms/step -
accuracy: 0.9073 - loss: 0.2964 - val_accuracy: 0.9345 - val_loss: 0.2995
Epoch 11/50
33/33            2s 57ms/step -
accuracy: 0.9016 - loss: 0.3026 - val_accuracy: 0.9077 - val_loss: 0.2882
Epoch 12/50
33/33            2s 57ms/step -
accuracy: 0.9309 - loss: 0.2376 - val_accuracy: 0.9256 - val_loss: 0.2701
Epoch 13/50
33/33            2s 57ms/step -
accuracy: 0.9432 - loss: 0.1932 - val_accuracy: 0.9315 - val_loss: 0.2523
Epoch 14/50
33/33            2s 57ms/step -
accuracy: 0.9289 - loss: 0.2178 - val_accuracy: 0.9345 - val_loss: 0.2342
Epoch 15/50
33/33            2s 58ms/step -
accuracy: 0.9477 - loss: 0.1849 - val_accuracy: 0.9286 - val_loss: 0.2551
Epoch 16/50
33/33            2s 57ms/step -
accuracy: 0.9497 - loss: 0.1671 - val_accuracy: 0.9315 - val_loss: 0.2590
```

```
Epoch 17/50
33/33            2s 58ms/step -
accuracy: 0.9595 - loss: 0.1436 - val_accuracy: 0.9375 - val_loss: 0.2479
Epoch 18/50
33/33            2s 57ms/step -
accuracy: 0.9604 - loss: 0.1278 - val_accuracy: 0.9405 - val_loss: 0.2442
Epoch 19/50
33/33            2s 57ms/step -
accuracy: 0.9584 - loss: 0.1361 - val_accuracy: 0.9435 - val_loss: 0.2160
Epoch 20/50
33/33            2s 58ms/step -
accuracy: 0.9703 - loss: 0.1049 - val_accuracy: 0.9375 - val_loss: 0.2093
Epoch 21/50
33/33            2s 57ms/step -
accuracy: 0.9712 - loss: 0.0950 - val_accuracy: 0.9494 - val_loss: 0.2177
Epoch 22/50
33/33            2s 57ms/step -
accuracy: 0.9823 - loss: 0.0710 - val_accuracy: 0.9464 - val_loss: 0.2012
Epoch 23/50
33/33            2s 58ms/step -
accuracy: 0.9891 - loss: 0.0625 - val_accuracy: 0.9464 - val_loss: 0.2278
Epoch 24/50
33/33            2s 57ms/step -
accuracy: 0.9858 - loss: 0.0709 - val_accuracy: 0.9554 - val_loss: 0.2116
Epoch 25/50
33/33            2s 57ms/step -
accuracy: 0.9717 - loss: 0.0920 - val_accuracy: 0.9494 - val_loss: 0.2137
Epoch 26/50
33/33            2s 57ms/step -
accuracy: 0.9850 - loss: 0.0601 - val_accuracy: 0.9464 - val_loss: 0.2231
Epoch 27/50
33/33            2s 57ms/step -
accuracy: 0.9859 - loss: 0.0671 - val_accuracy: 0.9524 - val_loss: 0.2212
Epoch 28/50
33/33            2s 57ms/step -
accuracy: 0.9715 - loss: 0.0796 - val_accuracy: 0.9494 - val_loss: 0.1972
Epoch 29/50
33/33            2s 57ms/step -
accuracy: 0.9775 - loss: 0.0761 - val_accuracy: 0.9464 - val_loss: 0.2183
Epoch 30/50
33/33            2s 57ms/step -
accuracy: 0.9776 - loss: 0.0717 - val_accuracy: 0.9494 - val_loss: 0.2055
Epoch 31/50
33/33            2s 58ms/step -
accuracy: 0.9883 - loss: 0.0615 - val_accuracy: 0.9583 - val_loss: 0.1956
Epoch 32/50
33/33            2s 57ms/step -
accuracy: 0.9803 - loss: 0.0625 - val_accuracy: 0.9375 - val_loss: 0.2155
```

```
Epoch 33/50
33/33           2s 58ms/step -
accuracy: 0.9762 - loss: 0.0773 - val_accuracy: 0.9435 - val_loss: 0.2170
Epoch 34/50
33/33           2s 58ms/step -
accuracy: 0.9891 - loss: 0.0378 - val_accuracy: 0.9554 - val_loss: 0.2109
Epoch 35/50
33/33           2s 58ms/step -
accuracy: 0.9964 - loss: 0.0424 - val_accuracy: 0.9375 - val_loss: 0.2225
Epoch 36/50
33/33           2s 57ms/step -
accuracy: 0.9771 - loss: 0.0602 - val_accuracy: 0.9494 - val_loss: 0.1995
Epoch 37/50
33/33           2s 57ms/step -
accuracy: 0.9789 - loss: 0.0585 - val_accuracy: 0.9583 - val_loss: 0.2124
Epoch 38/50
33/33           2s 58ms/step -
accuracy: 0.9784 - loss: 0.0666 - val_accuracy: 0.9435 - val_loss: 0.2283
Epoch 39/50
33/33           2s 57ms/step -
accuracy: 0.9971 - loss: 0.0298 - val_accuracy: 0.9554 - val_loss: 0.2114
Epoch 40/50
33/33           2s 62ms/step -
accuracy: 0.9794 - loss: 0.0561 - val_accuracy: 0.9524 - val_loss: 0.2150
Epoch 41/50
33/33           2s 59ms/step -
accuracy: 0.9921 - loss: 0.0344 - val_accuracy: 0.9554 - val_loss: 0.2165
Epoch 42/50
33/33           2s 61ms/step -
accuracy: 0.9945 - loss: 0.0277 - val_accuracy: 0.9494 - val_loss: 0.2186
Epoch 43/50
33/33           2s 59ms/step -
accuracy: 0.9828 - loss: 0.0478 - val_accuracy: 0.9494 - val_loss: 0.2196
Epoch 44/50
33/33           2s 60ms/step -
accuracy: 0.9871 - loss: 0.0463 - val_accuracy: 0.9435 - val_loss: 0.2366
Epoch 45/50
33/33           2s 59ms/step -
accuracy: 0.9891 - loss: 0.0384 - val_accuracy: 0.9494 - val_loss: 0.2233
Epoch 46/50
33/33           2s 61ms/step -
accuracy: 0.9922 - loss: 0.0300 - val_accuracy: 0.9554 - val_loss: 0.2446
Epoch 47/50
33/33           2s 60ms/step -
accuracy: 0.9822 - loss: 0.0556 - val_accuracy: 0.9524 - val_loss: 0.2354
Epoch 48/50
33/33           2s 61ms/step -
accuracy: 0.9876 - loss: 0.0357 - val_accuracy: 0.9464 - val_loss: 0.2289
```

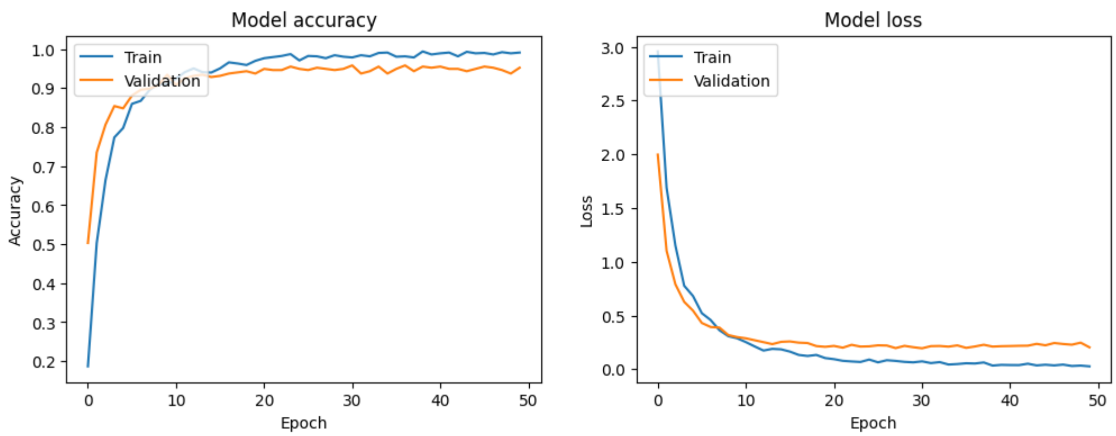
```
Epoch 49/50
33/33            2s 60ms/step -
accuracy: 0.9943 - loss: 0.0296 - val_accuracy: 0.9375 - val_loss: 0.2478
Epoch 50/50
33/33            2s 60ms/step -
accuracy: 0.9911 - loss: 0.0266 - val_accuracy: 0.9524 - val_loss: 0.2045
```

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

```
11/11            1s 53ms/step -
accuracy: 0.9274 - loss: 0.2769
Test Accuracy: 0.9403
Test Loss: 0.2418
```

[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()
```

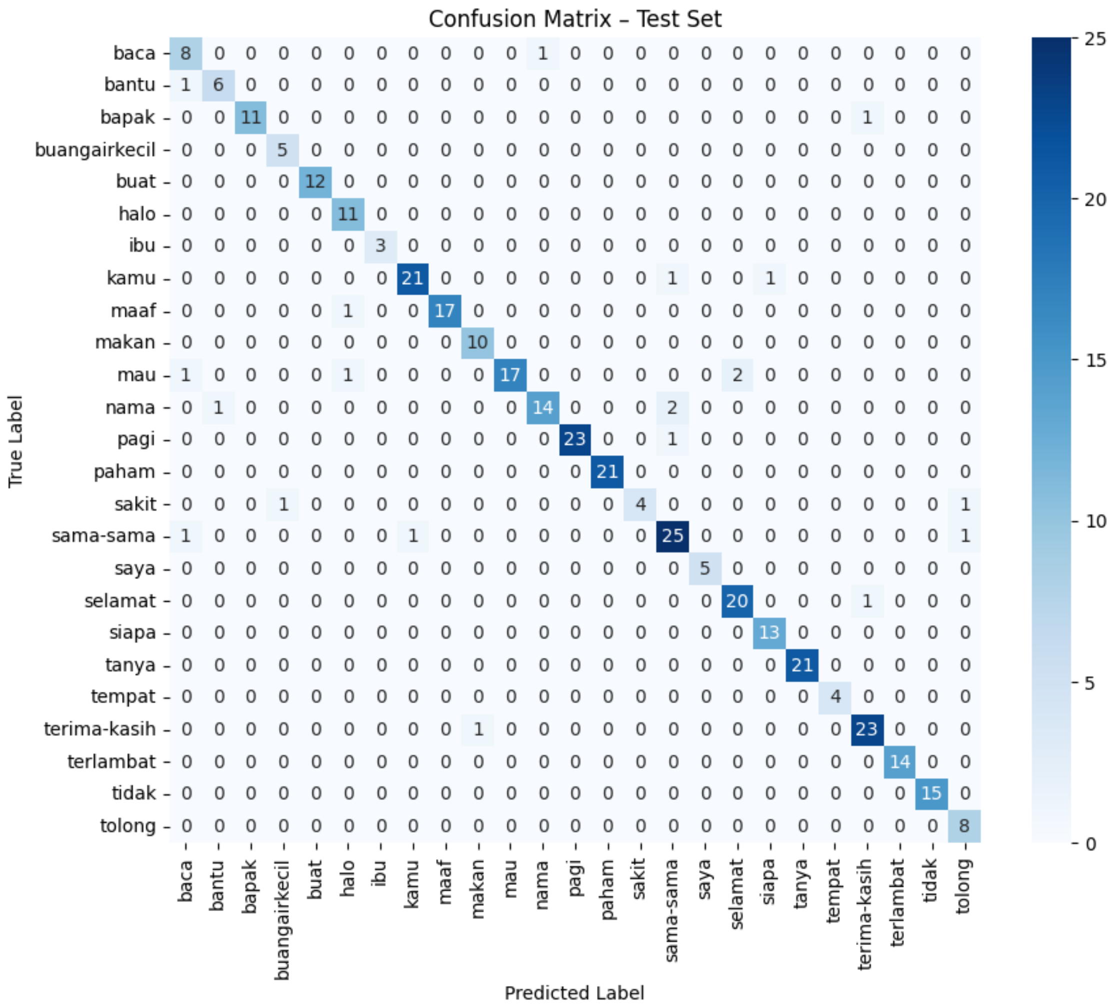


[7]:

```
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

        baca      0.727     0.889     0.800         9
       bantu      0.857     0.857     0.857         7
```

bapak	1.000	0.917	0.957	12
buangairkecil	0.833	1.000	0.909	5
buat	1.000	1.000	1.000	12
halo	0.846	1.000	0.917	11
ibu	1.000	1.000	1.000	3
kamu	0.955	0.913	0.933	23
maaf	1.000	0.944	0.971	18
makan	0.909	1.000	0.952	10
mau	1.000	0.810	0.895	21
nama	0.933	0.824	0.875	17
pagi	1.000	0.958	0.979	24
paham	1.000	1.000	1.000	21
sakit	1.000	0.667	0.800	6
sama-sama	0.862	0.893	0.877	28
saya	1.000	1.000	1.000	5
selamat	0.909	0.952	0.930	21
siapa	0.929	1.000	0.963	13
tanya	1.000	1.000	1.000	21
tempat	1.000	1.000	1.000	4
terima-kasih	0.920	0.958	0.939	24
terlambat	1.000	1.000	1.000	14
tidak	1.000	1.000	1.000	15
tolong	0.800	1.000	0.889	8
accuracy			0.940	352
macro avg	0.939	0.943	0.938	352
weighted avg	0.945	0.940	0.940	352


Confusion Matrix – Test Set
True Label
Predicted Label