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 2155 files belonging to 25 classes.
Using 1293 files for training.
Found 2155 files belonging to 25 classes.
Using 862 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
41/41            11s 113ms/step -
accuracy: 0.1360 - loss: 3.1582 - val_accuracy: 0.5697 - val_loss: 1.6207
Epoch 2/50
41/41            2s 59ms/step -
accuracy: 0.4942 - loss: 1.6853 - val_accuracy: 0.8077 - val_loss: 0.8321
Epoch 3/50
41/41            2s 61ms/step -
accuracy: 0.6972 - loss: 1.0776 - val_accuracy: 0.8317 - val_loss: 0.5994
Epoch 4/50
41/41            2s 59ms/step -
accuracy: 0.7903 - loss: 0.7457 - val_accuracy: 0.8702 - val_loss: 0.4732
Epoch 5/50
41/41            2s 60ms/step -
accuracy: 0.8484 - loss: 0.5683 - val_accuracy: 0.8846 - val_loss: 0.3759
Epoch 6/50
41/41            3s 64ms/step -
accuracy: 0.8688 - loss: 0.4592 - val_accuracy: 0.9014 - val_loss: 0.3311
Epoch 7/50
41/41            2s 59ms/step -
accuracy: 0.8774 - loss: 0.3873 - val_accuracy: 0.9062 - val_loss: 0.2794
Epoch 8/50
41/41            2s 61ms/step -
accuracy: 0.9092 - loss: 0.2968 - val_accuracy: 0.9159 - val_loss: 0.2626
Epoch 9/50
41/41            3s 61ms/step -
accuracy: 0.9273 - loss: 0.2643 - val_accuracy: 0.9207 - val_loss: 0.2447
Epoch 10/50
41/41            3s 62ms/step -
accuracy: 0.9333 - loss: 0.2328 - val_accuracy: 0.9351 - val_loss: 0.2349
Epoch 11/50
41/41            3s 70ms/step -
accuracy: 0.9308 - loss: 0.2127 - val_accuracy: 0.9303 - val_loss: 0.2185
Epoch 12/50
41/41            3s 84ms/step -
accuracy: 0.9389 - loss: 0.2111 - val_accuracy: 0.9399 - val_loss: 0.1955
Epoch 13/50
41/41            3s 73ms/step -
accuracy: 0.9559 - loss: 0.1666 - val_accuracy: 0.9447 - val_loss: 0.2057
Epoch 14/50
41/41            3s 72ms/step -
accuracy: 0.9576 - loss: 0.1443 - val_accuracy: 0.9471 - val_loss: 0.2013
Epoch 15/50
41/41            3s 70ms/step -
accuracy: 0.9472 - loss: 0.1496 - val_accuracy: 0.9471 - val_loss: 0.1771
Epoch 16/50
41/41            3s 71ms/step -
accuracy: 0.9614 - loss: 0.1413 - val_accuracy: 0.9303 - val_loss: 0.1828
```

```
Epoch 17/50
41/41            3s 65ms/step -
accuracy: 0.9641 - loss: 0.1267 - val_accuracy: 0.9351 - val_loss: 0.1936
Epoch 18/50
41/41            3s 65ms/step -
accuracy: 0.9547 - loss: 0.1444 - val_accuracy: 0.9519 - val_loss: 0.1646
Epoch 19/50
41/41            3s 64ms/step -
accuracy: 0.9653 - loss: 0.1189 - val_accuracy: 0.9543 - val_loss: 0.1730
Epoch 20/50
41/41            3s 66ms/step -
accuracy: 0.9780 - loss: 0.1047 - val_accuracy: 0.9471 - val_loss: 0.1678
Epoch 21/50
41/41            3s 64ms/step -
accuracy: 0.9789 - loss: 0.0950 - val_accuracy: 0.9495 - val_loss: 0.1606
Epoch 22/50
41/41            3s 63ms/step -
accuracy: 0.9689 - loss: 0.0963 - val_accuracy: 0.9471 - val_loss: 0.1524
Epoch 23/50
41/41            3s 69ms/step -
accuracy: 0.9818 - loss: 0.0781 - val_accuracy: 0.9447 - val_loss: 0.1546
Epoch 24/50
41/41            3s 64ms/step -
accuracy: 0.9836 - loss: 0.0733 - val_accuracy: 0.9567 - val_loss: 0.1505
Epoch 25/50
41/41            3s 64ms/step -
accuracy: 0.9764 - loss: 0.0744 - val_accuracy: 0.9567 - val_loss: 0.1427
Epoch 26/50
41/41            3s 66ms/step -
accuracy: 0.9817 - loss: 0.0737 - val_accuracy: 0.9639 - val_loss: 0.1470
Epoch 27/50
41/41            3s 65ms/step -
accuracy: 0.9846 - loss: 0.0717 - val_accuracy: 0.9423 - val_loss: 0.1641
Epoch 28/50
41/41            3s 63ms/step -
accuracy: 0.9849 - loss: 0.0611 - val_accuracy: 0.9543 - val_loss: 0.1484
Epoch 29/50
41/41            3s 64ms/step -
accuracy: 0.9782 - loss: 0.0675 - val_accuracy: 0.9471 - val_loss: 0.1583
Epoch 30/50
41/41            3s 64ms/step -
accuracy: 0.9829 - loss: 0.0579 - val_accuracy: 0.9591 - val_loss: 0.1424
Epoch 31/50
41/41            3s 63ms/step -
accuracy: 0.9807 - loss: 0.0713 - val_accuracy: 0.9519 - val_loss: 0.1420
Epoch 32/50
41/41            3s 64ms/step -
accuracy: 0.9852 - loss: 0.0627 - val_accuracy: 0.9543 - val_loss: 0.1588
```

```
Epoch 33/50
41/41            3s 63ms/step -
accuracy: 0.9798 - loss: 0.0651 - val_accuracy: 0.9543 - val_loss: 0.1476
Epoch 34/50
41/41            3s 63ms/step -
accuracy: 0.9885 - loss: 0.0465 - val_accuracy: 0.9591 - val_loss: 0.1405
Epoch 35/50
41/41            3s 65ms/step -
accuracy: 0.9857 - loss: 0.0576 - val_accuracy: 0.9567 - val_loss: 0.1379
Epoch 36/50
41/41            3s 64ms/step -
accuracy: 0.9714 - loss: 0.0698 - val_accuracy: 0.9567 - val_loss: 0.1500
Epoch 37/50
41/41            3s 63ms/step -
accuracy: 0.9883 - loss: 0.0468 - val_accuracy: 0.9471 - val_loss: 0.1497
Epoch 38/50
41/41            3s 64ms/step -
accuracy: 0.9887 - loss: 0.0658 - val_accuracy: 0.9447 - val_loss: 0.1734
Epoch 39/50
41/41            3s 63ms/step -
accuracy: 0.9895 - loss: 0.0450 - val_accuracy: 0.9615 - val_loss: 0.1277
Epoch 40/50
41/41            3s 63ms/step -
accuracy: 0.9929 - loss: 0.0412 - val_accuracy: 0.9543 - val_loss: 0.1516
Epoch 41/50
41/41            3s 66ms/step -
accuracy: 0.9814 - loss: 0.0594 - val_accuracy: 0.9567 - val_loss: 0.1565
Epoch 42/50
41/41            3s 65ms/step -
accuracy: 0.9858 - loss: 0.0498 - val_accuracy: 0.9495 - val_loss: 0.1477
Epoch 43/50
41/41            3s 63ms/step -
accuracy: 0.9871 - loss: 0.0386 - val_accuracy: 0.9615 - val_loss: 0.1177
Epoch 44/50
41/41            3s 65ms/step -
accuracy: 0.9841 - loss: 0.0386 - val_accuracy: 0.9567 - val_loss: 0.1377
Epoch 45/50
41/41            3s 63ms/step -
accuracy: 0.9957 - loss: 0.0307 - val_accuracy: 0.9591 - val_loss: 0.1215
Epoch 46/50
41/41            3s 63ms/step -
accuracy: 0.9871 - loss: 0.0391 - val_accuracy: 0.9639 - val_loss: 0.1501
Epoch 47/50
41/41            3s 64ms/step -
accuracy: 0.9855 - loss: 0.0476 - val_accuracy: 0.9591 - val_loss: 0.1380
Epoch 48/50
41/41            3s 63ms/step -
accuracy: 0.9846 - loss: 0.0477 - val_accuracy: 0.9495 - val_loss: 0.1449
```

```
Epoch 49/50
41/41              3s 64ms/step -
accuracy: 0.9928 - loss: 0.0351 - val_accuracy: 0.9471 - val_loss: 0.1871
Epoch 50/50
41/41              3s 63ms/step -
accuracy: 0.9709 - loss: 0.0681 - val_accuracy: 0.9567 - val_loss: 0.1567
```

[4]:

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

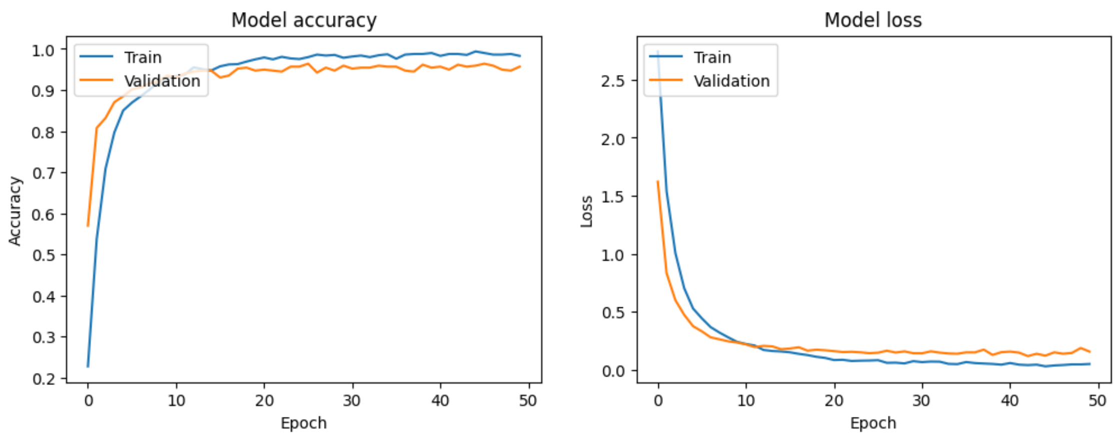
```
14/14              1s 55ms/step -
accuracy: 0.9590 - loss: 0.1121
Test Accuracy: 0.9641
Test Loss: 0.1134
```

[5]:

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

[6]:

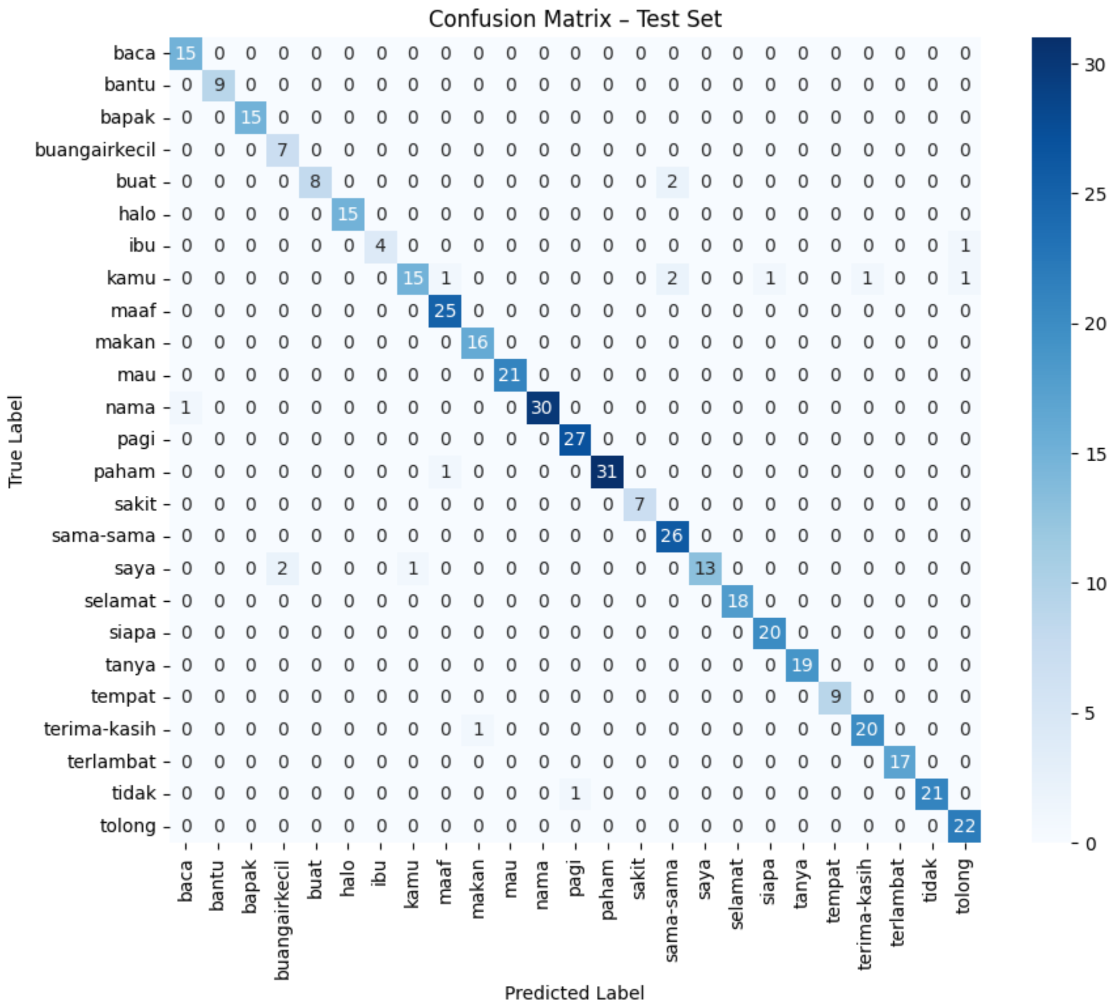
```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
plt.subplot(1,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

        baca      0.938     1.000     0.968        15
       bantu      1.000     1.000     1.000         9
```

bapak	1.000	1.000	1.000	15
buangairkecil	0.778	1.000	0.875	7
buat	1.000	0.800	0.889	10
halo	1.000	1.000	1.000	15
ibu	1.000	0.800	0.889	5
kamu	0.938	0.714	0.811	21
maaf	0.926	1.000	0.962	25
makan	0.941	1.000	0.970	16
mau	1.000	1.000	1.000	21
nama	1.000	0.968	0.984	31
pagi	0.964	1.000	0.982	27
paham	1.000	0.969	0.984	32
sakit	1.000	1.000	1.000	7
sama-sama	0.867	1.000	0.929	26
saya	1.000	0.812	0.897	16
selamat	1.000	1.000	1.000	18
siapa	0.952	1.000	0.976	20
tanya	1.000	1.000	1.000	19
tempat	1.000	1.000	1.000	9
terima-kasih	0.952	0.952	0.952	21
terlambat	1.000	1.000	1.000	17
tidak	1.000	0.955	0.977	22
tolong	0.917	1.000	0.957	22
accuracy			0.964	446
macro avg	0.967	0.959	0.960	446
weighted avg	0.967	0.964	0.963	446


Confusion Matrix – Test Set
True Label
Predicted Label
baca
bantu
bapak
buangairkecil
buat
halo
ibu
kamu
maaf
makan
mau
nama
pagi
paham
sakit
sama-sama
saya
selamat
siapa
tanya
tempat
terima-kasih
terlambat
tidak
tolong