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
```

```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 5643 files belonging to 51 classes.
Using 3386 files for training.
Found 5643 files belonging to 51 classes.
Using 2257 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
106/106             31s 145ms/step -
accuracy: 0.0821 - loss: 3.8709 - val_accuracy: 0.3357 - val_loss: 2.4915
Epoch 2/50
106/106             7s 62ms/step -
accuracy: 0.2959 - loss: 2.4460 - val_accuracy: 0.5071 - val_loss: 1.8254
Epoch 3/50
106/106             7s 63ms/step -
accuracy: 0.4433 - loss: 1.9017 - val_accuracy: 0.5750 - val_loss: 1.5179
Epoch 4/50
106/106             8s 74ms/step -
accuracy: 0.5172 - loss: 1.5514 - val_accuracy: 0.6045 - val_loss: 1.3667
Epoch 5/50
106/106             8s 75ms/step -
accuracy: 0.5785 - loss: 1.3306 - val_accuracy: 0.6205 - val_loss: 1.2773
Epoch 6/50
106/106             7s 70ms/step -
accuracy: 0.6324 - loss: 1.1819 - val_accuracy: 0.6143 - val_loss: 1.2361
Epoch 7/50
106/106             7s 69ms/step -
accuracy: 0.6704 - loss: 1.0707 - val_accuracy: 0.6313 - val_loss: 1.1976
Epoch 8/50
106/106             7s 69ms/step -
accuracy: 0.7010 - loss: 0.9367 - val_accuracy: 0.6313 - val_loss: 1.2032
Epoch 9/50
106/106             7s 69ms/step -
accuracy: 0.7164 - loss: 0.8524 - val_accuracy: 0.6375 - val_loss: 1.1630
Epoch 10/50
106/106             7s 68ms/step -
accuracy: 0.7279 - loss: 0.8470 - val_accuracy: 0.6304 - val_loss: 1.1502
Epoch 11/50
106/106             7s 68ms/step -
accuracy: 0.7580 - loss: 0.7494 - val_accuracy: 0.6491 - val_loss: 1.1470
Epoch 12/50
106/106             7s 68ms/step -
accuracy: 0.7682 - loss: 0.6955 - val_accuracy: 0.6429 - val_loss: 1.1309
Epoch 13/50
106/106             7s 70ms/step -
accuracy: 0.7842 - loss: 0.6664 - val_accuracy: 0.6464 - val_loss: 1.1419
Epoch 14/50
106/106             7s 68ms/step -
accuracy: 0.8032 - loss: 0.6217 - val_accuracy: 0.6455 - val_loss: 1.1317
Epoch 15/50
106/106             7s 67ms/step -
accuracy: 0.8145 - loss: 0.5845 - val_accuracy: 0.6321 - val_loss: 1.1303
Epoch 16/50
106/106             7s 69ms/step -
accuracy: 0.8300 - loss: 0.5458 - val_accuracy: 0.6420 - val_loss: 1.1488
```

```
Epoch 17/50
106/106            7s 69ms/step -
accuracy: 0.8207 - loss: 0.5450 - val_accuracy: 0.6357 - val_loss: 1.1385
Epoch 18/50
106/106            7s 68ms/step -
accuracy: 0.8334 - loss: 0.5418 - val_accuracy: 0.6339 - val_loss: 1.1635
Epoch 19/50
106/106            7s 68ms/step -
accuracy: 0.8348 - loss: 0.4920 - val_accuracy: 0.6429 - val_loss: 1.1648
Epoch 20/50
106/106            7s 68ms/step -
accuracy: 0.8400 - loss: 0.4762 - val_accuracy: 0.6411 - val_loss: 1.1972
Epoch 21/50
106/106            7s 68ms/step -
accuracy: 0.8398 - loss: 0.4677 - val_accuracy: 0.6455 - val_loss: 1.1394
Epoch 22/50
106/106            7s 68ms/step -
accuracy: 0.8455 - loss: 0.4510 - val_accuracy: 0.6491 - val_loss: 1.1743
Epoch 23/50
106/106            7s 69ms/step -
accuracy: 0.8602 - loss: 0.4289 - val_accuracy: 0.6509 - val_loss: 1.2067
Epoch 24/50
106/106            7s 70ms/step -
accuracy: 0.8722 - loss: 0.3744 - val_accuracy: 0.6446 - val_loss: 1.1939
Epoch 25/50
106/106            7s 70ms/step -
accuracy: 0.8728 - loss: 0.3862 - val_accuracy: 0.6509 - val_loss: 1.2045
Epoch 26/50
106/106            7s 69ms/step -
accuracy: 0.8809 - loss: 0.3705 - val_accuracy: 0.6580 - val_loss: 1.2087
Epoch 27/50
106/106            7s 69ms/step -
accuracy: 0.8793 - loss: 0.3573 - val_accuracy: 0.6491 - val_loss: 1.2118
Epoch 28/50
106/106            7s 69ms/step -
accuracy: 0.8769 - loss: 0.3834 - val_accuracy: 0.6429 - val_loss: 1.2569
Epoch 29/50
106/106            7s 69ms/step -
accuracy: 0.8884 - loss: 0.3356 - val_accuracy: 0.6402 - val_loss: 1.2485
Epoch 30/50
106/106            7s 69ms/step -
accuracy: 0.8828 - loss: 0.3550 - val_accuracy: 0.6554 - val_loss: 1.2400
Epoch 31/50
106/106            7s 69ms/step -
accuracy: 0.9039 - loss: 0.2936 - val_accuracy: 0.6268 - val_loss: 1.2811
Epoch 32/50
106/106            7s 69ms/step -
accuracy: 0.8928 - loss: 0.3340 - val_accuracy: 0.6384 - val_loss: 1.2495
```

```
Epoch 33/50
106/106            7s 69ms/step -
accuracy: 0.9081 - loss: 0.2934 - val_accuracy: 0.6384 - val_loss: 1.2868
Epoch 34/50
106/106            7s 69ms/step -
accuracy: 0.8912 - loss: 0.3095 - val_accuracy: 0.6357 - val_loss: 1.2883
Epoch 35/50
106/106            7s 69ms/step -
accuracy: 0.8916 - loss: 0.3126 - val_accuracy: 0.6313 - val_loss: 1.3358
Epoch 36/50
106/106            7s 69ms/step -
accuracy: 0.8985 - loss: 0.3034 - val_accuracy: 0.6348 - val_loss: 1.3427
Epoch 37/50
106/106            7s 69ms/step -
accuracy: 0.9015 - loss: 0.2791 - val_accuracy: 0.6339 - val_loss: 1.3538
Epoch 38/50
106/106            7s 71ms/step -
accuracy: 0.8889 - loss: 0.3075 - val_accuracy: 0.6384 - val_loss: 1.3442
Epoch 39/50
106/106            7s 69ms/step -
accuracy: 0.9178 - loss: 0.2520 - val_accuracy: 0.6321 - val_loss: 1.4135
Epoch 40/50
106/106            7s 69ms/step -
accuracy: 0.9062 - loss: 0.2875 - val_accuracy: 0.6393 - val_loss: 1.3826
Epoch 41/50
106/106            7s 69ms/step -
accuracy: 0.9207 - loss: 0.2517 - val_accuracy: 0.6286 - val_loss: 1.4341
Epoch 42/50
106/106            7s 70ms/step -
accuracy: 0.9042 - loss: 0.2847 - val_accuracy: 0.6411 - val_loss: 1.3822
Epoch 43/50
106/106            8s 76ms/step -
accuracy: 0.9171 - loss: 0.2554 - val_accuracy: 0.6259 - val_loss: 1.4259
Epoch 44/50
106/106            8s 73ms/step -
accuracy: 0.9032 - loss: 0.2738 - val_accuracy: 0.6518 - val_loss: 1.3415
Epoch 45/50
106/106            8s 74ms/step -
accuracy: 0.9165 - loss: 0.2473 - val_accuracy: 0.6509 - val_loss: 1.3767
Epoch 46/50
106/106            8s 76ms/step -
accuracy: 0.9134 - loss: 0.2555 - val_accuracy: 0.6411 - val_loss: 1.4024
Epoch 47/50
106/106            8s 74ms/step -
accuracy: 0.9362 - loss: 0.2144 - val_accuracy: 0.6375 - val_loss: 1.4269
Epoch 48/50
106/106            8s 75ms/step -
accuracy: 0.9284 - loss: 0.2237 - val_accuracy: 0.6339 - val_loss: 1.4466
```

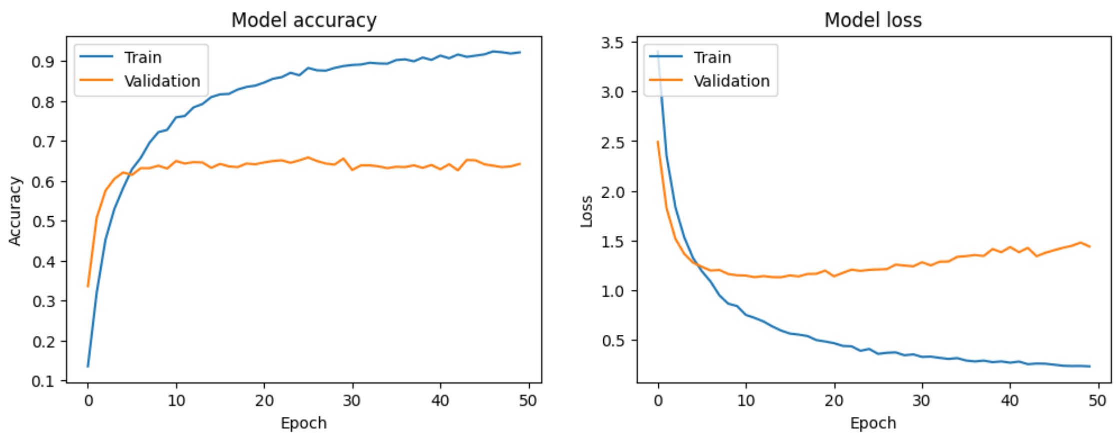
```
Epoch 49/50
106/106              8s 74ms/step -
accuracy: 0.9265 - loss: 0.2347 - val_accuracy: 0.6357 - val_loss: 1.4796
Epoch 50/50
106/106              8s 73ms/step -
accuracy: 0.9199 - loss: 0.2294 - val_accuracy: 0.6420 - val_loss: 1.4395
```

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

```
36/36              8s 226ms/step -
accuracy: 0.6434 - loss: 1.4337
Test Accuracy: 0.6376
Test Loss: 1.4552
```

[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      0.667     0.632     0.649        19
           B      0.389     0.259     0.311        27
```

C	0.275	0.609	0.378	23
D	0.257	0.429	0.321	21
E	0.643	0.600	0.621	30
F	0.381	0.320	0.348	25
G	0.585	0.774	0.667	31
H	0.654	0.586	0.618	29
I	0.286	0.273	0.279	22
J	0.400	0.500	0.444	24
K	0.611	0.344	0.440	32
L	0.571	0.615	0.593	26
M	0.355	0.256	0.297	43
N	0.179	0.250	0.209	28
O	0.476	0.385	0.426	26
P	0.333	0.208	0.256	24
Q	0.348	0.444	0.390	18
R	0.471	0.381	0.421	21
S	0.667	0.500	0.571	32
T	0.379	0.500	0.431	22
U	0.286	0.182	0.222	33
V	0.452	0.633	0.528	30
W	0.476	0.385	0.426	26
X	0.500	0.478	0.489	23
Y	0.478	0.379	0.423	29
Z	0.828	0.857	0.842	28
baca	1.000	0.818	0.900	11
bantu	0.933	0.933	0.933	15
bapak	0.950	1.000	0.974	19
buangairkecil	1.000	0.727	0.842	11
buat	0.800	1.000	0.889	12
halo	1.000	1.000	1.000	15
ibu	0.857	1.000	0.923	6
kamu	1.000	0.781	0.877	32
maaf	1.000	1.000	1.000	22
makan	1.000	0.900	0.947	20
mau	1.000	1.000	1.000	23
nama	0.952	0.870	0.909	23
pagi	0.778	0.955	0.857	22
paham	1.000	1.000	1.000	32
sakit	1.000	1.000	1.000	7
sama-sama	0.950	0.950	0.950	20
saya	0.909	0.909	0.909	11
selamat	0.957	0.917	0.936	24
siapa	0.769	0.909	0.833	22
tanya	0.920	0.958	0.939	24
tempat	1.000	0.900	0.947	10
terima-kasih	0.810	1.000	0.895	17
terlambat	0.944	1.000	0.971	17
tidak	0.900	0.900	0.900	10

tolong	0.944	0.850	0.895	20
accuracy			0.638	1137
macro avg	0.693	0.687	0.683	1137
weighted avg	0.650	0.638	0.636	1137