COMPARISON_MobileNetV2

June 21, 2025

[ ]:
```
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 3413 files belonging to 51 classes.
Using 2048 files for training.
Found 3413 files belonging to 51 classes.
Using 1365 files for validation.
```

[5]:
```
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'])
```

[6]:
```
history = model.fit(train_ds, validation_data=val_ds, epochs=50)
```

```
Epoch 1/50
64/64            18s 132ms/step -
accuracy: 0.0737 - loss: 4.0134 - val_accuracy: 0.3348 - val_loss: 2.6370
Epoch 2/50
64/64            4s 61ms/step -
accuracy: 0.2878 - loss: 2.6495 - val_accuracy: 0.4866 - val_loss: 1.9093
Epoch 3/50
64/64            4s 60ms/step -
accuracy: 0.4534 - loss: 1.9079 - val_accuracy: 0.5417 - val_loss: 1.5783
Epoch 4/50
64/64            4s 60ms/step -
accuracy: 0.5672 - loss: 1.4972 - val_accuracy: 0.6071 - val_loss: 1.3757
Epoch 5/50
64/64            4s 60ms/step -
accuracy: 0.6182 - loss: 1.2579 - val_accuracy: 0.6205 - val_loss: 1.2890
Epoch 6/50
64/64            4s 60ms/step -
accuracy: 0.6848 - loss: 1.0517 - val_accuracy: 0.6205 - val_loss: 1.2737
Epoch 7/50
64/64            4s 60ms/step -
accuracy: 0.7213 - loss: 0.9044 - val_accuracy: 0.6235 - val_loss: 1.2364
Epoch 8/50
64/64            4s 60ms/step -
accuracy: 0.7767 - loss: 0.7365 - val_accuracy: 0.6295 - val_loss: 1.2137
Epoch 9/50
64/64            4s 60ms/step -
accuracy: 0.7840 - loss: 0.7048 - val_accuracy: 0.6533 - val_loss: 1.1466
Epoch 10/50
64/64            4s 61ms/step -
accuracy: 0.8021 - loss: 0.6284 - val_accuracy: 0.6562 - val_loss: 1.1585
Epoch 11/50
64/64            4s 60ms/step -
accuracy: 0.8454 - loss: 0.5215 - val_accuracy: 0.6592 - val_loss: 1.1608
Epoch 12/50
64/64            4s 60ms/step -
accuracy: 0.8459 - loss: 0.5075 - val_accuracy: 0.6488 - val_loss: 1.1600
Epoch 13/50
64/64            4s 60ms/step -
accuracy: 0.8566 - loss: 0.4898 - val_accuracy: 0.6622 - val_loss: 1.1625
Epoch 14/50
64/64            4s 60ms/step -
accuracy: 0.8676 - loss: 0.4579 - val_accuracy: 0.6399 - val_loss: 1.1999
Epoch 15/50
64/64            4s 61ms/step -
accuracy: 0.8525 - loss: 0.4536 - val_accuracy: 0.6726 - val_loss: 1.1428
Epoch 16/50
64/64            4s 64ms/step -
accuracy: 0.8711 - loss: 0.3842 - val_accuracy: 0.6607 - val_loss: 1.1278
```

```
Epoch 17/50
64/64              4s 64ms/step -
accuracy: 0.8895 - loss: 0.3503 - val_accuracy: 0.6592 - val_loss: 1.1862
Epoch 18/50
64/64              4s 63ms/step -
accuracy: 0.9022 - loss: 0.3197 - val_accuracy: 0.6637 - val_loss: 1.1861
Epoch 19/50
64/64              4s 63ms/step -
accuracy: 0.9059 - loss: 0.3081 - val_accuracy: 0.6682 - val_loss: 1.1808
Epoch 20/50
64/64              4s 63ms/step -
accuracy: 0.9200 - loss: 0.2911 - val_accuracy: 0.6622 - val_loss: 1.2058
Epoch 21/50
64/64              4s 64ms/step -
accuracy: 0.9169 - loss: 0.2797 - val_accuracy: 0.6652 - val_loss: 1.2206
Epoch 22/50
64/64              4s 63ms/step -
accuracy: 0.9109 - loss: 0.2708 - val_accuracy: 0.6577 - val_loss: 1.2065
Epoch 23/50
64/64              4s 63ms/step -
accuracy: 0.9284 - loss: 0.2414 - val_accuracy: 0.6696 - val_loss: 1.1733
Epoch 24/50
64/64              4s 64ms/step -
accuracy: 0.9346 - loss: 0.2258 - val_accuracy: 0.6771 - val_loss: 1.2226
Epoch 25/50
64/64              4s 63ms/step -
accuracy: 0.9218 - loss: 0.2604 - val_accuracy: 0.6622 - val_loss: 1.2074
Epoch 26/50
64/64              4s 63ms/step -
accuracy: 0.9451 - loss: 0.2080 - val_accuracy: 0.6652 - val_loss: 1.2440
Epoch 27/50
64/64              4s 63ms/step -
accuracy: 0.9360 - loss: 0.2120 - val_accuracy: 0.6771 - val_loss: 1.1871
Epoch 28/50
64/64              4s 62ms/step -
accuracy: 0.9481 - loss: 0.1854 - val_accuracy: 0.6592 - val_loss: 1.2634
Epoch 29/50
64/64              4s 62ms/step -
accuracy: 0.9457 - loss: 0.1882 - val_accuracy: 0.6667 - val_loss: 1.2261
Epoch 30/50
64/64              4s 64ms/step -
accuracy: 0.9424 - loss: 0.1867 - val_accuracy: 0.6845 - val_loss: 1.2612
Epoch 31/50
64/64              4s 62ms/step -
accuracy: 0.9516 - loss: 0.1662 - val_accuracy: 0.6577 - val_loss: 1.3016
Epoch 32/50
64/64              4s 62ms/step -
accuracy: 0.9450 - loss: 0.1642 - val_accuracy: 0.6667 - val_loss: 1.2758
```

```
Epoch 33/50
64/64            4s 62ms/step -
accuracy: 0.9435 - loss: 0.1751 - val_accuracy: 0.6667 - val_loss: 1.2586
Epoch 34/50
64/64            4s 63ms/step -
accuracy: 0.9500 - loss: 0.1688 - val_accuracy: 0.6667 - val_loss: 1.2970
Epoch 35/50
64/64            4s 62ms/step -
accuracy: 0.9497 - loss: 0.1456 - val_accuracy: 0.6771 - val_loss: 1.2745
Epoch 36/50
64/64            4s 62ms/step -
accuracy: 0.9612 - loss: 0.1431 - val_accuracy: 0.6771 - val_loss: 1.3573
Epoch 37/50
64/64            4s 63ms/step -
accuracy: 0.9459 - loss: 0.1550 - val_accuracy: 0.6741 - val_loss: 1.3014
Epoch 38/50
64/64            4s 62ms/step -
accuracy: 0.9541 - loss: 0.1509 - val_accuracy: 0.6756 - val_loss: 1.3280
Epoch 39/50
64/64            4s 63ms/step -
accuracy: 0.9491 - loss: 0.1604 - val_accuracy: 0.6741 - val_loss: 1.3327
Epoch 40/50
64/64            4s 63ms/step -
accuracy: 0.9643 - loss: 0.1189 - val_accuracy: 0.6682 - val_loss: 1.3403
Epoch 41/50
64/64            4s 64ms/step -
accuracy: 0.9601 - loss: 0.1220 - val_accuracy: 0.6741 - val_loss: 1.3032
Epoch 42/50
64/64            4s 64ms/step -
accuracy: 0.9578 - loss: 0.1246 - val_accuracy: 0.6860 - val_loss: 1.3289
Epoch 43/50
64/64            4s 64ms/step -
accuracy: 0.9584 - loss: 0.1316 - val_accuracy: 0.6815 - val_loss: 1.3691
Epoch 44/50
64/64            4s 63ms/step -
accuracy: 0.9554 - loss: 0.1309 - val_accuracy: 0.6786 - val_loss: 1.3708
Epoch 45/50
64/64            4s 64ms/step -
accuracy: 0.9563 - loss: 0.1302 - val_accuracy: 0.6935 - val_loss: 1.3532
Epoch 46/50
64/64            4s 63ms/step -
accuracy: 0.9596 - loss: 0.1284 - val_accuracy: 0.6756 - val_loss: 1.3537
Epoch 47/50
64/64            4s 63ms/step -
accuracy: 0.9667 - loss: 0.1133 - val_accuracy: 0.6786 - val_loss: 1.4074
Epoch 48/50
64/64            4s 63ms/step -
accuracy: 0.9715 - loss: 0.1120 - val_accuracy: 0.6756 - val_loss: 1.3911
```

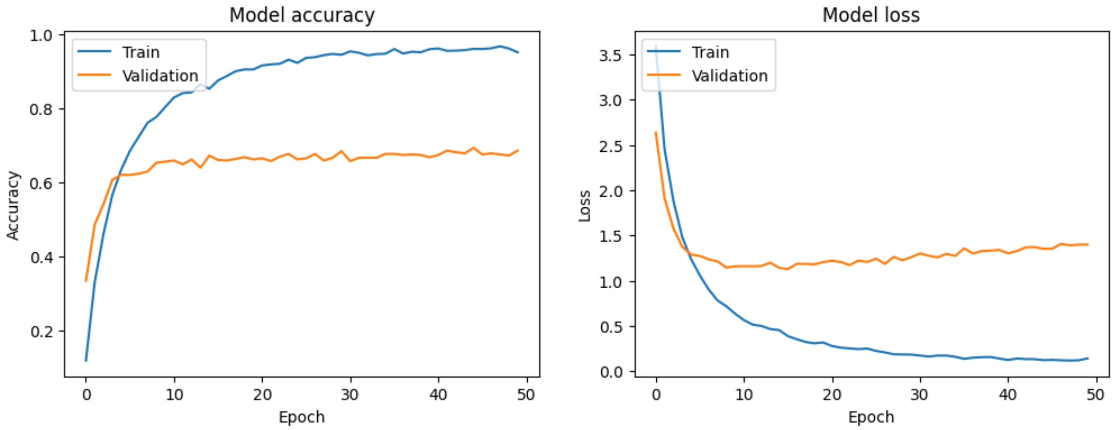
```
Epoch 49/50
64/64              4s 64ms/step -
accuracy: 0.9687 - loss: 0.1138 - val_accuracy: 0.6726 - val_loss: 1.3987
Epoch 50/50
64/64              4s 64ms/step -
accuracy: 0.9553 - loss: 0.1447 - val_accuracy: 0.6860 - val_loss: 1.3996
```

[7]:
```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              4s 188ms/step -
accuracy: 0.7031 - loss: 1.3048
Test Accuracy: 0.6797
Test Loss: 1.4072
```

[8]:
```python
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import numpy as np
```

[9]:
```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()
```



[10]:
```
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()
```

```
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

               precision    recall  f1-score   support

           A      1.000     0.286     0.444         7
           B      0.300     0.500     0.375         6
           C      0.444     0.400     0.421        20
           D      0.500     0.200     0.286        10
           E      0.375     0.600     0.462        10
           F      0.333     0.444     0.381         9
           G      0.625     0.455     0.526        11
           H      0.375     0.300     0.333        10
           I      0.229     0.400     0.291        20
           J      0.370     0.455     0.408        22
           K      0.143     0.100     0.118        10
           L      0.474     0.529     0.500        17
           M      0.500     0.833     0.625         6
           N      0.000     0.000     0.000         9
           O      0.400     0.421     0.410        19
           P      0.385     0.556     0.455         9
           Q      0.182     0.182     0.182        11
           R      0.300     0.391     0.340        23
           S      0.500     0.400     0.444        15
           T      0.462     0.600     0.522        10
           U      0.167     0.053     0.080        19
           V      0.375     0.286     0.324        21
           W      0.667     0.571     0.615        14
           X      0.500     0.333     0.400         6
           Y      0.333     0.200     0.250         5
           Z      0.720     0.818     0.766        22
        baca      0.917     1.000     0.957        11
       bantu      0.769     1.000     0.870        10
       bapak      0.923     0.923     0.923        13
buangairkecil      1.000     0.667     0.800         9
        buat      0.900     0.900     0.900        10
        halo      1.000     1.000     1.000        16
         ibu      1.000     1.000     1.000         6
        kamu      0.955     0.913     0.933        23
```

maaf	1.000	0.944	0.971	18
makan	0.882	1.000	0.938	15
mau	1.000	1.000	1.000	12
nama	1.000	0.833	0.909	18
pagi	1.000	1.000	1.000	15
paham	1.000	0.952	0.976	21
sakit	1.000	1.000	1.000	2
sama-sama	0.958	0.958	0.958	24
saya	1.000	0.714	0.833	7
selamat	0.935	0.967	0.951	30
siapa	0.882	0.938	0.909	16
tanya	1.000	0.933	0.966	15
tempat	1.000	0.800	0.889	5
terima-kasih	0.880	1.000	0.936	22
terlambat	0.923	1.000	0.960	12
tidak	0.900	1.000	0.947	9
tolong	1.000	1.000	1.000	13
accuracy			0.680	693
macro avg	0.676	0.662	0.657	693
weighted avg	0.682	0.680	0.672	693