COMPARISON_CNN

June 21, 2025

[8]:
```python
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
import os
from PIL import Image, UnidentifiedImageError
import shutil

# Configuration
IMG_SIZE = (28, 28)
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="grayscale",
    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="grayscale",
    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.
Using 1293 files for training.
Found 2155 files belonging to 25 classes.
Using 862 files for validation.
```

[10]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout,␣
↪BatchNormalization,Input
from tensorflow.keras.layers import Flatten, Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
model = Sequential([
    Input((28, 28, 1)),
    Conv2D(16, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.1),

    Conv2D(32, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    GlobalAveragePooling2D(),
    Flatten(),

    Dense(128, activation='relu'),
    Dropout(0.2),
```

```
    Dense(num_classes, activation='softmax')
])

model.compile(optimizer=Adam(1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[11]:
```
history = model.fit(train_ds, validation_data=val_ds, epochs=50)
```

```
Epoch 1/50
41/41                 4s 30ms/step -
accuracy: 0.0792 - loss: 3.1791 - val_accuracy: 0.0553 - val_loss: 3.2039
Epoch 2/50
41/41                 0s 8ms/step -
accuracy: 0.0990 - loss: 2.9955 - val_accuracy: 0.0721 - val_loss: 3.1999
Epoch 3/50
41/41                 0s 8ms/step -
accuracy: 0.1329 - loss: 2.8954 - val_accuracy: 0.0697 - val_loss: 3.2144
Epoch 4/50
41/41                 0s 7ms/step -
accuracy: 0.1812 - loss: 2.7469 - val_accuracy: 0.0457 - val_loss: 3.3359
Epoch 5/50
41/41                 0s 8ms/step -
accuracy: 0.1987 - loss: 2.6381 - val_accuracy: 0.0577 - val_loss: 3.5013
Epoch 6/50
41/41                 0s 7ms/step -
accuracy: 0.2533 - loss: 2.5510 - val_accuracy: 0.0385 - val_loss: 3.7691
Epoch 7/50
41/41                 0s 7ms/step -
accuracy: 0.3091 - loss: 2.4013 - val_accuracy: 0.0457 - val_loss: 3.9707
Epoch 8/50
41/41                 0s 8ms/step -
accuracy: 0.3289 - loss: 2.2726 - val_accuracy: 0.0962 - val_loss: 3.9763
Epoch 9/50
41/41                 0s 8ms/step -
accuracy: 0.3697 - loss: 2.1622 - val_accuracy: 0.0793 - val_loss: 4.0191
Epoch 10/50
41/41                 0s 8ms/step -
accuracy: 0.4231 - loss: 1.9860 - val_accuracy: 0.0697 - val_loss: 3.9028
Epoch 11/50
41/41                 0s 8ms/step -
accuracy: 0.4787 - loss: 1.8349 - val_accuracy: 0.0601 - val_loss: 3.8484
Epoch 12/50
41/41                 0s 7ms/step -
accuracy: 0.4719 - loss: 1.7966 - val_accuracy: 0.1010 - val_loss: 3.4286
Epoch 13/50
41/41                 0s 8ms/step -
accuracy: 0.4923 - loss: 1.6918 - val_accuracy: 0.1202 - val_loss: 3.2569
```

```
Epoch 14/50
41/41            0s 8ms/step -
accuracy: 0.5286 - loss: 1.5895 - val_accuracy: 0.1538 - val_loss: 3.2706
Epoch 15/50
41/41            0s 8ms/step -
accuracy: 0.5598 - loss: 1.5081 - val_accuracy: 0.1923 - val_loss: 2.5698
Epoch 16/50
41/41            0s 7ms/step -
accuracy: 0.5595 - loss: 1.3816 - val_accuracy: 0.2404 - val_loss: 2.4932
Epoch 17/50
41/41            0s 8ms/step -
accuracy: 0.5971 - loss: 1.3521 - val_accuracy: 0.2404 - val_loss: 2.4065
Epoch 18/50
41/41            0s 8ms/step -
accuracy: 0.6317 - loss: 1.3178 - val_accuracy: 0.3245 - val_loss: 2.1631
Epoch 19/50
41/41            0s 8ms/step -
accuracy: 0.6310 - loss: 1.2264 - val_accuracy: 0.3438 - val_loss: 2.1272
Epoch 20/50
41/41            0s 7ms/step -
accuracy: 0.6476 - loss: 1.2063 - val_accuracy: 0.5264 - val_loss: 1.5468
Epoch 21/50
41/41            0s 8ms/step -
accuracy: 0.6589 - loss: 1.1364 - val_accuracy: 0.5457 - val_loss: 1.5659
Epoch 22/50
41/41            0s 8ms/step -
accuracy: 0.6765 - loss: 1.0622 - val_accuracy: 0.5673 - val_loss: 1.4684
Epoch 23/50
41/41            0s 8ms/step -
accuracy: 0.6960 - loss: 1.0273 - val_accuracy: 0.6322 - val_loss: 1.2309
Epoch 24/50
41/41            0s 8ms/step -
accuracy: 0.6858 - loss: 1.0117 - val_accuracy: 0.5553 - val_loss: 1.3881
Epoch 25/50
41/41            0s 8ms/step -
accuracy: 0.6939 - loss: 1.0023 - val_accuracy: 0.6635 - val_loss: 1.1033
Epoch 26/50
41/41            0s 8ms/step -
accuracy: 0.7328 - loss: 0.8993 - val_accuracy: 0.7212 - val_loss: 1.0240
Epoch 27/50
41/41            0s 8ms/step -
accuracy: 0.7280 - loss: 0.9257 - val_accuracy: 0.4808 - val_loss: 1.6999
Epoch 28/50
41/41            0s 7ms/step -
accuracy: 0.7392 - loss: 0.8670 - val_accuracy: 0.6562 - val_loss: 1.1325
Epoch 29/50
41/41            0s 8ms/step -
accuracy: 0.7125 - loss: 0.9107 - val_accuracy: 0.7476 - val_loss: 0.9960
```

```
Epoch 30/50
41/41            0s 8ms/step -
accuracy: 0.7633 - loss: 0.8272 - val_accuracy: 0.5457 - val_loss: 1.5624
Epoch 31/50
41/41            0s 7ms/step -
accuracy: 0.7518 - loss: 0.8066 - val_accuracy: 0.6971 - val_loss: 1.0020
Epoch 32/50
41/41            0s 7ms/step -
accuracy: 0.7725 - loss: 0.7367 - val_accuracy: 0.6442 - val_loss: 1.1853
Epoch 33/50
41/41            0s 7ms/step -
accuracy: 0.7559 - loss: 0.7928 - val_accuracy: 0.7043 - val_loss: 0.9951
Epoch 34/50
41/41            0s 8ms/step -
accuracy: 0.7950 - loss: 0.7116 - val_accuracy: 0.8005 - val_loss: 0.7544
Epoch 35/50
41/41            0s 7ms/step -
accuracy: 0.7948 - loss: 0.6844 - val_accuracy: 0.6827 - val_loss: 1.0278
Epoch 36/50
41/41            0s 8ms/step -
accuracy: 0.7684 - loss: 0.7328 - val_accuracy: 0.7548 - val_loss: 0.8494
Epoch 37/50
41/41            0s 8ms/step -
accuracy: 0.7618 - loss: 0.7166 - val_accuracy: 0.8413 - val_loss: 0.6279
Epoch 38/50
41/41            0s 7ms/step -
accuracy: 0.8095 - loss: 0.6516 - val_accuracy: 0.8101 - val_loss: 0.6800
Epoch 39/50
41/41            0s 8ms/step -
accuracy: 0.8033 - loss: 0.6262 - val_accuracy: 0.8029 - val_loss: 0.6571
Epoch 40/50
41/41            0s 7ms/step -
accuracy: 0.7766 - loss: 0.7060 - val_accuracy: 0.7692 - val_loss: 0.8074
Epoch 41/50
41/41            0s 8ms/step -
accuracy: 0.7863 - loss: 0.6773 - val_accuracy: 0.7981 - val_loss: 0.7020
Epoch 42/50
41/41            0s 8ms/step -
accuracy: 0.8257 - loss: 0.5982 - val_accuracy: 0.7091 - val_loss: 0.9816
Epoch 43/50
41/41            0s 8ms/step -
accuracy: 0.8237 - loss: 0.5878 - val_accuracy: 0.6442 - val_loss: 1.1126
Epoch 44/50
41/41            0s 7ms/step -
accuracy: 0.7964 - loss: 0.6137 - val_accuracy: 0.7308 - val_loss: 0.8835
Epoch 45/50
41/41            0s 7ms/step -
accuracy: 0.7883 - loss: 0.6493 - val_accuracy: 0.7885 - val_loss: 0.7244
```

```
Epoch 46/50
41/41              0s 7ms/step -
accuracy: 0.8074 - loss: 0.5850 - val_accuracy: 0.7260 - val_loss: 0.8995
Epoch 47/50
41/41              0s 7ms/step -
accuracy: 0.8247 - loss: 0.5645 - val_accuracy: 0.7620 - val_loss: 0.7885
Epoch 48/50
41/41              0s 7ms/step -
accuracy: 0.8261 - loss: 0.5454 - val_accuracy: 0.8125 - val_loss: 0.6782
Epoch 49/50
41/41              0s 7ms/step -
accuracy: 0.8410 - loss: 0.5211 - val_accuracy: 0.7163 - val_loss: 1.0502
Epoch 50/50
41/41              0s 7ms/step -
accuracy: 0.8434 - loss: 0.5401 - val_accuracy: 0.7308 - val_loss: 0.9555
```

[12]:

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

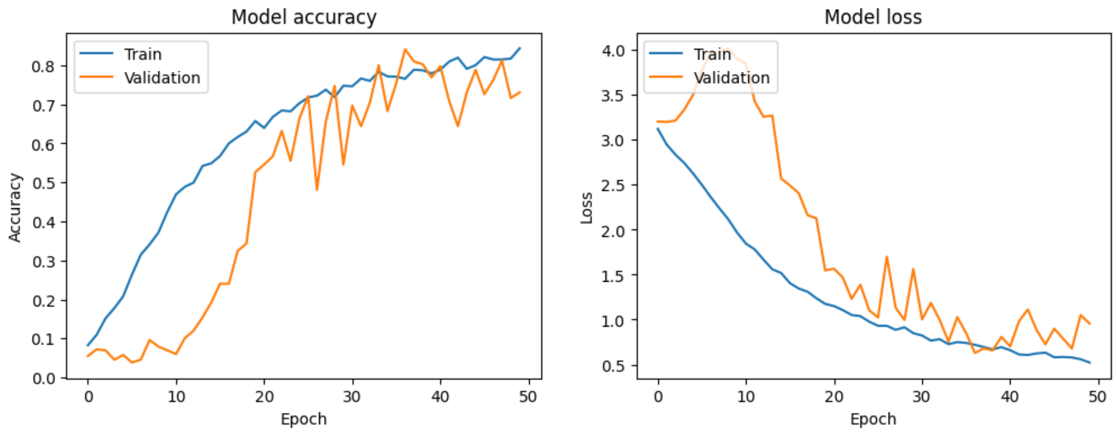
```
14/14              0s 16ms/step -
accuracy: 0.7293 - loss: 0.9152
Test Accuracy: 0.7399
Test Loss: 0.9218
```

[13]:

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

[14]:

```
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()
```



[15]:

```
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()
```

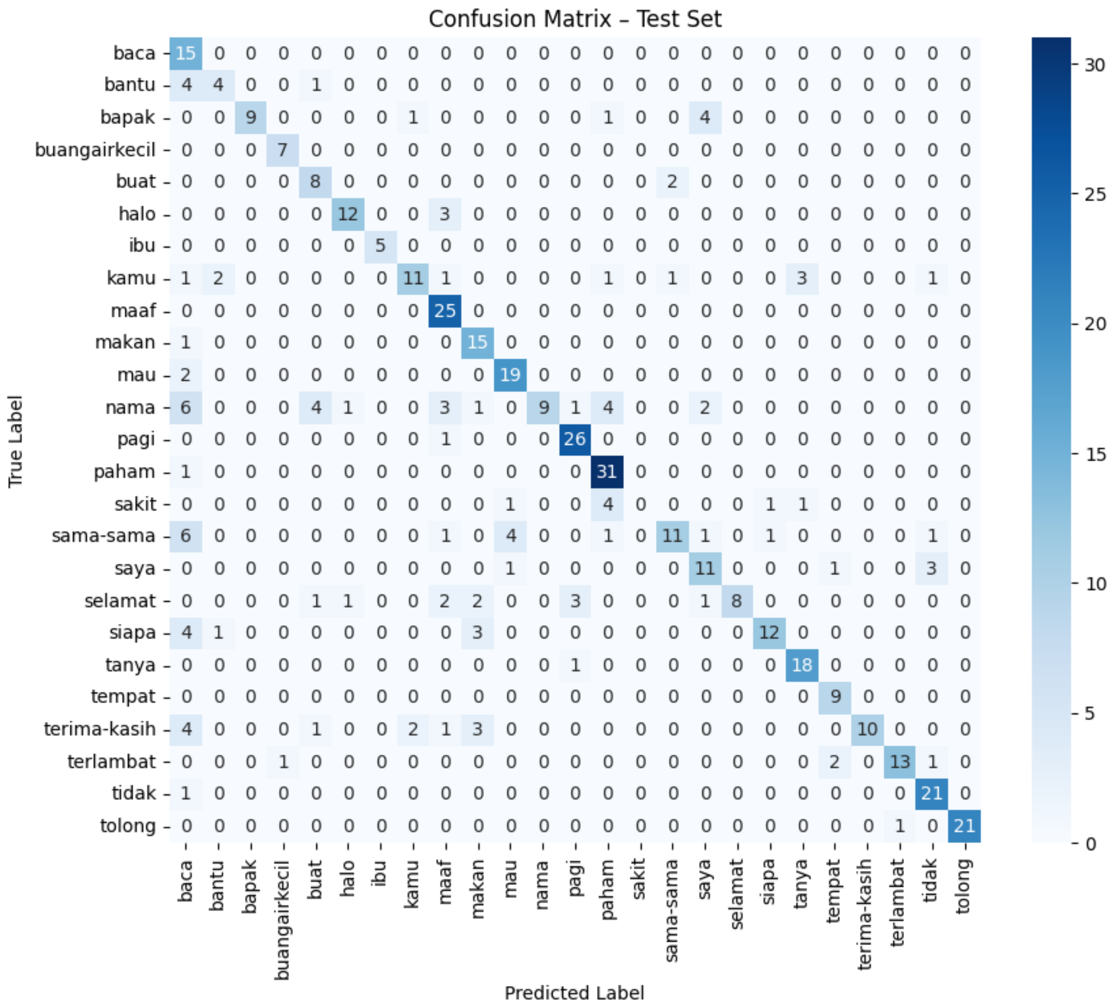
```
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
```

```
  _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

         baca      0.333     1.000     0.500        15
        bantu      0.571     0.444     0.500         9
        bapak      1.000     0.600     0.750        15
buangairkecil      0.875     1.000     0.933         7
         buat      0.533     0.800     0.640        10
         halo      0.857     0.800     0.828        15
          ibu      1.000     1.000     1.000         5
         kamu      0.786     0.524     0.629        21
         maaf      0.676     1.000     0.806        25
        makan      0.625     0.938     0.750        16
          mau      0.760     0.905     0.826        21
         nama      1.000     0.290     0.450        31
         pagi      0.839     0.963     0.897        27
        paham      0.738     0.969     0.838        32
        sakit      0.000     0.000     0.000         7
    sama-sama      0.786     0.423     0.550        26
         saya      0.579     0.688     0.629        16
      selamat      1.000     0.444     0.615        18
        siapa      0.857     0.600     0.706        20
        tanya      0.818     0.947     0.878        19
       tempat      0.750     1.000     0.857         9
 terima-kasih      1.000     0.476     0.645        21
    terlambat      0.929     0.765     0.839        17
        tidak      0.778     0.955     0.857        22
       tolong      1.000     0.955     0.977        22

     accuracy                          0.740       446
    macro avg      0.764     0.739     0.716       446
 weighted avg      0.794     0.740     0.725       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
30
25
20
15
10
5
0