COMPARISON_CNN

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 = (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
```

```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="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 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,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'])
```

[3]:
```
history = model.fit(train_ds, validation_data=val_ds, epochs=50)
```

```
Epoch 1/50
106/106             19s 67ms/step -
accuracy: 0.0285 - loss: 3.9030 - val_accuracy: 0.0188 - val_loss: 3.9602
Epoch 2/50
106/106             1s 7ms/step -
accuracy: 0.0746 - loss: 3.4015 - val_accuracy: 0.0205 - val_loss: 4.2724
Epoch 3/50
106/106             1s 7ms/step -
accuracy: 0.0855 - loss: 3.2238 - val_accuracy: 0.0232 - val_loss: 4.3350
Epoch 4/50
106/106             1s 7ms/step -
accuracy: 0.0961 - loss: 3.1364 - val_accuracy: 0.0259 - val_loss: 3.9981
Epoch 5/50
106/106             1s 7ms/step -
accuracy: 0.1320 - loss: 3.0458 - val_accuracy: 0.0473 - val_loss: 3.6321
Epoch 6/50
106/106             1s 7ms/step -
accuracy: 0.1539 - loss: 2.9616 - val_accuracy: 0.1437 - val_loss: 3.1492
Epoch 7/50
106/106             1s 7ms/step -
accuracy: 0.1671 - loss: 2.8833 - val_accuracy: 0.1375 - val_loss: 3.1823
Epoch 8/50
106/106             1s 7ms/step -
accuracy: 0.2108 - loss: 2.8134 - val_accuracy: 0.1009 - val_loss: 3.2990
Epoch 9/50
106/106             1s 7ms/step -
accuracy: 0.2060 - loss: 2.7633 - val_accuracy: 0.1411 - val_loss: 3.0562
Epoch 10/50
106/106             1s 7ms/step -
accuracy: 0.2275 - loss: 2.6895 - val_accuracy: 0.1857 - val_loss: 2.7917
Epoch 11/50
106/106             1s 7ms/step -
accuracy: 0.2492 - loss: 2.6086 - val_accuracy: 0.2232 - val_loss: 2.7362
Epoch 12/50
106/106             1s 7ms/step -
accuracy: 0.2650 - loss: 2.5422 - val_accuracy: 0.1804 - val_loss: 2.8804
Epoch 13/50
106/106             1s 7ms/step -
accuracy: 0.2629 - loss: 2.5373 - val_accuracy: 0.2464 - val_loss: 2.6650
Epoch 14/50
```

```
106/106              1s 7ms/step -
accuracy: 0.2880 - loss: 2.4425 - val_accuracy: 0.2482 - val_loss: 2.6002
Epoch 15/50
106/106              1s 7ms/step -
accuracy: 0.2977 - loss: 2.4105 - val_accuracy: 0.2688 - val_loss: 2.6124
Epoch 16/50
106/106              1s 7ms/step -
accuracy: 0.3190 - loss: 2.3837 - val_accuracy: 0.2438 - val_loss: 2.6162
Epoch 17/50
106/106              1s 7ms/step -
accuracy: 0.3104 - loss: 2.3365 - val_accuracy: 0.2812 - val_loss: 2.4488
Epoch 18/50
106/106              1s 7ms/step -
accuracy: 0.3286 - loss: 2.2932 - val_accuracy: 0.2759 - val_loss: 2.4608
Epoch 19/50
106/106              1s 7ms/step -
accuracy: 0.3545 - loss: 2.2151 - val_accuracy: 0.1884 - val_loss: 2.8882
Epoch 20/50
106/106              1s 7ms/step -
accuracy: 0.3476 - loss: 2.2272 - val_accuracy: 0.3071 - val_loss: 2.3932
Epoch 21/50
106/106              1s 7ms/step -
accuracy: 0.3327 - loss: 2.2491 - val_accuracy: 0.1875 - val_loss: 2.8813
Epoch 22/50
106/106              1s 7ms/step -
accuracy: 0.3645 - loss: 2.1698 - val_accuracy: 0.2536 - val_loss: 2.5397
Epoch 23/50
106/106              1s 7ms/step -
accuracy: 0.3512 - loss: 2.1625 - val_accuracy: 0.2929 - val_loss: 2.4505
Epoch 24/50
106/106              1s 7ms/step -
accuracy: 0.3633 - loss: 2.1475 - val_accuracy: 0.3170 - val_loss: 2.3849
Epoch 25/50
106/106              1s 7ms/step -
accuracy: 0.3645 - loss: 2.1227 - val_accuracy: 0.1929 - val_loss: 2.8297
Epoch 26/50
106/106              1s 7ms/step -
accuracy: 0.3727 - loss: 2.1300 - val_accuracy: 0.2536 - val_loss: 2.6489
Epoch 27/50
106/106              1s 7ms/step -
accuracy: 0.3834 - loss: 2.1089 - val_accuracy: 0.2750 - val_loss: 2.4914
Epoch 28/50
106/106              1s 7ms/step -
accuracy: 0.3833 - loss: 2.0938 - val_accuracy: 0.2812 - val_loss: 2.5172
Epoch 29/50
106/106              1s 7ms/step -
accuracy: 0.3908 - loss: 2.0487 - val_accuracy: 0.3357 - val_loss: 2.2638
Epoch 30/50
```

```
106/106              1s 7ms/step -
accuracy: 0.4000 - loss: 2.0492 - val_accuracy: 0.3071 - val_loss: 2.4016
Epoch 31/50
106/106              1s 7ms/step -
accuracy: 0.3992 - loss: 2.0285 - val_accuracy: 0.2973 - val_loss: 2.4034
Epoch 32/50
106/106              1s 7ms/step -
accuracy: 0.3928 - loss: 1.9994 - val_accuracy: 0.2875 - val_loss: 2.4529
Epoch 33/50
106/106              1s 7ms/step -
accuracy: 0.4004 - loss: 2.0200 - val_accuracy: 0.3366 - val_loss: 2.2722
Epoch 34/50
106/106              1s 7ms/step -
accuracy: 0.3933 - loss: 2.0160 - val_accuracy: 0.3134 - val_loss: 2.3764
Epoch 35/50
106/106              1s 7ms/step -
accuracy: 0.4115 - loss: 1.9637 - val_accuracy: 0.2937 - val_loss: 2.4489
Epoch 36/50
106/106              1s 7ms/step -
accuracy: 0.4028 - loss: 1.9667 - val_accuracy: 0.3384 - val_loss: 2.2480
Epoch 37/50
106/106              1s 7ms/step -
accuracy: 0.3932 - loss: 1.9684 - val_accuracy: 0.3170 - val_loss: 2.3287
Epoch 38/50
106/106              1s 7ms/step -
accuracy: 0.4251 - loss: 1.9092 - val_accuracy: 0.2625 - val_loss: 2.6642
Epoch 39/50
106/106              1s 8ms/step -
accuracy: 0.4244 - loss: 1.9183 - val_accuracy: 0.2589 - val_loss: 2.5868
Epoch 40/50
106/106              1s 8ms/step -
accuracy: 0.4159 - loss: 1.9364 - val_accuracy: 0.3241 - val_loss: 2.2809
Epoch 41/50
106/106              1s 8ms/step -
accuracy: 0.4319 - loss: 1.8823 - val_accuracy: 0.3446 - val_loss: 2.2653
Epoch 42/50
106/106              1s 8ms/step -
accuracy: 0.4493 - loss: 1.8265 - val_accuracy: 0.3313 - val_loss: 2.2376
Epoch 43/50
106/106              1s 8ms/step -
accuracy: 0.4392 - loss: 1.8844 - val_accuracy: 0.3509 - val_loss: 2.2370
Epoch 44/50
106/106              1s 7ms/step -
accuracy: 0.4439 - loss: 1.8559 - val_accuracy: 0.3634 - val_loss: 2.2481
Epoch 45/50
106/106              1s 7ms/step -
accuracy: 0.4469 - loss: 1.8406 - val_accuracy: 0.3482 - val_loss: 2.2282
Epoch 46/50
```

```
106/106 1s 7ms/step -
accuracy: 0.4548 - loss: 1.8436 - val_accuracy: 0.3580 - val_loss: 2.2125
Epoch 47/50
106/106 1s 7ms/step -
accuracy: 0.4498 - loss: 1.8072 - val_accuracy: 0.3036 - val_loss: 2.4754
Epoch 48/50
106/106 1s 7ms/step -
accuracy: 0.4469 - loss: 1.8241 - val_accuracy: 0.3393 - val_loss: 2.2447
Epoch 49/50
106/106 1s 7ms/step -
accuracy: 0.4348 - loss: 1.8508 - val_accuracy: 0.2866 - val_loss: 2.5828
Epoch 50/50
106/106 1s 7ms/step -
accuracy: 0.4575 - loss: 1.7862 - val_accuracy: 0.3625 - val_loss: 2.2806
```

[4]:

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

```
36/36 5s 136ms/step -
accuracy: 0.3699 - loss: 2.2056
Test Accuracy: 0.3817
Test Loss: 2.2056
```

[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()
```

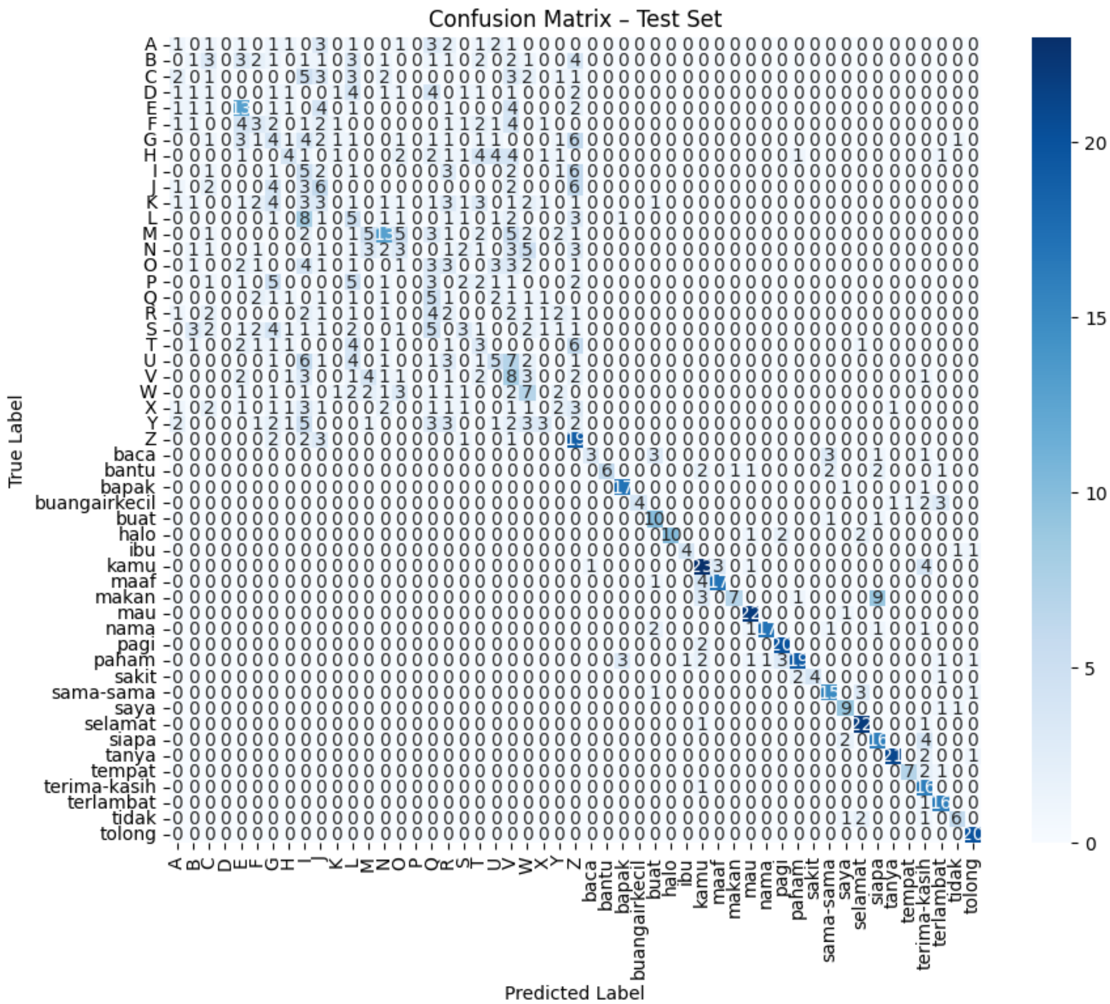
```
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
```

```
  _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      0.083     0.053     0.065        19
           B      0.091     0.037     0.053        27
           C      0.050     0.043     0.047        23
           D      0.000     0.000     0.000        21
           E      0.361     0.433     0.394        30
           F      0.188     0.120     0.146        25
           G      0.105     0.129     0.116        31
           H      0.286     0.138     0.186        29
           I      0.083     0.227     0.122        22
           J      0.162     0.250     0.197        24
           K      0.000     0.000     0.000        32
           L      0.116     0.192     0.145        26
           M      0.333     0.116     0.172        43
           N      0.067     0.071     0.069        28
           O      0.048     0.038     0.043        26
           P      0.000     0.000     0.000        24
           Q      0.122     0.278     0.169        18
           R      0.062     0.095     0.075        21
           S      0.200     0.094     0.128        32
           T      0.115     0.136     0.125        22
           U      0.238     0.152     0.185        33
           V      0.129     0.267     0.174        30
           W      0.206     0.269     0.233        26
           X      0.000     0.000     0.000        23
           Y      0.000     0.000     0.000        29
           Z      0.260     0.679     0.376        28
        baca      0.750     0.273     0.400        11
       bantu      1.000     0.400     0.571        15
       bapak      0.810     0.895     0.850        19
buangairkecil      1.000     0.364     0.533        11
        buat      0.556     0.833     0.667        12
        halo      1.000     0.667     0.800        15
         ibu      0.800     0.667     0.727         6
        kamu      0.605     0.719     0.657        32
```

maaf	0.850	0.773	0.810	22
makan	0.875	0.350	0.500	20
mau	0.815	0.957	0.880	23
nama	0.944	0.739	0.829	23
pagi	0.800	0.909	0.851	22
paham	0.826	0.594	0.691	32
sakit	1.000	0.571	0.727	7
sama-sama	0.682	0.750	0.714	20
saya	0.643	0.818	0.720	11
selamat	0.733	0.917	0.815	24
siapa	0.533	0.727	0.615	22
tanya	0.913	0.875	0.894	24
tempat	0.875	0.700	0.778	10
terima-kasih	0.432	0.941	0.593	17
terlambat	0.640	0.941	0.762	17
tidak	0.667	0.600	0.632	10
tolong	0.833	1.000	0.909	20
accuracy			0.382	1137
macro avg	0.449	0.427	0.415	1137
weighted avg	0.388	0.382	0.365	1137


Confusion Matrix – Test Set
True Label
Predicted Label