COMPARISON_CNN

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 = (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 3413 files belonging to 51 classes.
Using 2048 files for training.
Found 3413 files belonging to 51 classes.
Using 1365 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
64/64                 12s 66ms/step -
accuracy: 0.0291 - loss: 3.9148 - val_accuracy: 0.0283 - val_loss: 3.8983
Epoch 2/50
64/64                 0s 7ms/step -
accuracy: 0.0739 - loss: 3.4691 - val_accuracy: 0.0312 - val_loss: 3.9690
Epoch 3/50
64/64                 0s 7ms/step -
accuracy: 0.0885 - loss: 3.2448 - val_accuracy: 0.0342 - val_loss: 4.1611
Epoch 4/50
64/64                 0s 7ms/step -
accuracy: 0.1019 - loss: 3.1518 - val_accuracy: 0.0327 - val_loss: 4.2911
Epoch 5/50
64/64                 0s 7ms/step -
accuracy: 0.1232 - loss: 2.9983 - val_accuracy: 0.0283 - val_loss: 4.3891
Epoch 6/50
64/64                 0s 7ms/step -
accuracy: 0.1406 - loss: 2.9476 - val_accuracy: 0.0298 - val_loss: 4.3673
Epoch 7/50
64/64                 0s 7ms/step -
accuracy: 0.1614 - loss: 2.8477 - val_accuracy: 0.0268 - val_loss: 3.9981
Epoch 8/50
64/64                 0s 7ms/step -
accuracy: 0.1785 - loss: 2.7901 - val_accuracy: 0.0238 - val_loss: 3.9778
Epoch 9/50
64/64                 0s 8ms/step -
accuracy: 0.2282 - loss: 2.6664 - val_accuracy: 0.1533 - val_loss: 3.1297
Epoch 10/50
64/64                 0s 7ms/step -
accuracy: 0.2433 - loss: 2.6476 - val_accuracy: 0.1652 - val_loss: 3.0007
Epoch 11/50
64/64                 0s 7ms/step -
accuracy: 0.2553 - loss: 2.5648 - val_accuracy: 0.2679 - val_loss: 2.7729
Epoch 12/50
64/64                 0s 7ms/step -
accuracy: 0.2484 - loss: 2.5006 - val_accuracy: 0.1786 - val_loss: 2.8793
Epoch 13/50
64/64                 0s 8ms/step -
accuracy: 0.2863 - loss: 2.4378 - val_accuracy: 0.2530 - val_loss: 2.6473
Epoch 14/50
```

```
64/64              0s 7ms/step -
accuracy: 0.3062 - loss: 2.3813 - val_accuracy: 0.1458 - val_loss: 3.0356
Epoch 15/50
64/64              0s 7ms/step -
accuracy: 0.3170 - loss: 2.3353 - val_accuracy: 0.2485 - val_loss: 2.5999
Epoch 16/50
64/64              0s 7ms/step -
accuracy: 0.3181 - loss: 2.3354 - val_accuracy: 0.2857 - val_loss: 2.4830
Epoch 17/50
64/64              0s 7ms/step -
accuracy: 0.3285 - loss: 2.2751 - val_accuracy: 0.3006 - val_loss: 2.4481
Epoch 18/50
64/64              0s 7ms/step -
accuracy: 0.3242 - loss: 2.2067 - val_accuracy: 0.3333 - val_loss: 2.3466
Epoch 19/50
64/64              0s 7ms/step -
accuracy: 0.3539 - loss: 2.1365 - val_accuracy: 0.3170 - val_loss: 2.4240
Epoch 20/50
64/64              0s 7ms/step -
accuracy: 0.3646 - loss: 2.1836 - val_accuracy: 0.2991 - val_loss: 2.3726
Epoch 21/50
64/64              0s 7ms/step -
accuracy: 0.3792 - loss: 2.0903 - val_accuracy: 0.2812 - val_loss: 2.4256
Epoch 22/50
64/64              0s 7ms/step -
accuracy: 0.4028 - loss: 2.0472 - val_accuracy: 0.3646 - val_loss: 2.1623
Epoch 23/50
64/64              0s 7ms/step -
accuracy: 0.3948 - loss: 2.0149 - val_accuracy: 0.3393 - val_loss: 2.2211
Epoch 24/50
64/64              0s 7ms/step -
accuracy: 0.4132 - loss: 2.0069 - val_accuracy: 0.3512 - val_loss: 2.2401
Epoch 25/50
64/64              0s 7ms/step -
accuracy: 0.3904 - loss: 2.0454 - val_accuracy: 0.3304 - val_loss: 2.2851
Epoch 26/50
64/64              0s 7ms/step -
accuracy: 0.4307 - loss: 1.9510 - val_accuracy: 0.3661 - val_loss: 2.2174
Epoch 27/50
64/64              0s 7ms/step -
accuracy: 0.4220 - loss: 1.9560 - val_accuracy: 0.3705 - val_loss: 2.1760
Epoch 28/50
64/64              1s 8ms/step -
accuracy: 0.4523 - loss: 1.8791 - val_accuracy: 0.3482 - val_loss: 2.2106
Epoch 29/50
64/64              0s 7ms/step -
accuracy: 0.4453 - loss: 1.8819 - val_accuracy: 0.3452 - val_loss: 2.1968
Epoch 30/50
```

```
64/64            0s 7ms/step -
accuracy: 0.4520 - loss: 1.8477 - val_accuracy: 0.3765 - val_loss: 2.1629
Epoch 31/50
64/64            0s 7ms/step -
accuracy: 0.4347 - loss: 1.8650 - val_accuracy: 0.3854 - val_loss: 2.0757
Epoch 32/50
64/64            0s 7ms/step -
accuracy: 0.4503 - loss: 1.8200 - val_accuracy: 0.3810 - val_loss: 2.1927
Epoch 33/50
64/64            0s 7ms/step -
accuracy: 0.4655 - loss: 1.7853 - val_accuracy: 0.3125 - val_loss: 2.3682
Epoch 34/50
64/64            0s 7ms/step -
accuracy: 0.4765 - loss: 1.7456 - val_accuracy: 0.3929 - val_loss: 2.0664
Epoch 35/50
64/64            0s 7ms/step -
accuracy: 0.4916 - loss: 1.7083 - val_accuracy: 0.3497 - val_loss: 2.1840
Epoch 36/50
64/64            0s 7ms/step -
accuracy: 0.4986 - loss: 1.7002 - val_accuracy: 0.3676 - val_loss: 2.2507
Epoch 37/50
64/64            0s 7ms/step -
accuracy: 0.4835 - loss: 1.7310 - val_accuracy: 0.3914 - val_loss: 2.1696
Epoch 38/50
64/64            0s 7ms/step -
accuracy: 0.4799 - loss: 1.7522 - val_accuracy: 0.4048 - val_loss: 2.0658
Epoch 39/50
64/64            0s 7ms/step -
accuracy: 0.5071 - loss: 1.6724 - val_accuracy: 0.4062 - val_loss: 2.0165
Epoch 40/50
64/64            0s 7ms/step -
accuracy: 0.5237 - loss: 1.6141 - val_accuracy: 0.3988 - val_loss: 2.0732
Epoch 41/50
64/64            0s 8ms/step -
accuracy: 0.5276 - loss: 1.5672 - val_accuracy: 0.3170 - val_loss: 2.3418
Epoch 42/50
64/64            0s 7ms/step -
accuracy: 0.4836 - loss: 1.6447 - val_accuracy: 0.3929 - val_loss: 2.0333
Epoch 43/50
64/64            1s 8ms/step -
accuracy: 0.5172 - loss: 1.6043 - val_accuracy: 0.4003 - val_loss: 2.0836
Epoch 44/50
64/64            0s 7ms/step -
accuracy: 0.5183 - loss: 1.6361 - val_accuracy: 0.3690 - val_loss: 2.1583
Epoch 45/50
64/64            0s 7ms/step -
accuracy: 0.4937 - loss: 1.6420 - val_accuracy: 0.4241 - val_loss: 2.0176
Epoch 46/50
```

```
64/64              1s 8ms/step -
accuracy: 0.5389 - loss: 1.5580 - val_accuracy: 0.4048 - val_loss: 2.1232
Epoch 47/50
64/64              0s 7ms/step -
accuracy: 0.5182 - loss: 1.5634 - val_accuracy: 0.3750 - val_loss: 2.2071
Epoch 48/50
64/64              0s 7ms/step -
accuracy: 0.5255 - loss: 1.5538 - val_accuracy: 0.4211 - val_loss: 1.9964
Epoch 49/50
64/64              0s 7ms/step -
accuracy: 0.5236 - loss: 1.5418 - val_accuracy: 0.3884 - val_loss: 2.1845
Epoch 50/50
64/64              0s 8ms/step -
accuracy: 0.5273 - loss: 1.5547 - val_accuracy: 0.4182 - val_loss: 2.0673
```

[4]:

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

```
22/22              3s 111ms/step -
accuracy: 0.4134 - loss: 2.1195
Test Accuracy: 0.4185
Test Loss: 2.1381
```

[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]:

```python
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()
```

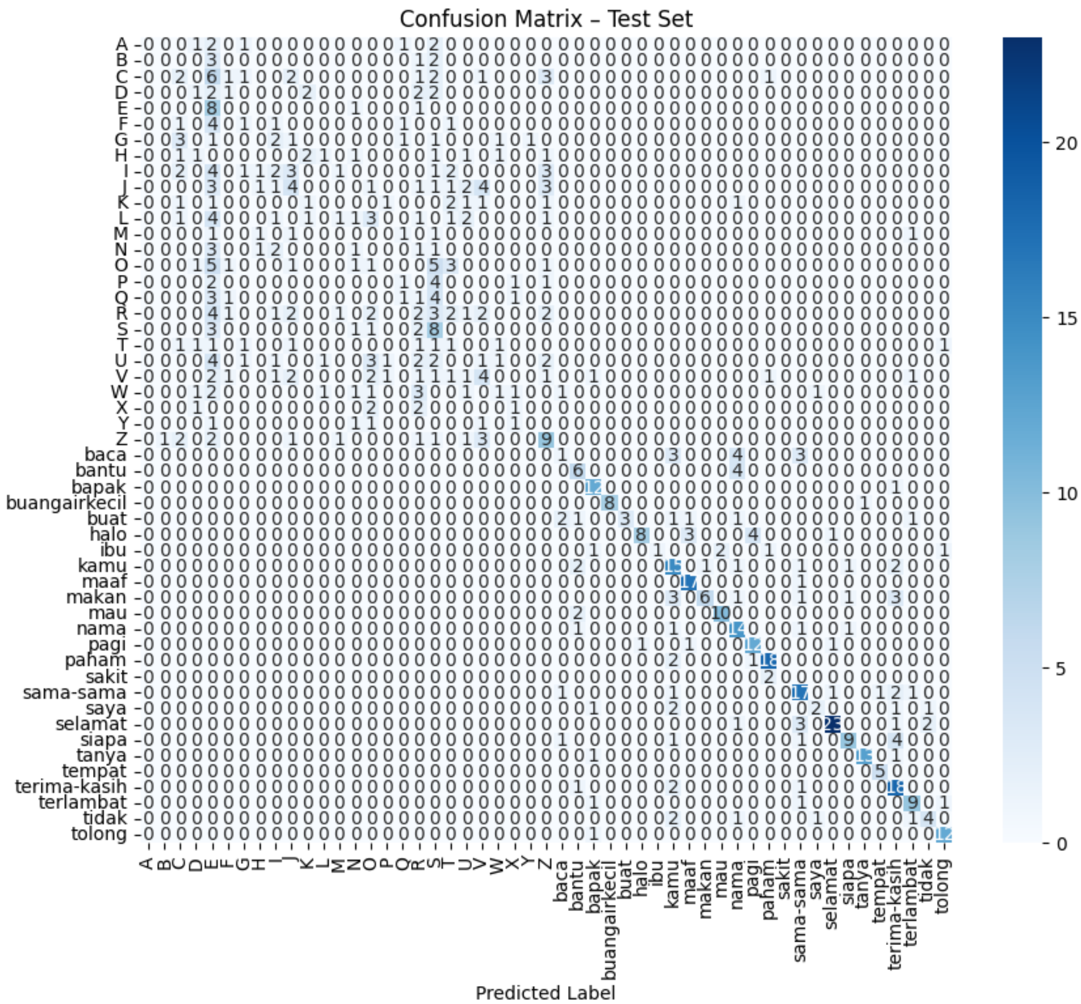
```
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
```

```
  _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      0.000     0.000     0.000         7
           B      0.000     0.000     0.000         6
           C      0.143     0.100     0.118        20
           D      0.143     0.100     0.118        10
           E      0.113     0.800     0.198        10
           F      0.000     0.000     0.000         9
           G      0.000     0.000     0.000        11
           H      0.000     0.000     0.000        10
           I      0.167     0.100     0.125        20
           J      0.222     0.182     0.200        22
           K      0.167     0.100     0.125        10
           L      0.000     0.000     0.000        17
           M      0.000     0.000     0.000         6
           N      0.125     0.111     0.118         9
           O      0.059     0.053     0.056        19
           P      0.000     0.000     0.000         9
           Q      0.167     0.091     0.118        11
           R      0.087     0.087     0.087        23
           S      0.186     0.533     0.276        15
           T      0.071     0.100     0.083        10
           U      0.000     0.000     0.000        19
           V      0.235     0.190     0.211        21
           W      0.200     0.071     0.105        14
           X      0.200     0.167     0.182         6
           Y      0.000     0.000     0.000         5
           Z      0.321     0.409     0.360        22
        baca      0.167     0.091     0.118        11
       bantu      0.462     0.600     0.522        10
       bapak      0.667     0.923     0.774        13
buangairkecil      1.000     0.889     0.941         9
        buat      1.000     0.300     0.462        10
        halo      0.889     0.500     0.640        16
         ibu      1.000     0.167     0.286         6
        kamu      0.455     0.652     0.536        23
```

maaf	0.773	0.944	0.850	18
makan	0.857	0.400	0.545	15
mau	0.833	0.833	0.833	12
nama	0.500	0.778	0.609	18
pagi	0.706	0.800	0.750	15
paham	0.783	0.857	0.818	21
sakit	0.000	0.000	0.000	2
sama-sama	0.567	0.708	0.630	24
saya	0.500	0.286	0.364	7
selamat	0.885	0.767	0.821	30
siapa	0.750	0.562	0.643	16
tanya	0.929	0.867	0.897	15
tempat	0.833	1.000	0.909	5
terima-kasih	0.545	0.818	0.655	22
terlambat	0.643	0.750	0.692	12
tidak	0.571	0.444	0.500	9
tolong	0.800	0.923	0.857	13
accuracy			0.418	693
macro avg	0.387	0.374	0.355	693
weighted avg	0.415	0.418	0.396	693


Confusion Matrix – Test Set
True Label
Predicted Label