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 3488 files belonging to 26 classes.
Using 2093 files for training.
Found 3488 files belonging to 26 classes.
Using 1395 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
66/66                 20s 119ms/step -
accuracy: 0.0374 - loss: 3.3569 - val_accuracy: 0.0420 - val_loss: 3.2620
Epoch 2/50
66/66                 1s 8ms/step -
accuracy: 0.0560 - loss: 3.2389 - val_accuracy: 0.0420 - val_loss: 3.2741
Epoch 3/50
66/66                 1s 8ms/step -
accuracy: 0.0731 - loss: 3.2151 - val_accuracy: 0.0420 - val_loss: 3.2901
Epoch 4/50
66/66                 1s 10ms/step -
accuracy: 0.0801 - loss: 3.1843 - val_accuracy: 0.0420 - val_loss: 3.3281
Epoch 5/50
66/66                 1s 9ms/step -
accuracy: 0.0876 - loss: 3.1476 - val_accuracy: 0.0420 - val_loss: 3.3272
Epoch 6/50
66/66                 1s 8ms/step -
accuracy: 0.0914 - loss: 3.1385 - val_accuracy: 0.0521 - val_loss: 3.2766
Epoch 7/50
66/66                 1s 8ms/step -
accuracy: 0.1079 - loss: 3.1210 - val_accuracy: 0.0478 - val_loss: 3.3657
Epoch 8/50
66/66                 1s 8ms/step -
accuracy: 0.1065 - loss: 3.0950 - val_accuracy: 0.0478 - val_loss: 3.3188
Epoch 9/50
66/66                 1s 10ms/step -
accuracy: 0.1059 - loss: 3.0683 - val_accuracy: 0.0680 - val_loss: 3.2540
Epoch 10/50
66/66                 1s 12ms/step -
accuracy: 0.1172 - loss: 3.0599 - val_accuracy: 0.0507 - val_loss: 3.2357
Epoch 11/50
66/66                 1s 10ms/step -
accuracy: 0.1151 - loss: 3.0364 - val_accuracy: 0.0897 - val_loss: 3.1580
Epoch 12/50
66/66                 1s 9ms/step -
accuracy: 0.1343 - loss: 2.9602 - val_accuracy: 0.0912 - val_loss: 3.1718
Epoch 13/50
66/66                 1s 9ms/step -
accuracy: 0.1265 - loss: 2.9793 - val_accuracy: 0.0941 - val_loss: 3.1461
Epoch 14/50
```

```
66/66            1s 9ms/step -
accuracy: 0.1401 - loss: 2.9380 - val_accuracy: 0.0868 - val_loss: 3.1287
Epoch 15/50
66/66            1s 12ms/step -
accuracy: 0.1584 - loss: 2.8980 - val_accuracy: 0.0810 - val_loss: 3.1894
Epoch 16/50
66/66            1s 12ms/step -
accuracy: 0.1644 - loss: 2.8779 - val_accuracy: 0.1056 - val_loss: 3.1078
Epoch 17/50
66/66            1s 9ms/step -
accuracy: 0.1742 - loss: 2.8554 - val_accuracy: 0.0825 - val_loss: 3.1852
Epoch 18/50
66/66            1s 9ms/step -
accuracy: 0.1802 - loss: 2.8016 - val_accuracy: 0.0970 - val_loss: 3.1585
Epoch 19/50
66/66            1s 12ms/step -
accuracy: 0.1929 - loss: 2.7680 - val_accuracy: 0.1085 - val_loss: 3.0788
Epoch 20/50
66/66            1s 12ms/step -
accuracy: 0.2119 - loss: 2.7280 - val_accuracy: 0.1172 - val_loss: 3.0448
Epoch 21/50
66/66            1s 13ms/step -
accuracy: 0.1972 - loss: 2.7020 - val_accuracy: 0.1114 - val_loss: 3.0623
Epoch 22/50
66/66            1s 9ms/step -
accuracy: 0.1875 - loss: 2.7167 - val_accuracy: 0.1129 - val_loss: 3.0488
Epoch 23/50
66/66            1s 8ms/step -
accuracy: 0.2023 - loss: 2.6215 - val_accuracy: 0.1056 - val_loss: 3.1251
Epoch 24/50
66/66            1s 8ms/step -
accuracy: 0.2163 - loss: 2.6493 - val_accuracy: 0.1331 - val_loss: 2.9632
Epoch 25/50
66/66            1s 8ms/step -
accuracy: 0.2231 - loss: 2.6164 - val_accuracy: 0.1360 - val_loss: 3.1563
Epoch 26/50
66/66            1s 8ms/step -
accuracy: 0.2280 - loss: 2.5988 - val_accuracy: 0.1302 - val_loss: 2.9839
Epoch 27/50
66/66            1s 8ms/step -
accuracy: 0.2036 - loss: 2.6460 - val_accuracy: 0.1143 - val_loss: 3.2332
Epoch 28/50
66/66            0s 7ms/step -
accuracy: 0.2470 - loss: 2.5306 - val_accuracy: 0.1100 - val_loss: 3.1894
Epoch 29/50
66/66            0s 7ms/step -
accuracy: 0.2325 - loss: 2.5394 - val_accuracy: 0.1216 - val_loss: 3.0087
Epoch 30/50
```

```
66/66            1s 8ms/step -
accuracy: 0.2453 - loss: 2.5398 - val_accuracy: 0.1071 - val_loss: 3.0043
Epoch 31/50
66/66            1s 8ms/step -
accuracy: 0.2452 - loss: 2.5527 - val_accuracy: 0.1433 - val_loss: 2.9821
Epoch 32/50
66/66            0s 7ms/step -
accuracy: 0.2683 - loss: 2.4815 - val_accuracy: 0.1360 - val_loss: 3.0539
Epoch 33/50
66/66            0s 7ms/step -
accuracy: 0.2745 - loss: 2.4633 - val_accuracy: 0.1143 - val_loss: 3.2575
Epoch 34/50
66/66            0s 7ms/step -
accuracy: 0.2780 - loss: 2.4565 - val_accuracy: 0.1259 - val_loss: 2.9944
Epoch 35/50
66/66            1s 7ms/step -
accuracy: 0.2744 - loss: 2.4072 - val_accuracy: 0.1216 - val_loss: 3.4651
Epoch 36/50
66/66            1s 8ms/step -
accuracy: 0.2691 - loss: 2.3681 - val_accuracy: 0.1606 - val_loss: 2.9844
Epoch 37/50
66/66            0s 7ms/step -
accuracy: 0.2971 - loss: 2.3686 - val_accuracy: 0.1404 - val_loss: 2.9970
Epoch 38/50
66/66            0s 7ms/step -
accuracy: 0.2771 - loss: 2.3671 - val_accuracy: 0.1433 - val_loss: 3.0677
Epoch 39/50
66/66            1s 8ms/step -
accuracy: 0.2891 - loss: 2.3302 - val_accuracy: 0.1143 - val_loss: 3.3396
Epoch 40/50
66/66            1s 8ms/step -
accuracy: 0.2880 - loss: 2.3389 - val_accuracy: 0.1491 - val_loss: 2.9410
Epoch 41/50
66/66            1s 8ms/step -
accuracy: 0.3026 - loss: 2.3255 - val_accuracy: 0.1520 - val_loss: 2.9947
Epoch 42/50
66/66            1s 8ms/step -
accuracy: 0.3123 - loss: 2.3091 - val_accuracy: 0.1302 - val_loss: 3.2061
Epoch 43/50
66/66            1s 8ms/step -
accuracy: 0.3049 - loss: 2.3206 - val_accuracy: 0.1375 - val_loss: 3.0039
Epoch 44/50
66/66            1s 8ms/step -
accuracy: 0.3285 - loss: 2.2491 - val_accuracy: 0.1491 - val_loss: 2.9655
Epoch 45/50
66/66            1s 8ms/step -
accuracy: 0.3303 - loss: 2.2470 - val_accuracy: 0.1548 - val_loss: 2.9410
Epoch 46/50
```

```
66/66                1s 8ms/step -
accuracy: 0.3263 - loss: 2.2296 - val_accuracy: 0.1534 - val_loss: 3.0484
Epoch 47/50
66/66                1s 8ms/step -
accuracy: 0.3189 - loss: 2.2225 - val_accuracy: 0.1129 - val_loss: 3.2081
Epoch 48/50
66/66                1s 8ms/step -
accuracy: 0.3067 - loss: 2.2583 - val_accuracy: 0.1693 - val_loss: 2.8921
Epoch 49/50
66/66                1s 7ms/step -
accuracy: 0.3328 - loss: 2.1866 - val_accuracy: 0.1520 - val_loss: 3.0308
Epoch 50/50
66/66                1s 8ms/step -
accuracy: 0.3103 - loss: 2.2111 - val_accuracy: 0.1534 - val_loss: 3.0098
```

[4]:

```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                5s 217ms/step -
accuracy: 0.1527 - loss: 3.1793
Test Accuracy: 0.1634
Test Loss: 3.0588
```

[5]:

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

[6]:

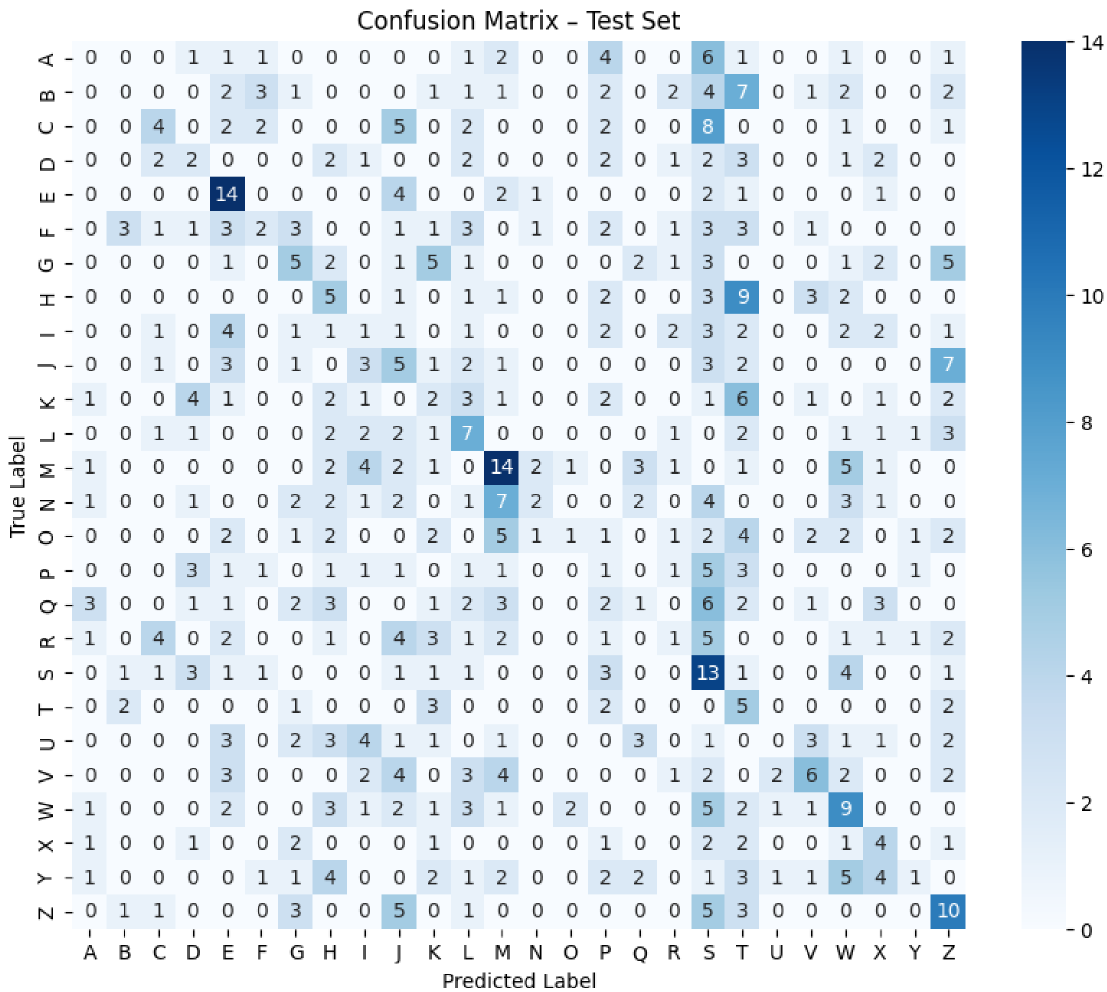
```python
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
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      0.000     0.000     0.000        19
           B      0.000     0.000     0.000        29
```

C	0.250	0.148	0.186	27
D	0.111	0.100	0.105	20
E	0.304	0.560	0.394	25
F	0.182	0.069	0.100	29
G	0.200	0.172	0.185	29
H	0.143	0.185	0.161	27
I	0.048	0.042	0.044	24
J	0.119	0.172	0.141	29
K	0.074	0.071	0.073	28
L	0.184	0.280	0.222	25
M	0.292	0.368	0.326	38
N	0.286	0.069	0.111	29
O	0.250	0.034	0.061	29
P	0.032	0.048	0.038	21
Q	0.077	0.032	0.045	31
R	0.077	0.033	0.047	30
S	0.146	0.406	0.215	32
T	0.081	0.333	0.130	15
U	0.000	0.000	0.000	26
V	0.300	0.194	0.235	31
W	0.205	0.265	0.231	34
X	0.167	0.250	0.200	16
Y	0.200	0.031	0.054	32
Z	0.227	0.345	0.274	29
accuracy			0.163	704
macro avg	0.152	0.162	0.138	704
weighted avg	0.160	0.163	0.142	704


Confusion Matrix – Test Set
True Label
Predicted Label