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 1691 files belonging to 26 classes.
Using 1015 files for training.
Using 1015 files for training.
Found 1691 files belonging to 26 classes.
Using 676 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
32/32                 10s 110ms/step -
accuracy: 0.0646 - loss: 3.3258 - val_accuracy: 0.0586 - val_loss: 3.2438
Epoch 2/50
32/32                 0s 8ms/step -
accuracy: 0.0779 - loss: 3.1777 - val_accuracy: 0.0432 - val_loss: 3.2305
Epoch 3/50
32/32                 0s 7ms/step -
accuracy: 0.0779 - loss: 3.1342 - val_accuracy: 0.0432 - val_loss: 3.2131
Epoch 4/50
32/32                 0s 7ms/step -
accuracy: 0.0853 - loss: 3.0912 - val_accuracy: 0.0432 - val_loss: 3.2172
Epoch 5/50
32/32                 0s 8ms/step -
accuracy: 0.1127 - loss: 3.0734 - val_accuracy: 0.0432 - val_loss: 3.2369
Epoch 6/50
32/32                 0s 7ms/step -
accuracy: 0.1025 - loss: 3.0758 - val_accuracy: 0.0432 - val_loss: 3.2703
Epoch 7/50
32/32                 0s 7ms/step -
accuracy: 0.0977 - loss: 3.0274 - val_accuracy: 0.0463 - val_loss: 3.2901
Epoch 8/50
32/32                 0s 8ms/step -
accuracy: 0.1109 - loss: 3.0025 - val_accuracy: 0.0648 - val_loss: 3.3376
Epoch 9/50
32/32                 0s 9ms/step -
accuracy: 0.1070 - loss: 3.0063 - val_accuracy: 0.0463 - val_loss: 3.3455
Epoch 10/50
32/32                 0s 9ms/step -
accuracy: 0.1312 - loss: 2.9960 - val_accuracy: 0.0463 - val_loss: 3.3476
Epoch 11/50
32/32                 0s 9ms/step -
accuracy: 0.1305 - loss: 2.9760 - val_accuracy: 0.0556 - val_loss: 3.3604
Epoch 12/50
32/32                 0s 8ms/step -
accuracy: 0.1403 - loss: 2.8979 - val_accuracy: 0.0432 - val_loss: 3.4240
Epoch 13/50
32/32                 0s 8ms/step -
accuracy: 0.1282 - loss: 2.9230 - val_accuracy: 0.0525 - val_loss: 3.2952
```

```
Epoch 14/50
32/32              0s 9ms/step -
accuracy: 0.1391 - loss: 2.9310 - val_accuracy: 0.0556 - val_loss: 3.2832
Epoch 15/50
32/32              0s 9ms/step -
accuracy: 0.1431 - loss: 2.8571 - val_accuracy: 0.0586 - val_loss: 3.3570
Epoch 16/50
32/32              0s 8ms/step -
accuracy: 0.1266 - loss: 2.8947 - val_accuracy: 0.0432 - val_loss: 3.3887
Epoch 17/50
32/32              0s 8ms/step -
accuracy: 0.1579 - loss: 2.8416 - val_accuracy: 0.0648 - val_loss: 3.3575
Epoch 18/50
32/32              0s 8ms/step -
accuracy: 0.1555 - loss: 2.8437 - val_accuracy: 0.0802 - val_loss: 3.2834
Epoch 19/50
32/32              0s 9ms/step -
accuracy: 0.1573 - loss: 2.8043 - val_accuracy: 0.0833 - val_loss: 3.3043
Epoch 20/50
32/32              0s 8ms/step -
accuracy: 0.1690 - loss: 2.7500 - val_accuracy: 0.0772 - val_loss: 3.2085
Epoch 21/50
32/32              0s 8ms/step -
accuracy: 0.1864 - loss: 2.7843 - val_accuracy: 0.0679 - val_loss: 3.1802
Epoch 22/50
32/32              0s 8ms/step -
accuracy: 0.1929 - loss: 2.7064 - val_accuracy: 0.0679 - val_loss: 3.1480
Epoch 23/50
32/32              0s 9ms/step -
accuracy: 0.1874 - loss: 2.7182 - val_accuracy: 0.0802 - val_loss: 3.1285
Epoch 24/50
32/32              0s 8ms/step -
accuracy: 0.1820 - loss: 2.6816 - val_accuracy: 0.0833 - val_loss: 3.2085
Epoch 25/50
32/32              0s 8ms/step -
accuracy: 0.2284 - loss: 2.6256 - val_accuracy: 0.1173 - val_loss: 3.1260
Epoch 26/50
32/32              0s 8ms/step -
accuracy: 0.1869 - loss: 2.6635 - val_accuracy: 0.1049 - val_loss: 3.1225
Epoch 27/50
32/32              0s 7ms/step -
accuracy: 0.2261 - loss: 2.5858 - val_accuracy: 0.0833 - val_loss: 3.1284
Epoch 28/50
32/32              0s 7ms/step -
accuracy: 0.2214 - loss: 2.6058 - val_accuracy: 0.0741 - val_loss: 3.4201
Epoch 29/50
32/32              0s 7ms/step -
accuracy: 0.2314 - loss: 2.5598 - val_accuracy: 0.1049 - val_loss: 3.1550
```

```
Epoch 30/50
32/32             0s 7ms/step -
accuracy: 0.2445 - loss: 2.5720 - val_accuracy: 0.1019 - val_loss: 3.2403
Epoch 31/50
32/32             0s 7ms/step -
accuracy: 0.2229 - loss: 2.5674 - val_accuracy: 0.1019 - val_loss: 3.1437
Epoch 32/50
32/32             0s 7ms/step -
accuracy: 0.2751 - loss: 2.5286 - val_accuracy: 0.1235 - val_loss: 3.1262
Epoch 33/50
32/32             0s 7ms/step -
accuracy: 0.2657 - loss: 2.4702 - val_accuracy: 0.1111 - val_loss: 3.2315
Epoch 34/50
32/32             0s 7ms/step -
accuracy: 0.2206 - loss: 2.5265 - val_accuracy: 0.0895 - val_loss: 3.1556
Epoch 35/50
32/32             0s 7ms/step -
accuracy: 0.2655 - loss: 2.4060 - val_accuracy: 0.1481 - val_loss: 3.1818
Epoch 36/50
32/32             0s 7ms/step -
accuracy: 0.2655 - loss: 2.4504 - val_accuracy: 0.1204 - val_loss: 3.0639
Epoch 37/50
32/32             0s 7ms/step -
accuracy: 0.2426 - loss: 2.4265 - val_accuracy: 0.0957 - val_loss: 3.4497
Epoch 38/50
32/32             0s 7ms/step -
accuracy: 0.2816 - loss: 2.3985 - val_accuracy: 0.1235 - val_loss: 3.0809
Epoch 39/50
32/32             0s 7ms/step -
accuracy: 0.2877 - loss: 2.3591 - val_accuracy: 0.1235 - val_loss: 3.1293
Epoch 40/50
32/32             0s 7ms/step -
accuracy: 0.2752 - loss: 2.3650 - val_accuracy: 0.0895 - val_loss: 3.2817
Epoch 41/50
32/32             0s 7ms/step -
accuracy: 0.2780 - loss: 2.3536 - val_accuracy: 0.1265 - val_loss: 3.1639
Epoch 42/50
32/32             0s 7ms/step -
accuracy: 0.2833 - loss: 2.3562 - val_accuracy: 0.1204 - val_loss: 3.1205
Epoch 43/50
32/32             0s 7ms/step -
accuracy: 0.3205 - loss: 2.3018 - val_accuracy: 0.1327 - val_loss: 3.1374
Epoch 44/50
32/32             0s 8ms/step -
accuracy: 0.3468 - loss: 2.2001 - val_accuracy: 0.1265 - val_loss: 3.2206
Epoch 45/50
32/32             0s 7ms/step -
accuracy: 0.3397 - loss: 2.2323 - val_accuracy: 0.1049 - val_loss: 3.2535
```

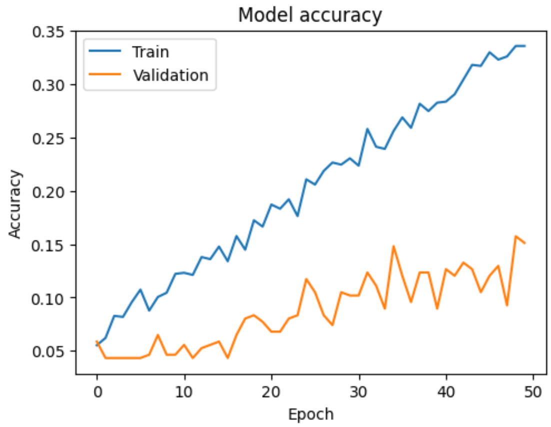
```
Epoch 46/50
32/32            0s 8ms/step -
accuracy: 0.3421 - loss: 2.2213 - val_accuracy: 0.1204 - val_loss: 3.2052
Epoch 47/50
32/32            0s 7ms/step -
accuracy: 0.3281 - loss: 2.2188 - val_accuracy: 0.1296 - val_loss: 3.1699
Epoch 48/50
32/32            0s 8ms/step -
accuracy: 0.3459 - loss: 2.1450 - val_accuracy: 0.0926 - val_loss: 3.3231
Epoch 49/50
32/32            0s 8ms/step -
accuracy: 0.3374 - loss: 2.1833 - val_accuracy: 0.1574 - val_loss: 3.1307
Epoch 50/50
32/32            0s 8ms/step -
accuracy: 0.3332 - loss: 2.1773 - val_accuracy: 0.1512 - val_loss: 3.2231
```

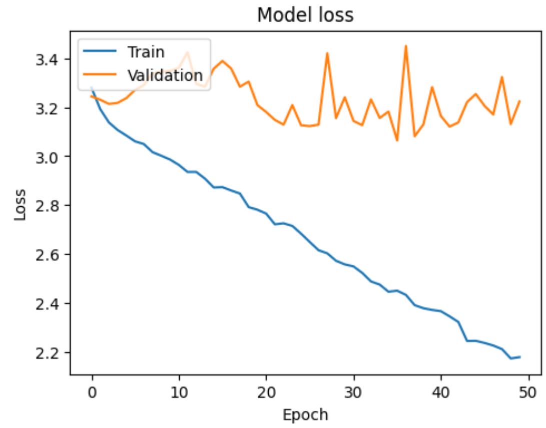
[4]:
```
test_loss, test_accuracy = model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
11/11            3s 269ms/step -
accuracy: 0.1087 - loss: 3.3143
Test Accuracy: 0.1108
Test Loss: 3.3241
```

[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()
```

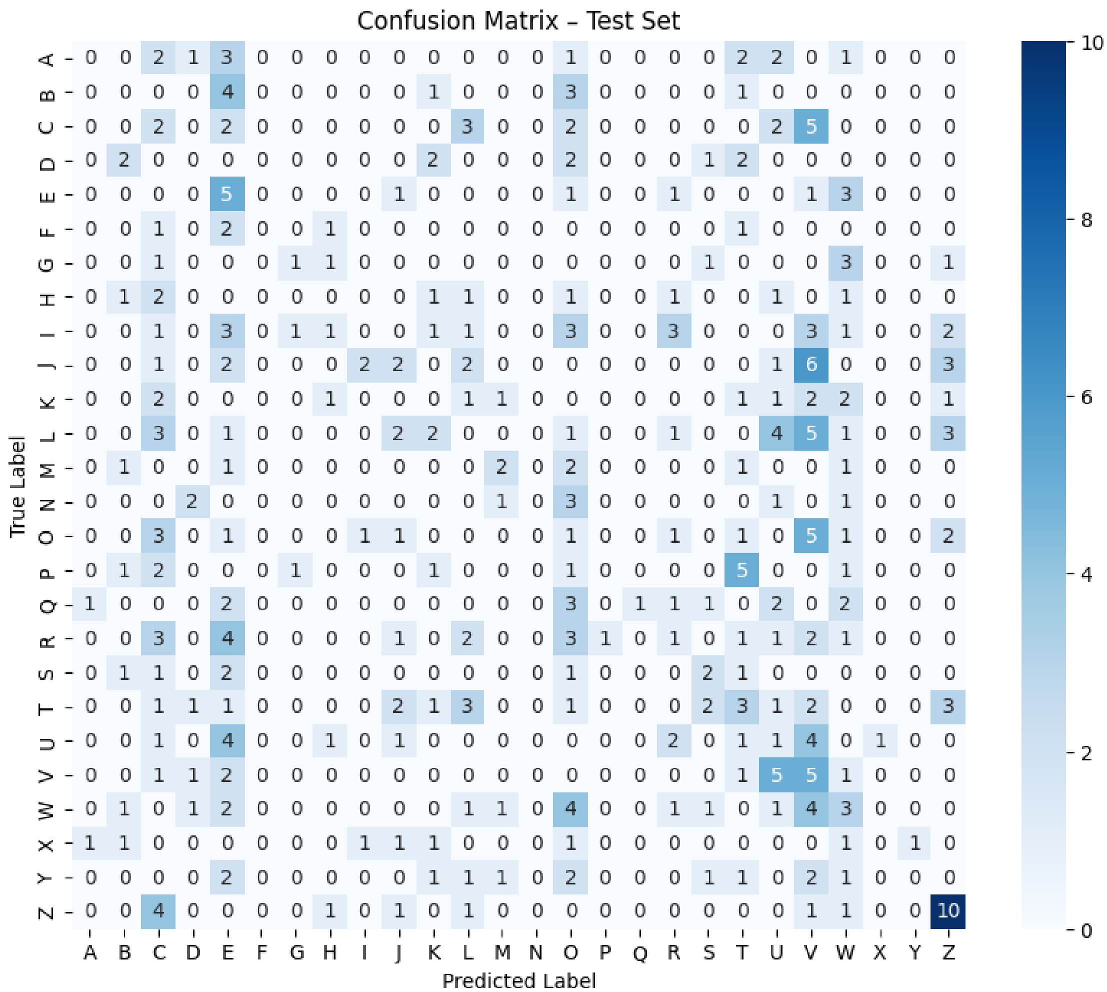
```
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
```

```
  _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      0.000     0.000     0.000        12
           B      0.000     0.000     0.000         9
           C      0.065     0.125     0.085        16
           D      0.000     0.000     0.000         9
           E      0.116     0.417     0.182        12
           F      0.000     0.000     0.000         5
           G      0.333     0.125     0.182         8
           H      0.000     0.000     0.000         9
           I      0.000     0.000     0.000        20
           J      0.167     0.105     0.129        19
           K      0.000     0.000     0.000        12
           L      0.000     0.000     0.000        23
           M      0.333     0.250     0.286         8
           N      0.000     0.000     0.000         8
           O      0.028     0.059     0.038        17
           P      0.000     0.000     0.000        12
           Q      1.000     0.077     0.143        13
           R      0.083     0.050     0.062        20
           S      0.222     0.250     0.235         8
           T      0.136     0.143     0.140        21
           U      0.043     0.062     0.051        16
           V      0.106     0.312     0.159        16
           W      0.115     0.150     0.130        20
           X      0.000     0.000     0.000         8
           Y      0.000     0.000     0.000        12
           Z      0.400     0.526     0.455        19

    accuracy                          0.111       352
   macro avg      0.121     0.102     0.088       352
weighted avg      0.122     0.111     0.093       352
```


Confusion Matrix – Test Set
True Label
Predicted Label