COMPARISON_MediaPipe+CNN

June 21, 2025

Paper Reference : https://j-innovative.org/index.php/Innovative/article/download/15199/10372/26113

[1]:
```
import os
from modules.SignLanguageProcessor import load_and_preprocess_data,parse_frame
```

[2]:
```
ROOT_PATH = ''
sequences,labels,label_map = load_and_preprocess_data(os.path.
 ↪join(ROOT_PATH,'data'))
```

[3]:
```
num_classes = len(label_map)
```

[4]:
```
len(labels)
```

[4]:
```
3488
```

[5]:
```
sequences.shape
```

[5]:
```
(3488, 3, 61, 3)
```

[6]:
```
from sklearn.model_selection import train_test_split

X_train, X_temp, y_train, y_temp = train_test_split(
    sequences, labels, test_size=0.4, stratify=labels, random_state=42
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42
)
```

[7]:
```
import numpy as np
def normalize_landmark_data(X):
    """
    Normalize the landmark features (x, y) to have zero mean and unit variance␣
 ↪across the training set.
    Assumes X shape is (N, F, L, T), where F=3 (x, y, vis).
    """
    X = X.copy()
    # Flatten across all samples, landmarks, and frames
```

```
    x_vals = X[:, 0, :, :].flatten()
    y_vals = X[:, 1, :, :].flatten()

    # Compute mean and std
    x_mean, x_std = np.mean(x_vals), np.std(x_vals)
    y_mean, y_std = np.mean(y_vals), np.std(y_vals)

    # Normalize
    X[:, 0, :, :] = (X[:, 0, :, :] - x_mean) / x_std
    X[:, 1, :, :] = (X[:, 1, :, :] - y_mean) / y_std

    return X, (x_mean, x_std), (y_mean, y_std)

def apply_normalization(X, x_mean, x_std, y_mean, y_std):
    X = X.copy()
    X[:, 0, :, :] = (X[:, 0, :, :] - x_mean) / x_std
    X[:, 1, :, :] = (X[:, 1, :, :] - y_mean) / y_std
    return X
```

[8]:
```
def reshape_frames_for_cnn(X, y):
    """
    Reshape a dataset of (N, F, L, T) into (N*T, L, F, 1) for Conv2D,
    where each frame becomes its own sample.

    Parameters:
    - X: np.ndarray of shape (N, F, L, T)
    - y: np.ndarray of shape (N,)

    Returns:
    - reshaped_X: np.ndarray of shape (N*T, L, F, 1)
    - reshaped_y: np.ndarray of shape (N*T,)
    """
    reshaped_X = []
    reshaped_y = []

    for sample, label in zip(X, y):
        T = sample.shape[-1]
        for t in range(T):
            frame = sample[:, :, t].T[..., np.newaxis]
            reshaped_X.append(frame)
            reshaped_y.append(label)

    reshaped_X = np.array(reshaped_X)
    reshaped_y = np.array(reshaped_y)
    return reshaped_X, reshaped_y
```

[9]:
```
X_train_norm, (x_mean, x_std), (y_mean, y_std) =␣
 ↪normalize_landmark_data(X_train)
X_val_norm  = apply_normalization(X_val, x_mean, x_std, y_mean, y_std)
X_test_norm = apply_normalization(X_test, x_mean, x_std, y_mean, y_std)

X_train_cnn, y_train_cnn = reshape_frames_for_cnn(X_train_norm, y_train)
X_val_cnn, y_val_cnn     = reshape_frames_for_cnn(X_val_norm, y_val)
X_test_cnn, y_test_cnn   = reshape_frames_for_cnn(X_test_norm, y_test)

print(X_train_cnn.shape)
print(y_train_cnn.shape)
```

```
(6276, 61, 3, 1)
(6276,)
```

[10]:
```
input_shape = X_train_cnn.shape[1:]
print(input_shape)
```

```
(61, 3, 1)
```

[11]:
```
import tensorflow as tf

train_ds = tf.data.Dataset.from_tensor_slices((X_train_cnn, y_train_cnn))
train_ds = train_ds.shuffle(buffer_size=1000).batch(64).prefetch(tf.data.
 ↪AUTOTUNE)

val_ds = tf.data.Dataset.from_tensor_slices((X_val_cnn, y_val_cnn))
val_ds = val_ds.batch(64).prefetch(tf.data.AUTOTUNE)

test_ds = tf.data.Dataset.from_tensor_slices((X_test_cnn, y_test_cnn))
test_ds = test_ds.batch(64).prefetch(tf.data.AUTOTUNE)
```

[12]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten,␣
 ↪Dense, BatchNormalization,Input

cnn_model = Sequential([
    Input(input_shape),
    Conv2D(32, (3, 2), activation='relu', padding='same'),
    MaxPooling2D((2, 1)),
    Dropout(0.25),
    Conv2D(64, (3, 2), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 1)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(num_classes, activation='softmax')
```

```
])
cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[13]:
```
history = cnn_model.fit(train_ds,validation_data=val_ds, epochs=50,␣
 ↪batch_size=64)
```

```
Epoch 1/50
99/99             2s 10ms/step -
accuracy: 0.0621 - loss: 3.2028 - val_accuracy: 0.0755 - val_loss: 3.0281
Epoch 2/50
99/99             1s 8ms/step -
accuracy: 0.0837 - loss: 3.0460 - val_accuracy: 0.1442 - val_loss: 2.9196
Epoch 3/50
99/99             1s 7ms/step -
accuracy: 0.1141 - loss: 2.9342 - val_accuracy: 0.1925 - val_loss: 2.7892
Epoch 4/50
99/99             1s 8ms/step -
accuracy: 0.1579 - loss: 2.8334 - val_accuracy: 0.2311 - val_loss: 2.6526
Epoch 5/50
99/99             1s 7ms/step -
accuracy: 0.2037 - loss: 2.7012 - val_accuracy: 0.2660 - val_loss: 2.5360
Epoch 6/50
99/99             1s 7ms/step -
accuracy: 0.2149 - loss: 2.6085 - val_accuracy: 0.2865 - val_loss: 2.4274
Epoch 7/50
99/99             1s 8ms/step -
accuracy: 0.2623 - loss: 2.4799 - val_accuracy: 0.3075 - val_loss: 2.3120
Epoch 8/50
99/99             1s 8ms/step -
accuracy: 0.2798 - loss: 2.4115 - val_accuracy: 0.3257 - val_loss: 2.2450
Epoch 9/50
99/99             1s 8ms/step -
accuracy: 0.2861 - loss: 2.3507 - val_accuracy: 0.3472 - val_loss: 2.2033
Epoch 10/50
99/99             1s 8ms/step -
accuracy: 0.3213 - loss: 2.2603 - val_accuracy: 0.3701 - val_loss: 2.1223
Epoch 11/50
99/99             1s 8ms/step -
accuracy: 0.3305 - loss: 2.1955 - val_accuracy: 0.3691 - val_loss: 2.0914
Epoch 12/50
99/99             1s 8ms/step -
accuracy: 0.3553 - loss: 2.1311 - val_accuracy: 0.3940 - val_loss: 2.0411
Epoch 13/50
99/99             1s 8ms/step -
accuracy: 0.3548 - loss: 2.0990 - val_accuracy: 0.4026 - val_loss: 2.0022
Epoch 14/50
99/99             1s 7ms/step -
```

```
accuracy: 0.3699 - loss: 2.0700 - val_accuracy: 0.3992 - val_loss: 1.9866
Epoch 15/50
99/99             1s 7ms/step -
accuracy: 0.3717 - loss: 2.0297 - val_accuracy: 0.4136 - val_loss: 1.9672
Epoch 16/50
99/99             1s 7ms/step -
accuracy: 0.3809 - loss: 2.0186 - val_accuracy: 0.4145 - val_loss: 1.9575
Epoch 17/50
99/99             1s 7ms/step -
accuracy: 0.3963 - loss: 1.9619 - val_accuracy: 0.4250 - val_loss: 1.9298
Epoch 18/50
99/99             1s 7ms/step -
accuracy: 0.4087 - loss: 1.9288 - val_accuracy: 0.4288 - val_loss: 1.9290
Epoch 19/50
99/99             1s 7ms/step -
accuracy: 0.4094 - loss: 1.9127 - val_accuracy: 0.4269 - val_loss: 1.9130
Epoch 20/50
99/99             1s 7ms/step -
accuracy: 0.4147 - loss: 1.8970 - val_accuracy: 0.4422 - val_loss: 1.8787
Epoch 21/50
99/99             1s 7ms/step -
accuracy: 0.4035 - loss: 1.9047 - val_accuracy: 0.4403 - val_loss: 1.8753
Epoch 22/50
99/99             1s 7ms/step -
accuracy: 0.4225 - loss: 1.8598 - val_accuracy: 0.4436 - val_loss: 1.8774
Epoch 23/50
99/99             1s 7ms/step -
accuracy: 0.4101 - loss: 1.9026 - val_accuracy: 0.4475 - val_loss: 1.8698
Epoch 24/50
99/99             1s 7ms/step -
accuracy: 0.4127 - loss: 1.8617 - val_accuracy: 0.4394 - val_loss: 1.8733
Epoch 25/50
99/99             1s 7ms/step -
accuracy: 0.4363 - loss: 1.8366 - val_accuracy: 0.4436 - val_loss: 1.8658
Epoch 26/50
99/99             1s 7ms/step -
accuracy: 0.4459 - loss: 1.8088 - val_accuracy: 0.4484 - val_loss: 1.8531
Epoch 27/50
99/99             1s 7ms/step -
accuracy: 0.4384 - loss: 1.7893 - val_accuracy: 0.4589 - val_loss: 1.8344
Epoch 28/50
99/99             1s 7ms/step -
accuracy: 0.4246 - loss: 1.8089 - val_accuracy: 0.4604 - val_loss: 1.8209
Epoch 29/50
99/99             1s 7ms/step -
accuracy: 0.4400 - loss: 1.7718 - val_accuracy: 0.4585 - val_loss: 1.8223
Epoch 30/50
99/99             1s 7ms/step -
```

```
accuracy: 0.4454 - loss: 1.7512 - val_accuracy: 0.4565 - val_loss: 1.8060
Epoch 31/50
99/99             1s 7ms/step -
accuracy: 0.4556 - loss: 1.7272 - val_accuracy: 0.4618 - val_loss: 1.8175
Epoch 32/50
99/99             1s 7ms/step -
accuracy: 0.4457 - loss: 1.7570 - val_accuracy: 0.4594 - val_loss: 1.8248
Epoch 33/50
99/99             1s 7ms/step -
accuracy: 0.4594 - loss: 1.7307 - val_accuracy: 0.4642 - val_loss: 1.8099
Epoch 34/50
99/99             1s 7ms/step -
accuracy: 0.4576 - loss: 1.7191 - val_accuracy: 0.4542 - val_loss: 1.8274
Epoch 35/50
99/99             1s 7ms/step -
accuracy: 0.4633 - loss: 1.7097 - val_accuracy: 0.4737 - val_loss: 1.8015
Epoch 36/50
99/99             1s 7ms/step -
accuracy: 0.4643 - loss: 1.6996 - val_accuracy: 0.4680 - val_loss: 1.8017
Epoch 37/50
99/99             1s 7ms/step -
accuracy: 0.4675 - loss: 1.7059 - val_accuracy: 0.4704 - val_loss: 1.7891
Epoch 38/50
99/99             1s 7ms/step -
accuracy: 0.4697 - loss: 1.6859 - val_accuracy: 0.4675 - val_loss: 1.7920
Epoch 39/50
99/99             1s 7ms/step -
accuracy: 0.4575 - loss: 1.7230 - val_accuracy: 0.4737 - val_loss: 1.7919
Epoch 40/50
99/99             1s 7ms/step -
accuracy: 0.4653 - loss: 1.6982 - val_accuracy: 0.4737 - val_loss: 1.7900
Epoch 41/50
99/99             1s 7ms/step -
accuracy: 0.4747 - loss: 1.6608 - val_accuracy: 0.4628 - val_loss: 1.7955
Epoch 42/50
99/99             1s 7ms/step -
accuracy: 0.4713 - loss: 1.6927 - val_accuracy: 0.4713 - val_loss: 1.7978
Epoch 43/50
99/99             1s 7ms/step -
accuracy: 0.4668 - loss: 1.6666 - val_accuracy: 0.4685 - val_loss: 1.7960
Epoch 44/50
99/99             1s 7ms/step -
accuracy: 0.4817 - loss: 1.6413 - val_accuracy: 0.4694 - val_loss: 1.7832
Epoch 45/50
99/99             1s 7ms/step -
accuracy: 0.4804 - loss: 1.6477 - val_accuracy: 0.4723 - val_loss: 1.7977
Epoch 46/50
99/99             1s 8ms/step -
```

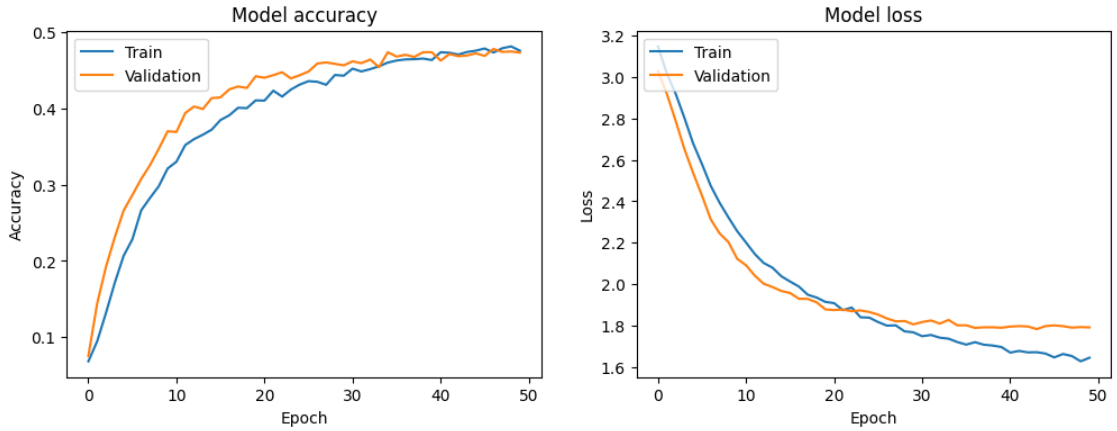
```
accuracy: 0.4801 - loss: 1.6422 - val_accuracy: 0.4690 - val_loss: 1.8012
Epoch 47/50
99/99             1s 7ms/step -
accuracy: 0.4695 - loss: 1.6629 - val_accuracy: 0.4780 - val_loss: 1.7972
Epoch 48/50
99/99             1s 7ms/step -
accuracy: 0.4800 - loss: 1.6365 - val_accuracy: 0.4742 - val_loss: 1.7905
Epoch 49/50
99/99             1s 7ms/step -
accuracy: 0.4832 - loss: 1.6430 - val_accuracy: 0.4747 - val_loss: 1.7929
Epoch 50/50
99/99             1s 7ms/step -
accuracy: 0.4812 - loss: 1.6256 - val_accuracy: 0.4733 - val_loss: 1.7915
```

[14]:
```
test_loss, test_accuracy = cnn_model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
33/33             0s 3ms/step -
accuracy: 0.4545 - loss: 1.8342
Test Accuracy: 0.4408
Test Loss: 1.8524
```

[15]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
```

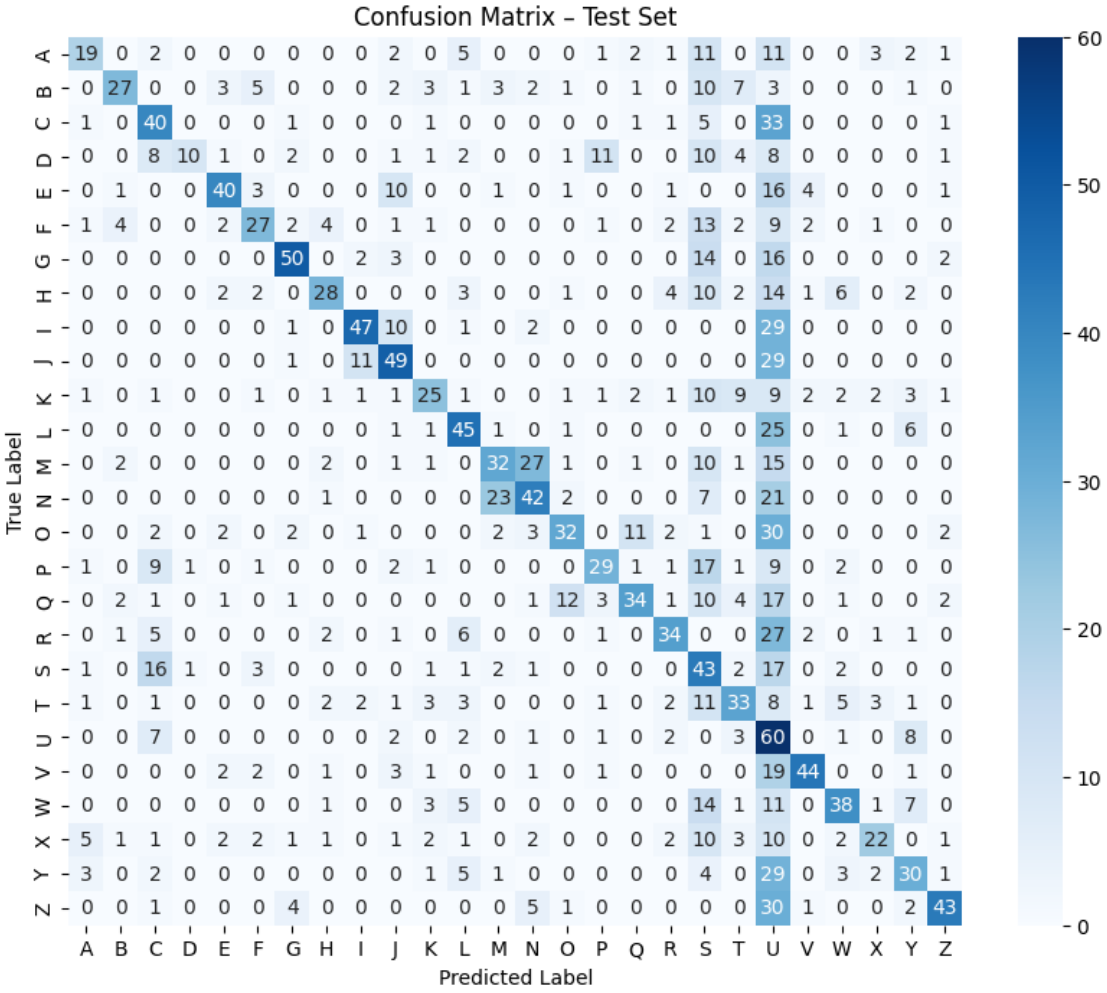
[16]:
```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
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()
```



[17]:
```
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 = cnn_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      0.576     0.317     0.409        60
           B      0.711     0.391     0.505        69
```

C	0.417	0.476	0.444	84
D	0.833	0.167	0.278	60
E	0.727	0.513	0.602	78
F	0.587	0.375	0.458	72
G	0.769	0.575	0.658	87
H	0.651	0.373	0.475	75
I	0.734	0.522	0.610	90
J	0.538	0.544	0.541	90
K	0.556	0.333	0.417	75
L	0.556	0.556	0.556	81
M	0.492	0.344	0.405	93
N	0.483	0.438	0.459	96
O	0.593	0.356	0.444	90
P	0.580	0.387	0.464	75
Q	0.642	0.378	0.476	90
R	0.630	0.420	0.504	81
S	0.205	0.478	0.287	90
T	0.458	0.423	0.440	78
U	0.119	0.690	0.203	87
V	0.772	0.587	0.667	75
W	0.603	0.469	0.528	81
X	0.629	0.319	0.423	69
Y	0.469	0.370	0.414	81
Z	0.768	0.494	0.601	87
accuracy			0.441	2094
macro avg	0.581	0.434	0.472	2094
weighted avg	0.574	0.441	0.474	2094


Confusion Matrix – Test Set
True Label
Predicted Label
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
60
50
40
30
20
10
0