COMPARISON_MediaPipe+CNN

June 21, 2025

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

[35]:
```
import os
from modules.SignLanguageProcessor import load_and_preprocess_data,parse_frame
```

[36]:
```
ROOT_PATH = ''
sequences,labels,label_map = load_and_preprocess_data(os.path.
 ↪join(ROOT_PATH,'data'))
```

[37]:
```
num_classes = len(label_map)
```

[38]:
```
len(labels)
```

[38]:
```
1691
```

[39]:
```
sequences.shape
```

[39]:
```
(1691, 3, 61, 3)
```

[40]:
```
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
)
```

[41]:
```
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
```

[42]:
```
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
```

[43]:
```
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)
```

```
(3042, 61, 3, 1)
(3042,)
```

[44]:
```
input_shape = X_train_cnn.shape[1:]
print(input_shape)
```

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

[45]:
```
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)
```

[46]:
```
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'])
```

[47]:
```
history = cnn_model.fit(train_ds,validation_data=val_ds, epochs=50,␣
 ↪batch_size=64)
```

```
Epoch 1/50
48/48             2s 11ms/step -
accuracy: 0.0904 - loss: 3.0575 - val_accuracy: 0.1381 - val_loss: 2.7668
Epoch 2/50
48/48             0s 8ms/step -
accuracy: 0.0911 - loss: 2.7957 - val_accuracy: 0.1538 - val_loss: 2.7181
Epoch 3/50
48/48             0s 8ms/step -
accuracy: 0.1092 - loss: 2.7262 - val_accuracy: 0.1499 - val_loss: 2.6980
Epoch 4/50
48/48             0s 8ms/step -
accuracy: 0.1324 - loss: 2.6836 - val_accuracy: 0.2022 - val_loss: 2.6165
Epoch 5/50
48/48             0s 8ms/step -
accuracy: 0.1533 - loss: 2.6193 - val_accuracy: 0.2327 - val_loss: 2.5446
Epoch 6/50
48/48             0s 8ms/step -
accuracy: 0.1966 - loss: 2.5627 - val_accuracy: 0.2367 - val_loss: 2.4469
Epoch 7/50
48/48             0s 8ms/step -
accuracy: 0.2477 - loss: 2.4422 - val_accuracy: 0.2840 - val_loss: 2.3435
Epoch 8/50
48/48             0s 8ms/step -
accuracy: 0.2480 - loss: 2.3828 - val_accuracy: 0.3166 - val_loss: 2.2589
Epoch 9/50
48/48             0s 8ms/step -
accuracy: 0.2697 - loss: 2.3122 - val_accuracy: 0.3018 - val_loss: 2.2201
Epoch 10/50
48/48             0s 8ms/step -
accuracy: 0.3220 - loss: 2.2122 - val_accuracy: 0.3422 - val_loss: 2.1451
Epoch 11/50
48/48             0s 8ms/step -
accuracy: 0.3103 - loss: 2.1731 - val_accuracy: 0.3698 - val_loss: 2.0923
Epoch 12/50
48/48             0s 8ms/step -
accuracy: 0.3526 - loss: 2.0696 - val_accuracy: 0.3807 - val_loss: 2.0433
Epoch 13/50
48/48             0s 8ms/step -
accuracy: 0.3572 - loss: 2.0013 - val_accuracy: 0.3964 - val_loss: 1.9966
Epoch 14/50
48/48             0s 8ms/step -
```

```
accuracy: 0.3872 - loss: 1.9384 - val_accuracy: 0.4211 - val_loss: 1.9546
Epoch 15/50
48/48             0s 8ms/step -
accuracy: 0.3749 - loss: 1.9495 - val_accuracy: 0.4280 - val_loss: 1.9226
Epoch 16/50
48/48             0s 9ms/step -
accuracy: 0.4076 - loss: 1.8710 - val_accuracy: 0.4250 - val_loss: 1.8884
Epoch 17/50
48/48             0s 8ms/step -
accuracy: 0.4109 - loss: 1.8306 - val_accuracy: 0.4438 - val_loss: 1.8472
Epoch 18/50
48/48             0s 8ms/step -
accuracy: 0.4400 - loss: 1.7675 - val_accuracy: 0.4497 - val_loss: 1.8523
Epoch 19/50
48/48             0s 9ms/step -
accuracy: 0.4424 - loss: 1.7634 - val_accuracy: 0.4576 - val_loss: 1.8154
Epoch 20/50
48/48             0s 9ms/step -
accuracy: 0.4333 - loss: 1.7657 - val_accuracy: 0.4586 - val_loss: 1.7924
Epoch 21/50
48/48             0s 9ms/step -
accuracy: 0.4505 - loss: 1.7105 - val_accuracy: 0.4803 - val_loss: 1.7606
Epoch 22/50
48/48             0s 9ms/step -
accuracy: 0.4767 - loss: 1.6456 - val_accuracy: 0.4684 - val_loss: 1.7597
Epoch 23/50
48/48             0s 9ms/step -
accuracy: 0.4670 - loss: 1.6707 - val_accuracy: 0.4901 - val_loss: 1.7343
Epoch 24/50
48/48             0s 8ms/step -
accuracy: 0.4698 - loss: 1.6432 - val_accuracy: 0.4813 - val_loss: 1.7407
Epoch 25/50
48/48             0s 8ms/step -
accuracy: 0.4816 - loss: 1.6549 - val_accuracy: 0.4862 - val_loss: 1.7311
Epoch 26/50
48/48             0s 9ms/step -
accuracy: 0.4777 - loss: 1.6018 - val_accuracy: 0.4941 - val_loss: 1.7282
Epoch 27/50
48/48             0s 8ms/step -
accuracy: 0.5058 - loss: 1.5661 - val_accuracy: 0.4872 - val_loss: 1.7241
Epoch 28/50
48/48             0s 9ms/step -
accuracy: 0.4843 - loss: 1.5462 - val_accuracy: 0.4990 - val_loss: 1.7084
Epoch 29/50
48/48             0s 8ms/step -
accuracy: 0.5160 - loss: 1.5168 - val_accuracy: 0.4931 - val_loss: 1.7207
Epoch 30/50
48/48             0s 8ms/step -
```

```
accuracy: 0.5271 - loss: 1.4818 - val_accuracy: 0.4961 - val_loss: 1.6864
Epoch 31/50
48/48             0s 9ms/step -
accuracy: 0.5296 - loss: 1.4856 - val_accuracy: 0.5000 - val_loss: 1.6934
Epoch 32/50
48/48             0s 8ms/step -
accuracy: 0.5136 - loss: 1.4916 - val_accuracy: 0.5128 - val_loss: 1.6812
Epoch 33/50
48/48             0s 9ms/step -
accuracy: 0.5092 - loss: 1.5041 - val_accuracy: 0.5197 - val_loss: 1.6661
Epoch 34/50
48/48             0s 8ms/step -
accuracy: 0.5243 - loss: 1.4503 - val_accuracy: 0.5168 - val_loss: 1.6876
Epoch 35/50
48/48             0s 8ms/step -
accuracy: 0.5279 - loss: 1.4485 - val_accuracy: 0.5227 - val_loss: 1.6911
Epoch 36/50
48/48             0s 8ms/step -
accuracy: 0.5521 - loss: 1.4126 - val_accuracy: 0.5148 - val_loss: 1.6823
Epoch 37/50
48/48             0s 8ms/step -
accuracy: 0.5213 - loss: 1.4278 - val_accuracy: 0.5118 - val_loss: 1.6950
Epoch 38/50
48/48             0s 8ms/step -
accuracy: 0.5472 - loss: 1.4155 - val_accuracy: 0.5128 - val_loss: 1.6960
Epoch 39/50
48/48             0s 8ms/step -
accuracy: 0.5369 - loss: 1.3905 - val_accuracy: 0.5217 - val_loss: 1.6967
Epoch 40/50
48/48             0s 8ms/step -
accuracy: 0.5492 - loss: 1.3782 - val_accuracy: 0.5158 - val_loss: 1.6559
Epoch 41/50
48/48             0s 9ms/step -
accuracy: 0.5418 - loss: 1.3778 - val_accuracy: 0.5345 - val_loss: 1.6655
Epoch 42/50
48/48             0s 8ms/step -
accuracy: 0.5558 - loss: 1.3814 - val_accuracy: 0.5207 - val_loss: 1.6759
Epoch 43/50
48/48             0s 8ms/step -
accuracy: 0.5388 - loss: 1.4072 - val_accuracy: 0.5128 - val_loss: 1.7150
Epoch 44/50
48/48             0s 8ms/step -
accuracy: 0.5573 - loss: 1.3425 - val_accuracy: 0.5148 - val_loss: 1.6906
Epoch 45/50
48/48             0s 8ms/step -
accuracy: 0.5417 - loss: 1.3707 - val_accuracy: 0.5187 - val_loss: 1.7115
Epoch 46/50
48/48             0s 8ms/step -
```

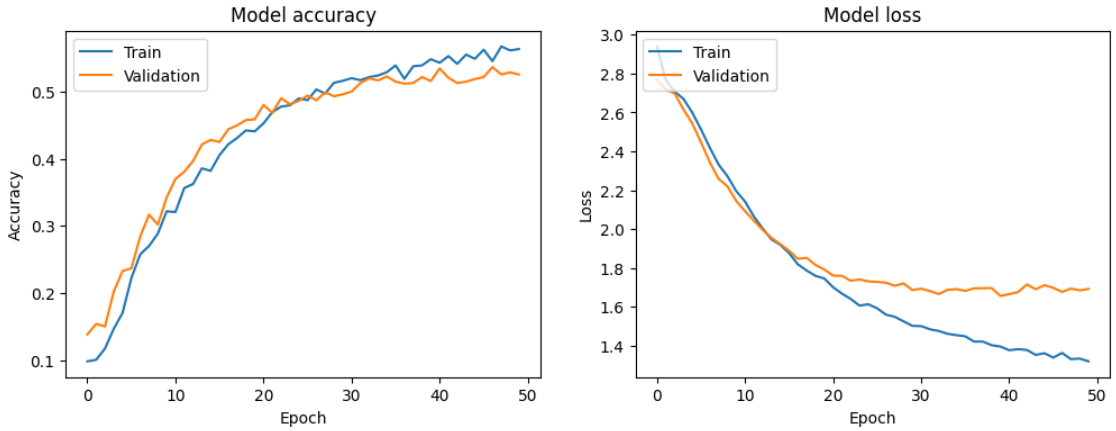
```
accuracy: 0.5710 - loss: 1.3095 - val_accuracy: 0.5217 - val_loss: 1.6977
Epoch 47/50
48/48             0s 8ms/step -
accuracy: 0.5454 - loss: 1.3636 - val_accuracy: 0.5365 - val_loss: 1.6770
Epoch 48/50
48/48             0s 8ms/step -
accuracy: 0.5668 - loss: 1.3234 - val_accuracy: 0.5256 - val_loss: 1.6938
Epoch 49/50
48/48             0s 8ms/step -
accuracy: 0.5531 - loss: 1.3517 - val_accuracy: 0.5286 - val_loss: 1.6847
Epoch 50/50
48/48             0s 8ms/step -
accuracy: 0.5706 - loss: 1.3118 - val_accuracy: 0.5256 - val_loss: 1.6922
```

[48]:
```
test_loss, test_accuracy = cnn_model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
16/16             0s 4ms/step -
accuracy: 0.5130 - loss: 1.6789
Test Accuracy: 0.5241
Test Loss: 1.6338
```

[49]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
```

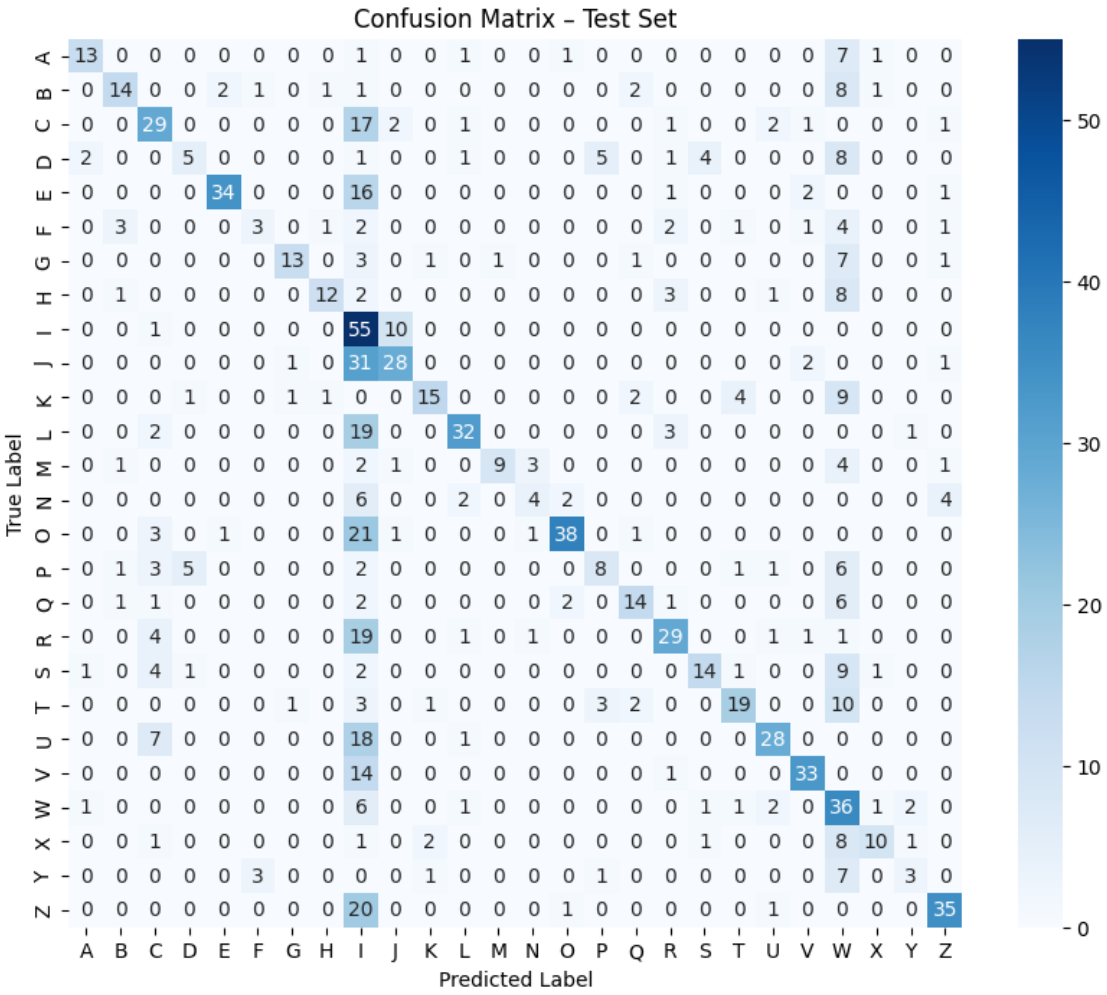
[50]:
```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
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()
```



[51]:
```
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      0.765     0.542     0.634        24
           B      0.667     0.467     0.549        30
```

C	0.527	0.537	0.532	54
D	0.417	0.185	0.256	27
E	0.919	0.630	0.747	54
F	0.429	0.167	0.240	18
G	0.812	0.481	0.605	27
H	0.800	0.444	0.571	27
I	0.208	0.833	0.333	66
J	0.667	0.444	0.533	63
K	0.750	0.455	0.566	33
L	0.800	0.561	0.660	57
M	0.900	0.429	0.581	21
N	0.444	0.222	0.296	18
O	0.864	0.576	0.691	66
P	0.471	0.296	0.364	27
Q	0.636	0.519	0.571	27
R	0.690	0.509	0.586	57
S	0.700	0.424	0.528	33
T	0.704	0.487	0.576	39
U	0.778	0.519	0.622	54
V	0.825	0.688	0.750	48
W	0.261	0.706	0.381	51
X	0.714	0.417	0.526	24
Y	0.429	0.200	0.273	15
Z	0.778	0.614	0.686	57
accuracy			0.524	1017
macro avg	0.652	0.475	0.525	1017
weighted avg	0.660	0.524	0.553	1017


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
50
40
30
20
10
0