COMPARISON_MediaPipe+CNN

June 21, 2025

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

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

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

[16]:
```
num_classes = len(label_map)
```

[17]:
```
len(labels)
```

[17]: 3413

[18]:
```
sequences.shape
```

[18]: (3413, 3, 61, 3)

[19]:
```
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
)
```

[20]:
```
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
```

[21]:
```
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
```

[22]:
```
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)
```

```
(6141, 61, 3, 1)
(6141,)
```

[23]:
```
input_shape = X_train_cnn.shape[1:]
print(input_shape)
```

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

[24]:
```
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)
```

[25]:
```
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'])
```

[26]:
```
history = cnn_model.fit(train_ds,validation_data=val_ds, epochs=50,␣
↪batch_size=64)
```

```
Epoch 1/50
96/96                 2s 10ms/step -
accuracy: 0.0670 - loss: 3.6065 - val_accuracy: 0.1215 - val_loss: 3.0120
Epoch 2/50
96/96                 1s 8ms/step -
accuracy: 0.1005 - loss: 3.0360 - val_accuracy: 0.1396 - val_loss: 2.8006
Epoch 3/50
96/96                 1s 8ms/step -
accuracy: 0.1372 - loss: 2.8652 - val_accuracy: 0.2016 - val_loss: 2.6945
Epoch 4/50
96/96                 1s 8ms/step -
accuracy: 0.1682 - loss: 2.7492 - val_accuracy: 0.2484 - val_loss: 2.5616
Epoch 5/50
96/96                 1s 8ms/step -
accuracy: 0.2075 - loss: 2.6354 - val_accuracy: 0.2596 - val_loss: 2.4914
Epoch 6/50
96/96                 1s 8ms/step -
accuracy: 0.2305 - loss: 2.5260 - val_accuracy: 0.2879 - val_loss: 2.3693
Epoch 7/50
96/96                 1s 8ms/step -
accuracy: 0.2644 - loss: 2.4390 - val_accuracy: 0.3011 - val_loss: 2.2906
Epoch 8/50
96/96                 1s 8ms/step -
accuracy: 0.2873 - loss: 2.3636 - val_accuracy: 0.3490 - val_loss: 2.2312
Epoch 9/50
96/96                 1s 8ms/step -
accuracy: 0.3036 - loss: 2.2794 - val_accuracy: 0.3631 - val_loss: 2.1365
Epoch 10/50
96/96                 1s 8ms/step -
accuracy: 0.3142 - loss: 2.2113 - val_accuracy: 0.3875 - val_loss: 2.0848
Epoch 11/50
96/96                 1s 8ms/step -
accuracy: 0.3441 - loss: 2.1542 - val_accuracy: 0.4109 - val_loss: 2.0155
Epoch 12/50
96/96                 1s 8ms/step -
accuracy: 0.3581 - loss: 2.0818 - val_accuracy: 0.4178 - val_loss: 1.9533
Epoch 13/50
96/96                 1s 8ms/step -
accuracy: 0.3731 - loss: 2.0186 - val_accuracy: 0.4378 - val_loss: 1.8896
Epoch 14/50
96/96                 1s 8ms/step -
```

```
accuracy: 0.3940 - loss: 1.9732 - val_accuracy: 0.4412 - val_loss: 1.8445
Epoch 15/50
96/96            1s 8ms/step -
accuracy: 0.4118 - loss: 1.9094 - val_accuracy: 0.4495 - val_loss: 1.8477
Epoch 16/50
96/96            1s 8ms/step -
accuracy: 0.4218 - loss: 1.8669 - val_accuracy: 0.4632 - val_loss: 1.7870
Epoch 17/50
96/96            1s 8ms/step -
accuracy: 0.4271 - loss: 1.8450 - val_accuracy: 0.4797 - val_loss: 1.7529
Epoch 18/50
96/96            1s 8ms/step -
accuracy: 0.4397 - loss: 1.8074 - val_accuracy: 0.4739 - val_loss: 1.7555
Epoch 19/50
96/96            1s 8ms/step -
accuracy: 0.4487 - loss: 1.7875 - val_accuracy: 0.4915 - val_loss: 1.7247
Epoch 20/50
96/96            1s 8ms/step -
accuracy: 0.4578 - loss: 1.7494 - val_accuracy: 0.4876 - val_loss: 1.7083
Epoch 21/50
96/96            1s 8ms/step -
accuracy: 0.4743 - loss: 1.7076 - val_accuracy: 0.4939 - val_loss: 1.7087
Epoch 22/50
96/96            1s 8ms/step -
accuracy: 0.4591 - loss: 1.7266 - val_accuracy: 0.5095 - val_loss: 1.6672
Epoch 23/50
96/96            1s 8ms/step -
accuracy: 0.4692 - loss: 1.6908 - val_accuracy: 0.5100 - val_loss: 1.6503
Epoch 24/50
96/96            1s 8ms/step -
accuracy: 0.4728 - loss: 1.6675 - val_accuracy: 0.5154 - val_loss: 1.6285
Epoch 25/50
96/96            1s 8ms/step -
accuracy: 0.4837 - loss: 1.6586 - val_accuracy: 0.5295 - val_loss: 1.6044
Epoch 26/50
96/96            1s 8ms/step -
accuracy: 0.4809 - loss: 1.6423 - val_accuracy: 0.5305 - val_loss: 1.6145
Epoch 27/50
96/96            1s 8ms/step -
accuracy: 0.5002 - loss: 1.5973 - val_accuracy: 0.5281 - val_loss: 1.6160
Epoch 28/50
96/96            1s 8ms/step -
accuracy: 0.4962 - loss: 1.5962 - val_accuracy: 0.5378 - val_loss: 1.5876
Epoch 29/50
96/96            1s 8ms/step -
accuracy: 0.4920 - loss: 1.5688 - val_accuracy: 0.5417 - val_loss: 1.5669
Epoch 30/50
96/96            1s 8ms/step -
```

```
accuracy: 0.5170 - loss: 1.5396 - val_accuracy: 0.5437 - val_loss: 1.5693
Epoch 31/50
96/96            1s 8ms/step -
accuracy: 0.5149 - loss: 1.5514 - val_accuracy: 0.5520 - val_loss: 1.5508
Epoch 32/50
96/96            1s 8ms/step -
accuracy: 0.5157 - loss: 1.5185 - val_accuracy: 0.5481 - val_loss: 1.5579
Epoch 33/50
96/96            1s 8ms/step -
accuracy: 0.5291 - loss: 1.4850 - val_accuracy: 0.5373 - val_loss: 1.5638
Epoch 34/50
96/96            1s 8ms/step -
accuracy: 0.5142 - loss: 1.5288 - val_accuracy: 0.5583 - val_loss: 1.5380
Epoch 35/50
96/96            1s 8ms/step -
accuracy: 0.5314 - loss: 1.4901 - val_accuracy: 0.5539 - val_loss: 1.5378
Epoch 36/50
96/96            1s 8ms/step -
accuracy: 0.5272 - loss: 1.4751 - val_accuracy: 0.5554 - val_loss: 1.5386
Epoch 37/50
96/96            1s 8ms/step -
accuracy: 0.5338 - loss: 1.4683 - val_accuracy: 0.5612 - val_loss: 1.5260
Epoch 38/50
96/96            1s 8ms/step -
accuracy: 0.5261 - loss: 1.4780 - val_accuracy: 0.5569 - val_loss: 1.5180
Epoch 39/50
96/96            1s 8ms/step -
accuracy: 0.5259 - loss: 1.4645 - val_accuracy: 0.5632 - val_loss: 1.5240
Epoch 40/50
96/96            1s 8ms/step -
accuracy: 0.5457 - loss: 1.4485 - val_accuracy: 0.5476 - val_loss: 1.5257
Epoch 41/50
96/96            1s 8ms/step -
accuracy: 0.5492 - loss: 1.4288 - val_accuracy: 0.5661 - val_loss: 1.5123
Epoch 42/50
96/96            1s 8ms/step -
accuracy: 0.5490 - loss: 1.4285 - val_accuracy: 0.5676 - val_loss: 1.5095
Epoch 43/50
96/96            1s 8ms/step -
accuracy: 0.5433 - loss: 1.4381 - val_accuracy: 0.5656 - val_loss: 1.5106
Epoch 44/50
96/96            1s 8ms/step -
accuracy: 0.5527 - loss: 1.3903 - val_accuracy: 0.5661 - val_loss: 1.4892
Epoch 45/50
96/96            1s 8ms/step -
accuracy: 0.5439 - loss: 1.4025 - val_accuracy: 0.5632 - val_loss: 1.5096
Epoch 46/50
96/96            1s 8ms/step -
```

```
accuracy: 0.5587 - loss: 1.3836 - val_accuracy: 0.5647 - val_loss: 1.5091
Epoch 47/50
96/96            1s 8ms/step -
accuracy: 0.5538 - loss: 1.3902 - val_accuracy: 0.5710 - val_loss: 1.4939
Epoch 48/50
96/96            1s 8ms/step -
accuracy: 0.5590 - loss: 1.3638 - val_accuracy: 0.5695 - val_loss: 1.5154
Epoch 49/50
96/96            1s 8ms/step -
accuracy: 0.5601 - loss: 1.3747 - val_accuracy: 0.5671 - val_loss: 1.5004
Epoch 50/50
96/96            1s 7ms/step -
accuracy: 0.5760 - loss: 1.3444 - val_accuracy: 0.5661 - val_loss: 1.5173
```

[27]:

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

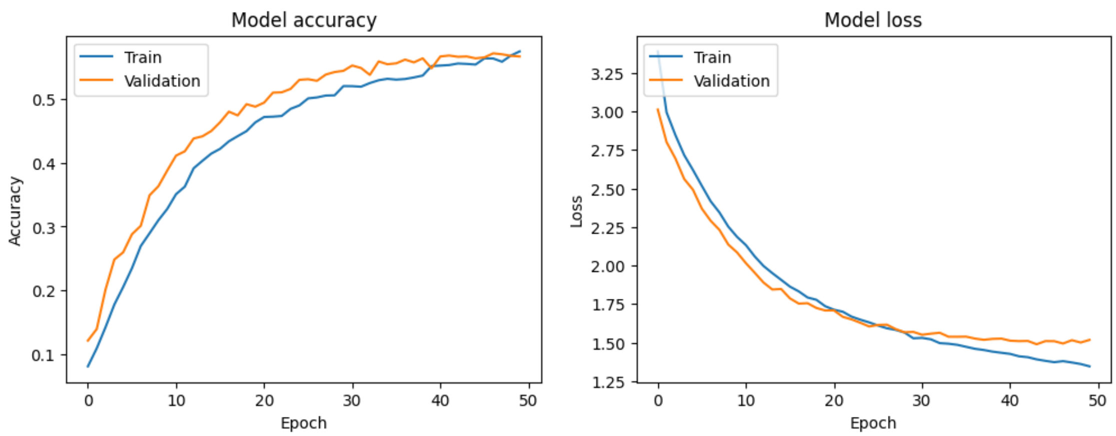
```
33/33            0s 3ms/step -
accuracy: 0.5737 - loss: 1.4620
Test Accuracy: 0.5588
Test Loss: 1.5120
```

[28]:

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

[29]:

```
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()
```



[30]:

```
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      0.812     0.542     0.650        24
           B      0.708     0.567     0.630        30
```

C	0.651	0.519	0.577	54
D	0.323	0.370	0.345	27
E	0.811	0.588	0.682	51
F	0.600	0.167	0.261	18
G	0.750	0.444	0.558	27
H	1.000	0.370	0.541	27
I	0.745	0.556	0.636	63
J	0.800	0.508	0.621	63
K	0.667	0.303	0.417	33
L	0.773	0.596	0.673	57
M	0.667	0.381	0.485	21
N	0.375	0.167	0.231	18
O	0.222	0.970	0.362	66
P	0.667	0.222	0.333	27
Q	0.692	0.333	0.450	27
R	0.667	0.491	0.566	57
S	0.846	0.306	0.449	36
T	0.531	0.436	0.479	39
U	0.464	0.481	0.473	54
V	0.718	0.583	0.644	48
W	0.221	0.627	0.327	51
X	0.714	0.417	0.526	24
Y	0.875	0.467	0.609	15
Z	0.660	0.579	0.617	57
baca	0.818	0.692	0.750	39
bantu	0.774	0.727	0.750	33
bapak	0.419	0.462	0.439	39
buangairkecil	0.875	0.667	0.757	21
buat	0.581	0.923	0.713	39
halo	0.621	0.759	0.683	54
ibu	0.750	0.250	0.375	12
kamu	0.750	0.474	0.581	57
maaf	0.635	0.611	0.623	54
makan	0.769	0.238	0.364	42
mau	0.549	0.765	0.639	51
nama	0.732	0.556	0.632	54
pagi	0.571	0.702	0.630	57
paham	0.677	0.700	0.689	60
sakit	1.000	0.667	0.800	9
sama-sama	0.783	0.653	0.712	72
saya	0.462	0.333	0.387	18
selamat	0.621	0.706	0.661	51
siapa	0.852	0.639	0.730	36
tanya	0.769	0.392	0.519	51
tempat	1.000	0.667	0.800	12
terima-kasih	0.757	0.519	0.615	54
terlambat	0.846	0.564	0.677	39
tidak	0.323	0.762	0.454	42

tolong	0.531	0.436	0.479	39
accuracy			0.559	2049
macro avg	0.675	0.527	0.561	2049
weighted avg	0.659	0.559	0.572	2049