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]: 5643

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

[5]: (5643, 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
```

```python
    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]:
```python
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)
```

```
(10155, 61, 3, 1)
(10155,)
```

[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
159/159              3s 9ms/step -
accuracy: 0.0588 - loss: 3.6041 - val_accuracy: 0.0880 - val_loss: 3.1822
Epoch 2/50
159/159              1s 8ms/step -
accuracy: 0.0937 - loss: 3.1580 - val_accuracy: 0.1305 - val_loss: 2.9874
Epoch 3/50
159/159              1s 8ms/step -
accuracy: 0.1380 - loss: 2.9858 - val_accuracy: 0.1904 - val_loss: 2.7930
Epoch 4/50
159/159              1s 8ms/step -
accuracy: 0.1812 - loss: 2.8329 - val_accuracy: 0.2400 - val_loss: 2.6368
Epoch 5/50
159/159              1s 8ms/step -
accuracy: 0.2153 - loss: 2.7008 - val_accuracy: 0.2764 - val_loss: 2.4937
Epoch 6/50
159/159              1s 8ms/step -
accuracy: 0.2449 - loss: 2.5736 - val_accuracy: 0.3198 - val_loss: 2.3608
Epoch 7/50
159/159              1s 9ms/step -
accuracy: 0.2809 - loss: 2.4456 - val_accuracy: 0.3381 - val_loss: 2.2513
Epoch 8/50
159/159              1s 8ms/step -
accuracy: 0.3034 - loss: 2.3623 - val_accuracy: 0.3679 - val_loss: 2.1709
Epoch 9/50
159/159              1s 8ms/step -
accuracy: 0.3202 - loss: 2.2838 - val_accuracy: 0.3841 - val_loss: 2.1101
Epoch 10/50
159/159              1s 8ms/step -
accuracy: 0.3427 - loss: 2.2191 - val_accuracy: 0.3912 - val_loss: 2.0464
Epoch 11/50
159/159              1s 9ms/step -
accuracy: 0.3567 - loss: 2.1630 - val_accuracy: 0.4012 - val_loss: 2.0283
Epoch 12/50
159/159              1s 8ms/step -
accuracy: 0.3647 - loss: 2.1226 - val_accuracy: 0.4175 - val_loss: 1.9710
Epoch 13/50
159/159              1s 9ms/step -
accuracy: 0.3741 - loss: 2.0781 - val_accuracy: 0.4234 - val_loss: 1.9347
Epoch 14/50
159/159              1s 9ms/step -
```

```
accuracy: 0.3816 - loss: 2.0405 - val_accuracy: 0.4343 - val_loss: 1.9141
Epoch 15/50
159/159            1s 8ms/step -
accuracy: 0.3923 - loss: 2.0112 - val_accuracy: 0.4455 - val_loss: 1.8798
Epoch 16/50
159/159            1s 8ms/step -
accuracy: 0.4066 - loss: 1.9682 - val_accuracy: 0.4529 - val_loss: 1.8571
Epoch 17/50
159/159            1s 8ms/step -
accuracy: 0.4110 - loss: 1.9506 - val_accuracy: 0.4600 - val_loss: 1.8291
Epoch 18/50
159/159            1s 8ms/step -
accuracy: 0.4256 - loss: 1.9082 - val_accuracy: 0.4585 - val_loss: 1.8236
Epoch 19/50
159/159            1s 9ms/step -
accuracy: 0.4246 - loss: 1.8985 - val_accuracy: 0.4680 - val_loss: 1.8238
Epoch 20/50
159/159            1s 8ms/step -
accuracy: 0.4275 - loss: 1.8891 - val_accuracy: 0.4609 - val_loss: 1.8221
Epoch 21/50
159/159            1s 8ms/step -
accuracy: 0.4261 - loss: 1.8734 - val_accuracy: 0.4665 - val_loss: 1.8021
Epoch 22/50
159/159            1s 8ms/step -
accuracy: 0.4270 - loss: 1.8606 - val_accuracy: 0.4774 - val_loss: 1.7894
Epoch 23/50
159/159            1s 8ms/step -
accuracy: 0.4391 - loss: 1.8242 - val_accuracy: 0.4683 - val_loss: 1.7828
Epoch 24/50
159/159            1s 9ms/step -
accuracy: 0.4530 - loss: 1.7970 - val_accuracy: 0.4762 - val_loss: 1.7594
Epoch 25/50
159/159            1s 9ms/step -
accuracy: 0.4466 - loss: 1.8124 - val_accuracy: 0.4845 - val_loss: 1.7465
Epoch 26/50
159/159            2s 9ms/step -
accuracy: 0.4511 - loss: 1.8027 - val_accuracy: 0.4836 - val_loss: 1.7454
Epoch 27/50
159/159            1s 9ms/step -
accuracy: 0.4547 - loss: 1.7692 - val_accuracy: 0.4845 - val_loss: 1.7369
Epoch 28/50
159/159            1s 8ms/step -
accuracy: 0.4646 - loss: 1.7616 - val_accuracy: 0.4854 - val_loss: 1.7397
Epoch 29/50
159/159            1s 8ms/step -
accuracy: 0.4616 - loss: 1.7565 - val_accuracy: 0.4877 - val_loss: 1.7325
Epoch 30/50
159/159            1s 9ms/step -
```

```
accuracy: 0.4697 - loss: 1.7294 - val_accuracy: 0.4957 - val_loss: 1.7190
Epoch 31/50
159/159            1s 9ms/step -
accuracy: 0.4674 - loss: 1.7113 - val_accuracy: 0.4880 - val_loss: 1.7206
Epoch 32/50
159/159            1s 8ms/step -
accuracy: 0.4693 - loss: 1.7273 - val_accuracy: 0.4951 - val_loss: 1.7226
Epoch 33/50
159/159            1s 9ms/step -
accuracy: 0.4778 - loss: 1.7043 - val_accuracy: 0.4966 - val_loss: 1.7135
Epoch 34/50
159/159            1s 9ms/step -
accuracy: 0.4794 - loss: 1.6963 - val_accuracy: 0.4960 - val_loss: 1.7193
Epoch 35/50
159/159            1s 9ms/step -
accuracy: 0.4821 - loss: 1.6897 - val_accuracy: 0.4939 - val_loss: 1.6957
Epoch 36/50
159/159            1s 9ms/step -
accuracy: 0.4774 - loss: 1.6923 - val_accuracy: 0.5019 - val_loss: 1.6964
Epoch 37/50
159/159            1s 9ms/step -
accuracy: 0.4886 - loss: 1.6742 - val_accuracy: 0.4981 - val_loss: 1.6871
Epoch 38/50
159/159            1s 9ms/step -
accuracy: 0.4912 - loss: 1.6628 - val_accuracy: 0.5028 - val_loss: 1.6894
Epoch 39/50
159/159            1s 9ms/step -
accuracy: 0.4861 - loss: 1.6629 - val_accuracy: 0.5075 - val_loss: 1.6850
Epoch 40/50
159/159            1s 9ms/step -
accuracy: 0.4832 - loss: 1.6728 - val_accuracy: 0.5084 - val_loss: 1.6824
Epoch 41/50
159/159            1s 9ms/step -
accuracy: 0.4941 - loss: 1.6576 - val_accuracy: 0.5049 - val_loss: 1.6904
Epoch 42/50
159/159            2s 10ms/step -
accuracy: 0.4956 - loss: 1.6316 - val_accuracy: 0.5013 - val_loss: 1.6966
Epoch 43/50
159/159            2s 9ms/step -
accuracy: 0.4857 - loss: 1.6400 - val_accuracy: 0.5004 - val_loss: 1.7038
Epoch 44/50
159/159            1s 9ms/step -
accuracy: 0.4972 - loss: 1.6144 - val_accuracy: 0.5069 - val_loss: 1.6763
Epoch 45/50
159/159            1s 9ms/step -
accuracy: 0.5008 - loss: 1.6082 - val_accuracy: 0.5096 - val_loss: 1.6744
Epoch 46/50
159/159            1s 9ms/step -
```

```
accuracy: 0.5145 - loss: 1.5788 - val_accuracy: 0.5117 - val_loss: 1.6758
Epoch 47/50
159/159          1s 9ms/step -
accuracy: 0.5075 - loss: 1.6016 - val_accuracy: 0.5120 - val_loss: 1.6782
Epoch 48/50
159/159          2s 10ms/step -
accuracy: 0.4999 - loss: 1.6121 - val_accuracy: 0.5140 - val_loss: 1.6717
Epoch 49/50
159/159          1s 9ms/step -
accuracy: 0.5100 - loss: 1.5664 - val_accuracy: 0.5146 - val_loss: 1.6774
Epoch 50/50
159/159          2s 9ms/step -
accuracy: 0.5016 - loss: 1.5960 - val_accuracy: 0.5128 - val_loss: 1.6649
```

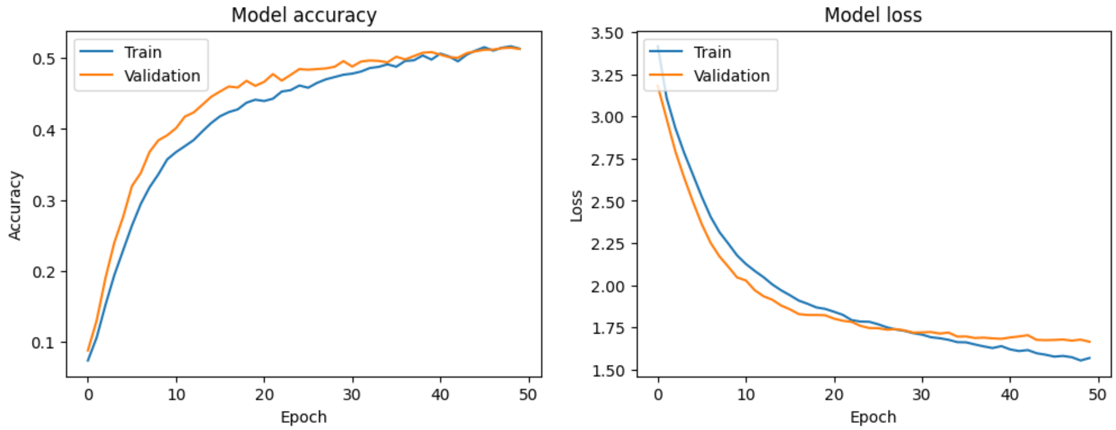
[14]:
```
test_loss, test_accuracy = cnn_model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
 1/53          0s 17ms/step - accuracy:
0.5156 - loss: 1.6272

53/53          0s 4ms/step -
accuracy: 0.5117 - loss: 1.6484
Test Accuracy: 0.5158
Test Loss: 1.6373
```

[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]:
```python
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      0.632     0.400     0.490        60
           B      0.710     0.319     0.440        69
```

C	0.603	0.580	0.591	81
D	0.370	0.270	0.312	63
E	0.683	0.551	0.610	78
F	0.690	0.403	0.509	72
G	0.849	0.517	0.643	87
H	0.627	0.427	0.508	75
I	0.690	0.444	0.541	90
J	0.627	0.578	0.601	90
K	0.400	0.333	0.364	78
L	0.566	0.531	0.548	81
M	0.493	0.376	0.427	93
N	0.552	0.333	0.416	96
O	0.620	0.473	0.537	93
P	0.267	0.107	0.152	75
Q	0.517	0.333	0.405	93
R	0.690	0.372	0.483	78
S	0.812	0.433	0.565	90
T	0.391	0.321	0.352	78
U	0.631	0.488	0.550	84
V	0.560	0.654	0.604	78
W	0.576	0.469	0.517	81
X	0.151	0.545	0.236	66
Y	0.633	0.244	0.352	78
Z	0.138	0.857	0.238	84
baca	0.762	0.667	0.711	48
bantu	0.812	0.619	0.703	42
bapak	0.719	0.511	0.597	45
buangairkecil	0.769	0.833	0.800	24
buat	0.698	0.771	0.733	48
halo	0.588	0.783	0.671	60
ibu	0.571	0.444	0.500	18
kamu	0.729	0.530	0.614	66
maaf	0.364	0.762	0.492	63
makan	0.793	0.451	0.575	51
mau	0.933	0.700	0.800	60
nama	0.671	0.707	0.688	75
pagi	0.800	0.611	0.693	72
paham	0.864	0.760	0.809	75
sakit	1.000	0.467	0.636	15
sama-sama	0.738	0.738	0.738	84
saya	0.562	0.462	0.507	39
selamat	0.617	0.587	0.602	63
siapa	0.655	0.396	0.494	48
tanya	0.600	0.650	0.624	60
tempat	0.789	0.625	0.698	24
terima-kasih	0.582	0.650	0.614	60
terlambat	0.776	0.745	0.760	51
tidak	0.667	0.588	0.625	51

tolong	0.483	0.519	0.500	54
accuracy			0.516	3387
macro avg	0.628	0.528	0.552	3387
weighted avg	0.612	0.516	0.538	3387