COMPARISON_MediaPipe+CNN+LSTM

June 21, 2025

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

[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]:
```
1691
```

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

[5]:
```
(1691, 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()
```

```python
    # 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):
    X = X.transpose(0, 3, 2, 1)  # (N, T, L, F)
    X = X[..., np.newaxis]       # (N, T, L, F, 1)
    return X,y
```

[9]:
```python
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)
```

```
(1014, 3, 61, 3, 1)
(1014,)
```

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

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

[11]:
```python
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 TimeDistributed, Conv2D, MaxPooling2D,␣
 ↪Flatten,Input
from tensorflow.keras.layers import LSTM, Dropout, Dense, BatchNormalization

model = Sequential([
    Input((3, 61, 3, 1)),
    TimeDistributed(Conv2D(32, (3, 2), activation='relu', padding='same')),
    TimeDistributed(BatchNormalization()),
    TimeDistributed(MaxPooling2D(pool_size=(2, 1))),
    TimeDistributed(Dropout(0.25)),

    TimeDistributed(Conv2D(64, (3, 2), activation='relu', padding='same')),
    TimeDistributed(BatchNormalization()),
    TimeDistributed(MaxPooling2D(pool_size=(2, 1))),
    TimeDistributed(Flatten()),

    LSTM(128, return_sequences=False),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

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

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

```
Epoch 1/50
16/16              6s 88ms/step -
accuracy: 0.0745 - loss: 3.1710 - val_accuracy: 0.1036 - val_loss: 3.0451
Epoch 2/50
16/16              1s 53ms/step -
accuracy: 0.1071 - loss: 2.8903 - val_accuracy: 0.1006 - val_loss: 2.9353
Epoch 3/50
16/16              1s 53ms/step -
accuracy: 0.0954 - loss: 2.8460 - val_accuracy: 0.1361 - val_loss: 2.8889
Epoch 4/50
16/16              1s 52ms/step -
accuracy: 0.1195 - loss: 2.7836 - val_accuracy: 0.1036 - val_loss: 2.8704
Epoch 5/50
16/16              1s 50ms/step -
accuracy: 0.1575 - loss: 2.7642 - val_accuracy: 0.1065 - val_loss: 2.8336
```

```
Epoch 6/50
16/16           1s 58ms/step -
accuracy: 0.1530 - loss: 2.7059 - val_accuracy: 0.0858 - val_loss: 2.7923
Epoch 7/50
16/16           1s 55ms/step -
accuracy: 0.1703 - loss: 2.6484 - val_accuracy: 0.1124 - val_loss: 2.7238
Epoch 8/50
16/16           1s 52ms/step -
accuracy: 0.2210 - loss: 2.5734 - val_accuracy: 0.1272 - val_loss: 2.6760
Epoch 9/50
16/16           1s 50ms/step -
accuracy: 0.2127 - loss: 2.5619 - val_accuracy: 0.1243 - val_loss: 2.6858
Epoch 10/50
16/16           1s 50ms/step -
accuracy: 0.2772 - loss: 2.4206 - val_accuracy: 0.2426 - val_loss: 2.5392
Epoch 11/50
16/16           1s 51ms/step -
accuracy: 0.3120 - loss: 2.3746 - val_accuracy: 0.3047 - val_loss: 2.4440
Epoch 12/50
16/16           1s 50ms/step -
accuracy: 0.3578 - loss: 2.2967 - val_accuracy: 0.3343 - val_loss: 2.3281
Epoch 13/50
16/16           1s 51ms/step -
accuracy: 0.3902 - loss: 2.1895 - val_accuracy: 0.4852 - val_loss: 2.1701
Epoch 14/50
16/16           1s 48ms/step -
accuracy: 0.4225 - loss: 2.0491 - val_accuracy: 0.4941 - val_loss: 2.0390
Epoch 15/50
16/16           1s 47ms/step -
accuracy: 0.4730 - loss: 1.9566 - val_accuracy: 0.5296 - val_loss: 1.9408
Epoch 16/50
16/16           1s 47ms/step -
accuracy: 0.5299 - loss: 1.8429 - val_accuracy: 0.5858 - val_loss: 1.8587
Epoch 17/50
16/16           1s 48ms/step -
accuracy: 0.5488 - loss: 1.7402 - val_accuracy: 0.5533 - val_loss: 1.8296
Epoch 18/50
16/16           1s 48ms/step -
accuracy: 0.5811 - loss: 1.6671 - val_accuracy: 0.6124 - val_loss: 1.7394
Epoch 19/50
16/16           1s 47ms/step -
accuracy: 0.5831 - loss: 1.6190 - val_accuracy: 0.6036 - val_loss: 1.7051
Epoch 20/50
16/16           1s 47ms/step -
accuracy: 0.5946 - loss: 1.5596 - val_accuracy: 0.6095 - val_loss: 1.5964
Epoch 21/50
16/16           1s 47ms/step -
accuracy: 0.6252 - loss: 1.4615 - val_accuracy: 0.6598 - val_loss: 1.5475
```

```
Epoch 22/50
16/16            1s 47ms/step -
accuracy: 0.6579 - loss: 1.3494 - val_accuracy: 0.6657 - val_loss: 1.4490
Epoch 23/50
16/16            1s 48ms/step -
accuracy: 0.6758 - loss: 1.3400 - val_accuracy: 0.7041 - val_loss: 1.4354
Epoch 24/50
16/16            1s 47ms/step -
accuracy: 0.6648 - loss: 1.2927 - val_accuracy: 0.6982 - val_loss: 1.3785
Epoch 25/50
16/16            1s 50ms/step -
accuracy: 0.6837 - loss: 1.2203 - val_accuracy: 0.7071 - val_loss: 1.3459
Epoch 26/50
16/16            1s 47ms/step -
accuracy: 0.7259 - loss: 1.1556 - val_accuracy: 0.6982 - val_loss: 1.3418
Epoch 27/50
16/16            1s 47ms/step -
accuracy: 0.7278 - loss: 1.0756 - val_accuracy: 0.7337 - val_loss: 1.2258
Epoch 28/50
16/16            1s 48ms/step -
accuracy: 0.7287 - loss: 1.0332 - val_accuracy: 0.7426 - val_loss: 1.2811
Epoch 29/50
16/16            1s 47ms/step -
accuracy: 0.7304 - loss: 1.0409 - val_accuracy: 0.7456 - val_loss: 1.1965
Epoch 30/50
16/16            1s 47ms/step -
accuracy: 0.7602 - loss: 0.9565 - val_accuracy: 0.7337 - val_loss: 1.2489
Epoch 31/50
16/16            1s 48ms/step -
accuracy: 0.7599 - loss: 0.9259 - val_accuracy: 0.7663 - val_loss: 1.1297
Epoch 32/50
16/16            1s 47ms/step -
accuracy: 0.7937 - loss: 0.8706 - val_accuracy: 0.7515 - val_loss: 1.1258
Epoch 33/50
16/16            1s 47ms/step -
accuracy: 0.7872 - loss: 0.8456 - val_accuracy: 0.7485 - val_loss: 1.0986
Epoch 34/50
16/16            1s 48ms/step -
accuracy: 0.8140 - loss: 0.8400 - val_accuracy: 0.7278 - val_loss: 1.1280
Epoch 35/50
16/16            1s 48ms/step -
accuracy: 0.7886 - loss: 0.8255 - val_accuracy: 0.7751 - val_loss: 1.0405
Epoch 36/50
16/16            1s 47ms/step -
accuracy: 0.7889 - loss: 0.7891 - val_accuracy: 0.7663 - val_loss: 1.0391
Epoch 37/50
16/16            1s 47ms/step -
accuracy: 0.8137 - loss: 0.7576 - val_accuracy: 0.7722 - val_loss: 1.0000
```

```
Epoch 38/50
16/16            1s 47ms/step -
accuracy: 0.8301 - loss: 0.7096 - val_accuracy: 0.7692 - val_loss: 1.0137
Epoch 39/50
16/16            1s 47ms/step -
accuracy: 0.8383 - loss: 0.6953 - val_accuracy: 0.7781 - val_loss: 1.0482
Epoch 40/50
16/16            1s 52ms/step -
accuracy: 0.8430 - loss: 0.6941 - val_accuracy: 0.7574 - val_loss: 1.0049
Epoch 41/50
16/16            1s 47ms/step -
accuracy: 0.8420 - loss: 0.6595 - val_accuracy: 0.7870 - val_loss: 0.9614
Epoch 42/50
16/16            1s 47ms/step -
accuracy: 0.8561 - loss: 0.6551 - val_accuracy: 0.7781 - val_loss: 0.9219
Epoch 43/50
16/16            1s 48ms/step -
accuracy: 0.8615 - loss: 0.5901 - val_accuracy: 0.7722 - val_loss: 0.9490
Epoch 44/50
16/16            1s 48ms/step -
accuracy: 0.8643 - loss: 0.6077 - val_accuracy: 0.7604 - val_loss: 0.9769
Epoch 45/50
16/16            1s 47ms/step -
accuracy: 0.8799 - loss: 0.5942 - val_accuracy: 0.7692 - val_loss: 0.9981
Epoch 46/50
16/16            1s 47ms/step -
accuracy: 0.8807 - loss: 0.5415 - val_accuracy: 0.7663 - val_loss: 0.9183
Epoch 47/50
16/16            1s 48ms/step -
accuracy: 0.9026 - loss: 0.5172 - val_accuracy: 0.7870 - val_loss: 0.9016
Epoch 48/50
16/16            1s 48ms/step -
accuracy: 0.8831 - loss: 0.4932 - val_accuracy: 0.7751 - val_loss: 0.8911
Epoch 49/50
16/16            1s 47ms/step -
accuracy: 0.8864 - loss: 0.4707 - val_accuracy: 0.7811 - val_loss: 0.8741
Epoch 50/50
16/16            1s 47ms/step -
accuracy: 0.8986 - loss: 0.4725 - val_accuracy: 0.7899 - val_loss: 0.8613
```

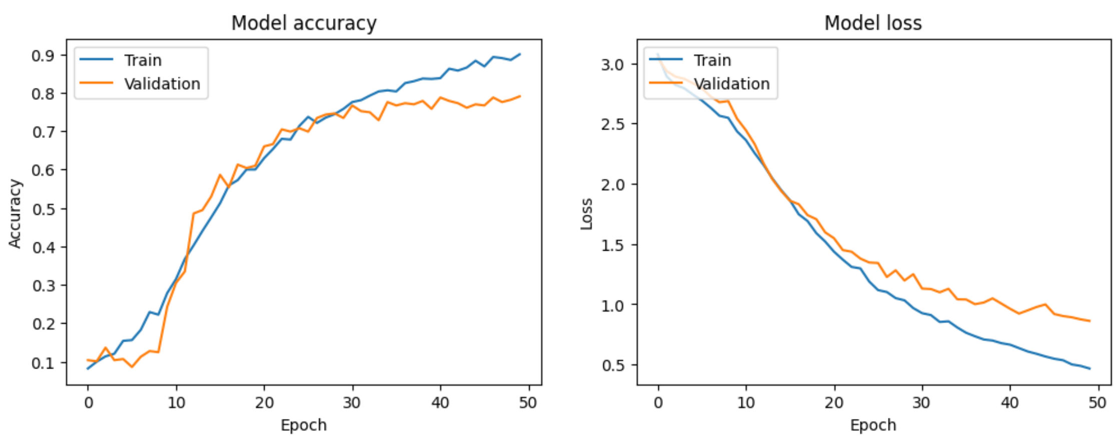
[14]:

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

```
6/6            0s 10ms/step -
accuracy: 0.7523 - loss: 0.9464
Test Accuracy: 0.7611
Test Loss: 0.9174
```

[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 = model.predict(X_batch, verbose=0)
    y_pred.append(np.argmax(batch_pred, axis=1))

y_true = np.concatenate(y_true)
```

```python
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      0.583     0.875     0.700         8
           B      1.000     0.600     0.750        10
           C      0.875     0.778     0.824        18
           D      0.235     0.444     0.308         9
           E      0.944     0.944     0.944        18
           F      0.500     0.667     0.571         6
           G      0.889     0.889     0.889         9
           H      0.833     0.556     0.667         9
           I      0.952     0.909     0.930        22
           J      0.656     1.000     0.792        21
           K      0.800     0.364     0.500        11
           L      1.000     0.737     0.848        19
           M      1.000     0.286     0.444         7
           N      0.333     0.333     0.333         6
           O      0.741     0.909     0.816        22
           P      0.667     0.222     0.333         9
           Q      1.000     0.556     0.714         9
           R      0.750     0.947     0.837        19
           S      0.833     0.455     0.588        11
           T      0.474     0.692     0.562        13
           U      0.850     0.944     0.895        18
           V      1.000     1.000     1.000        16
           W      0.765     0.765     0.765        17
           X      1.000     0.500     0.667         8
           Y      1.000     0.400     0.571         5
           Z      0.792     1.000     0.884        19
```

accuracy			0.761	339
macro avg	0.787	0.684	0.697	339
weighted avg	0.805	0.761	0.756	339