COMPARISON_MediaPipe+CNN+LSTM

June 21, 2025

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

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

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

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

[4]: 3488

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

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

[6]:
```python
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]:
```python
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):
    X = X.transpose(0, 3, 2, 1)  # (N, T, L, F)
    X = X[..., np.newaxis]       # (N, T, L, F, 1)
    return X,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)
```

```
(2092, 3, 61, 3, 1)
(2092,)
```

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

```
(3, 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 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
33/33              6s 66ms/step -
accuracy: 0.0526 - loss: 3.3014 - val_accuracy: 0.0745 - val_loss: 3.1521
Epoch 2/50
33/33              2s 50ms/step -
accuracy: 0.0682 - loss: 3.1579 - val_accuracy: 0.0774 - val_loss: 3.0845
Epoch 3/50
33/33              2s 50ms/step -
accuracy: 0.0759 - loss: 3.0987 - val_accuracy: 0.1089 - val_loss: 3.0402
Epoch 4/50
33/33              2s 48ms/step -
accuracy: 0.0840 - loss: 3.0622 - val_accuracy: 0.0960 - val_loss: 2.9985
Epoch 5/50
33/33              2s 51ms/step -
accuracy: 0.0886 - loss: 3.0486 - val_accuracy: 0.1418 - val_loss: 2.9539
```

```
Epoch 6/50
33/33            2s 61ms/step -
accuracy: 0.0963 - loss: 3.0170 - val_accuracy: 0.0960 - val_loss: 2.9128
Epoch 7/50
33/33            2s 64ms/step -
accuracy: 0.1267 - loss: 2.9517 - val_accuracy: 0.1605 - val_loss: 2.8491
Epoch 8/50
33/33            2s 58ms/step -
accuracy: 0.1392 - loss: 2.8937 - val_accuracy: 0.1834 - val_loss: 2.8173
Epoch 9/50
33/33            2s 53ms/step -
accuracy: 0.1743 - loss: 2.8339 - val_accuracy: 0.1848 - val_loss: 2.7654
Epoch 10/50
33/33            2s 58ms/step -
accuracy: 0.2060 - loss: 2.7474 - val_accuracy: 0.2464 - val_loss: 2.6593
Epoch 11/50
33/33            2s 59ms/step -
accuracy: 0.2264 - loss: 2.6703 - val_accuracy: 0.3195 - val_loss: 2.5625
Epoch 12/50
33/33            2s 57ms/step -
accuracy: 0.2485 - loss: 2.5978 - val_accuracy: 0.3467 - val_loss: 2.4301
Epoch 13/50
33/33            2s 56ms/step -
accuracy: 0.3020 - loss: 2.4781 - val_accuracy: 0.3496 - val_loss: 2.4377
Epoch 14/50
33/33            2s 49ms/step -
accuracy: 0.3139 - loss: 2.3855 - val_accuracy: 0.3840 - val_loss: 2.3150
Epoch 15/50
33/33            2s 48ms/step -
accuracy: 0.3704 - loss: 2.2743 - val_accuracy: 0.4513 - val_loss: 2.1860
Epoch 16/50
33/33            2s 51ms/step -
accuracy: 0.4008 - loss: 2.1787 - val_accuracy: 0.3653 - val_loss: 2.2596
Epoch 17/50
33/33            2s 52ms/step -
accuracy: 0.3991 - loss: 2.0795 - val_accuracy: 0.4513 - val_loss: 2.0517
Epoch 18/50
33/33            2s 50ms/step -
accuracy: 0.4269 - loss: 2.0340 - val_accuracy: 0.4398 - val_loss: 2.0470
Epoch 19/50
33/33            2s 49ms/step -
accuracy: 0.4710 - loss: 1.9426 - val_accuracy: 0.5000 - val_loss: 1.9191
Epoch 20/50
33/33            2s 49ms/step -
accuracy: 0.4595 - loss: 1.8732 - val_accuracy: 0.4771 - val_loss: 1.9459
Epoch 21/50
33/33            2s 48ms/step -
accuracy: 0.4952 - loss: 1.7686 - val_accuracy: 0.5244 - val_loss: 1.8199
```

```
Epoch 22/50
33/33            2s 49ms/step -
accuracy: 0.5095 - loss: 1.6887 - val_accuracy: 0.5086 - val_loss: 1.7860
Epoch 23/50
33/33            2s 50ms/step -
accuracy: 0.5202 - loss: 1.6639 - val_accuracy: 0.4871 - val_loss: 1.9429
Epoch 24/50
33/33            2s 49ms/step -
accuracy: 0.5427 - loss: 1.6006 - val_accuracy: 0.5029 - val_loss: 1.8247
Epoch 25/50
33/33            2s 49ms/step -
accuracy: 0.5362 - loss: 1.6011 - val_accuracy: 0.5688 - val_loss: 1.6381
Epoch 26/50
33/33            2s 48ms/step -
accuracy: 0.5516 - loss: 1.5341 - val_accuracy: 0.5458 - val_loss: 1.6210
Epoch 27/50
33/33            2s 48ms/step -
accuracy: 0.5697 - loss: 1.4711 - val_accuracy: 0.6418 - val_loss: 1.4740
Epoch 28/50
33/33            2s 50ms/step -
accuracy: 0.6006 - loss: 1.4071 - val_accuracy: 0.5989 - val_loss: 1.5208
Epoch 29/50
33/33            2s 49ms/step -
accuracy: 0.5919 - loss: 1.3827 - val_accuracy: 0.5287 - val_loss: 1.5904
Epoch 30/50
33/33            2s 49ms/step -
accuracy: 0.6250 - loss: 1.3623 - val_accuracy: 0.6218 - val_loss: 1.4242
Epoch 31/50
33/33            2s 48ms/step -
accuracy: 0.6179 - loss: 1.3073 - val_accuracy: 0.6490 - val_loss: 1.3192
Epoch 32/50
33/33            2s 50ms/step -
accuracy: 0.6405 - loss: 1.2929 - val_accuracy: 0.6289 - val_loss: 1.3961
Epoch 33/50
33/33            2s 54ms/step -
accuracy: 0.6491 - loss: 1.2397 - val_accuracy: 0.5487 - val_loss: 1.5479
Epoch 34/50
33/33            2s 53ms/step -
accuracy: 0.6258 - loss: 1.2620 - val_accuracy: 0.5888 - val_loss: 1.4296
Epoch 35/50
33/33            2s 51ms/step -
accuracy: 0.6500 - loss: 1.2119 - val_accuracy: 0.5745 - val_loss: 1.3688
Epoch 36/50
33/33            2s 55ms/step -
accuracy: 0.6546 - loss: 1.1764 - val_accuracy: 0.5673 - val_loss: 1.4578
Epoch 37/50
33/33            2s 50ms/step -
accuracy: 0.6684 - loss: 1.1202 - val_accuracy: 0.6189 - val_loss: 1.3412
```

```
Epoch 38/50
33/33             2s 53ms/step -
accuracy: 0.7068 - loss: 1.0492 - val_accuracy: 0.6232 - val_loss: 1.2689
Epoch 39/50
33/33             2s 51ms/step -
accuracy: 0.6788 - loss: 1.0312 - val_accuracy: 0.6461 - val_loss: 1.2609
Epoch 40/50
33/33             2s 53ms/step -
accuracy: 0.6981 - loss: 0.9774 - val_accuracy: 0.5989 - val_loss: 1.3219
Epoch 41/50
33/33             2s 53ms/step -
accuracy: 0.7071 - loss: 0.9732 - val_accuracy: 0.6361 - val_loss: 1.2340
Epoch 42/50
33/33             2s 53ms/step -
accuracy: 0.7175 - loss: 0.9505 - val_accuracy: 0.6275 - val_loss: 1.2091
Epoch 43/50
33/33             2s 53ms/step -
accuracy: 0.7206 - loss: 0.9661 - val_accuracy: 0.6920 - val_loss: 1.0771
Epoch 44/50
33/33             2s 51ms/step -
accuracy: 0.7210 - loss: 0.9566 - val_accuracy: 0.6304 - val_loss: 1.2521
Epoch 45/50
33/33             2s 52ms/step -
accuracy: 0.6993 - loss: 0.9491 - val_accuracy: 0.6218 - val_loss: 1.2147
Epoch 46/50
33/33             2s 55ms/step -
accuracy: 0.7266 - loss: 0.8977 - val_accuracy: 0.6633 - val_loss: 1.1671
Epoch 47/50
33/33             2s 52ms/step -
accuracy: 0.7521 - loss: 0.8862 - val_accuracy: 0.6619 - val_loss: 1.1203
Epoch 48/50
33/33             2s 51ms/step -
accuracy: 0.7667 - loss: 0.8230 - val_accuracy: 0.6246 - val_loss: 1.2619
Epoch 49/50
33/33             2s 55ms/step -
accuracy: 0.7548 - loss: 0.8559 - val_accuracy: 0.6189 - val_loss: 1.2847
Epoch 50/50
33/33             2s 53ms/step -
accuracy: 0.7470 - loss: 0.8342 - val_accuracy: 0.6633 - val_loss: 1.1711
```

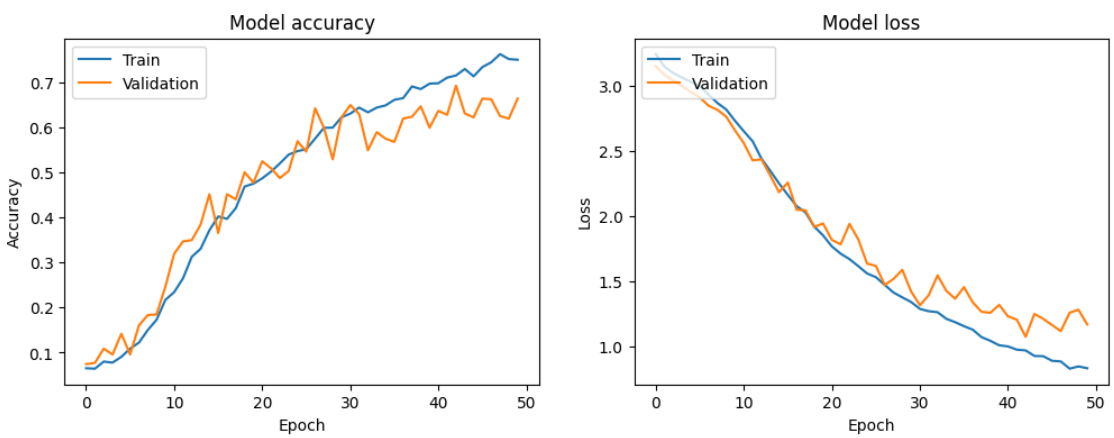
[14]:

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

```
11/11             0s 9ms/step -
accuracy: 0.6181 - loss: 1.2693
Test Accuracy: 0.6304
Test Loss: 1.2593
```

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

[16]:
```python
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 = 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      0.545     0.600     0.571        20
           B      0.810     0.739     0.773        23
           C      0.818     0.321     0.462        28
           D      0.476     0.500     0.488        20
           E      0.952     0.769     0.851        26
           F      0.789     0.625     0.698        24
           G      0.667     0.966     0.789        29
           H      0.621     0.720     0.667        25
           I      0.929     0.867     0.897        30
           J      0.933     0.933     0.933        30
           K      0.560     0.560     0.560        25
           L      1.000     0.481     0.650        27
           M      0.500     0.355     0.415        31
           N      0.438     0.656     0.525        32
           O      0.714     0.167     0.270        30
           P      1.000     0.400     0.571        25
           Q      0.512     0.733     0.603        30
           R      0.773     0.630     0.694        27
           S      0.455     0.667     0.541        30
           T      0.524     0.423     0.468        26
           U      0.512     0.724     0.600        29
           V      1.000     0.880     0.936        25
           W      0.773     0.630     0.694        27
           X      1.000     0.391     0.562        23
           Y      0.247     0.741     0.370        27
           Z      1.000     0.828     0.906        29
```

accuracy			0.630	698
macro avg	0.713	0.627	0.634	698
weighted avg	0.711	0.630	0.634	698