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]: 1722

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

[5]: (1722, 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)
```

```
(1033, 3, 61, 3, 1)
(1033,)
```

[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
17/17              6s 91ms/step -
accuracy: 0.0948 - loss: 3.1329 - val_accuracy: 0.1192 - val_loss: 3.1134
Epoch 2/50
17/17              1s 58ms/step -
accuracy: 0.2185 - loss: 2.6554 - val_accuracy: 0.0872 - val_loss: 3.0680
Epoch 3/50
17/17              1s 55ms/step -
accuracy: 0.3175 - loss: 2.3707 - val_accuracy: 0.1163 - val_loss: 3.0046
Epoch 4/50
17/17              1s 57ms/step -
accuracy: 0.3634 - loss: 2.2721 - val_accuracy: 0.1453 - val_loss: 2.9339
Epoch 5/50
17/17              1s 54ms/step -
accuracy: 0.4012 - loss: 2.0538 - val_accuracy: 0.2645 - val_loss: 2.7263
```

```
Epoch 6/50
17/17            1s 52ms/step -
accuracy: 0.4447 - loss: 1.9395 - val_accuracy: 0.2616 - val_loss: 2.5388
Epoch 7/50
17/17            1s 47ms/step -
accuracy: 0.4373 - loss: 1.9222 - val_accuracy: 0.3023 - val_loss: 2.4330
Epoch 8/50
17/17            1s 48ms/step -
accuracy: 0.5558 - loss: 1.7145 - val_accuracy: 0.4273 - val_loss: 2.1990
Epoch 9/50
17/17            1s 47ms/step -
accuracy: 0.5358 - loss: 1.6715 - val_accuracy: 0.4738 - val_loss: 2.0136
Epoch 10/50
17/17            1s 48ms/step -
accuracy: 0.5771 - loss: 1.5514 - val_accuracy: 0.5349 - val_loss: 1.8477
Epoch 11/50
17/17            1s 48ms/step -
accuracy: 0.6159 - loss: 1.4467 - val_accuracy: 0.5552 - val_loss: 1.6849
Epoch 12/50
17/17            1s 48ms/step -
accuracy: 0.6478 - loss: 1.3760 - val_accuracy: 0.5727 - val_loss: 1.6199
Epoch 13/50
17/17            1s 48ms/step -
accuracy: 0.6681 - loss: 1.2819 - val_accuracy: 0.5494 - val_loss: 1.5836
Epoch 14/50
17/17            1s 49ms/step -
accuracy: 0.6684 - loss: 1.2353 - val_accuracy: 0.5959 - val_loss: 1.4854
Epoch 15/50
17/17            1s 49ms/step -
accuracy: 0.6957 - loss: 1.1325 - val_accuracy: 0.6366 - val_loss: 1.3885
Epoch 16/50
17/17            1s 48ms/step -
accuracy: 0.6981 - loss: 1.1730 - val_accuracy: 0.6715 - val_loss: 1.3355
Epoch 17/50
17/17            1s 49ms/step -
accuracy: 0.7269 - loss: 1.0604 - val_accuracy: 0.6512 - val_loss: 1.3351
Epoch 18/50
17/17            1s 49ms/step -
accuracy: 0.7616 - loss: 0.9893 - val_accuracy: 0.6744 - val_loss: 1.2629
Epoch 19/50
17/17            1s 49ms/step -
accuracy: 0.7735 - loss: 0.9548 - val_accuracy: 0.6686 - val_loss: 1.2639
Epoch 20/50
17/17            1s 48ms/step -
accuracy: 0.7561 - loss: 0.9183 - val_accuracy: 0.6686 - val_loss: 1.2440
Epoch 21/50
17/17            1s 48ms/step -
accuracy: 0.7898 - loss: 0.8567 - val_accuracy: 0.7006 - val_loss: 1.1422
```

```
Epoch 22/50
17/17            1s 51ms/step -
accuracy: 0.8096 - loss: 0.7984 - val_accuracy: 0.7413 - val_loss: 1.0513
Epoch 23/50
17/17            1s 48ms/step -
accuracy: 0.7992 - loss: 0.7533 - val_accuracy: 0.7326 - val_loss: 1.0464
Epoch 24/50
17/17            1s 49ms/step -
accuracy: 0.8064 - loss: 0.7515 - val_accuracy: 0.7500 - val_loss: 0.9901
Epoch 25/50
17/17            1s 49ms/step -
accuracy: 0.8258 - loss: 0.6996 - val_accuracy: 0.7064 - val_loss: 1.0419
Epoch 26/50
17/17            1s 48ms/step -
accuracy: 0.8576 - loss: 0.6416 - val_accuracy: 0.7587 - val_loss: 0.9356
Epoch 27/50
17/17            1s 48ms/step -
accuracy: 0.8602 - loss: 0.6108 - val_accuracy: 0.7733 - val_loss: 0.9348
Epoch 28/50
17/17            1s 52ms/step -
accuracy: 0.8455 - loss: 0.6148 - val_accuracy: 0.7413 - val_loss: 0.9779
Epoch 29/50
17/17            1s 49ms/step -
accuracy: 0.8537 - loss: 0.6062 - val_accuracy: 0.7703 - val_loss: 0.9444
Epoch 30/50
17/17            1s 48ms/step -
accuracy: 0.8626 - loss: 0.5710 - val_accuracy: 0.8023 - val_loss: 0.8876
Epoch 31/50
17/17            1s 49ms/step -
accuracy: 0.8774 - loss: 0.5386 - val_accuracy: 0.7762 - val_loss: 0.8669
Epoch 32/50
17/17            1s 48ms/step -
accuracy: 0.8631 - loss: 0.5756 - val_accuracy: 0.7733 - val_loss: 0.8590
Epoch 33/50
17/17            1s 48ms/step -
accuracy: 0.8748 - loss: 0.5268 - val_accuracy: 0.7791 - val_loss: 0.8600
Epoch 34/50
17/17            1s 49ms/step -
accuracy: 0.8776 - loss: 0.4866 - val_accuracy: 0.7762 - val_loss: 0.8685
Epoch 35/50
17/17            1s 52ms/step -
accuracy: 0.9087 - loss: 0.4569 - val_accuracy: 0.7849 - val_loss: 0.9124
Epoch 36/50
17/17            1s 49ms/step -
accuracy: 0.8865 - loss: 0.4913 - val_accuracy: 0.8110 - val_loss: 0.8460
Epoch 37/50
17/17            1s 49ms/step -
accuracy: 0.8830 - loss: 0.4608 - val_accuracy: 0.8023 - val_loss: 0.8428
```

```
Epoch 38/50
17/17              1s 48ms/step -
accuracy: 0.8596 - loss: 0.5185 - val_accuracy: 0.7994 - val_loss: 0.8429
Epoch 39/50
17/17              1s 48ms/step -
accuracy: 0.8957 - loss: 0.4660 - val_accuracy: 0.8052 - val_loss: 0.8251
Epoch 40/50
17/17              1s 48ms/step -
accuracy: 0.9164 - loss: 0.3995 - val_accuracy: 0.8140 - val_loss: 0.7643
Epoch 41/50
17/17              1s 48ms/step -
accuracy: 0.8954 - loss: 0.4456 - val_accuracy: 0.7733 - val_loss: 0.7977
Epoch 42/50
17/17              1s 49ms/step -
accuracy: 0.9109 - loss: 0.4084 - val_accuracy: 0.7849 - val_loss: 0.7672
Epoch 43/50
17/17              1s 48ms/step -
accuracy: 0.9080 - loss: 0.3954 - val_accuracy: 0.8140 - val_loss: 0.7641
Epoch 44/50
17/17              1s 51ms/step -
accuracy: 0.9062 - loss: 0.3786 - val_accuracy: 0.8256 - val_loss: 0.7286
Epoch 45/50
17/17              1s 55ms/step -
accuracy: 0.8937 - loss: 0.3991 - val_accuracy: 0.7965 - val_loss: 0.7984
Epoch 46/50
17/17              1s 62ms/step -
accuracy: 0.9230 - loss: 0.3566 - val_accuracy: 0.7849 - val_loss: 0.8290
Epoch 47/50
17/17              1s 50ms/step -
accuracy: 0.9388 - loss: 0.3292 - val_accuracy: 0.8052 - val_loss: 0.7480
Epoch 48/50
17/17              1s 53ms/step -
accuracy: 0.9307 - loss: 0.3336 - val_accuracy: 0.7762 - val_loss: 0.8068
Epoch 49/50
17/17              1s 53ms/step -
accuracy: 0.9117 - loss: 0.3713 - val_accuracy: 0.7936 - val_loss: 0.7892
Epoch 50/50
17/17              1s 52ms/step -
accuracy: 0.9248 - loss: 0.3455 - val_accuracy: 0.8198 - val_loss: 0.7747
```

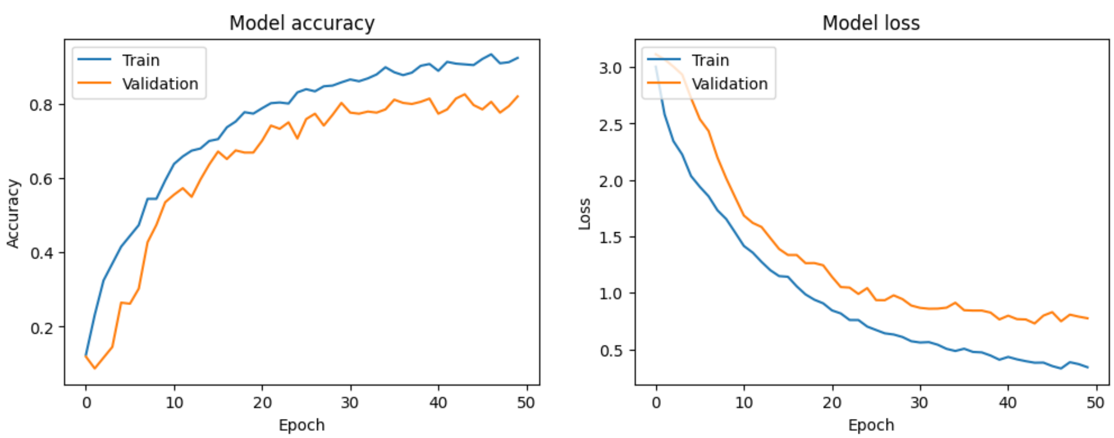
[14]:

```
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 9ms/step -
accuracy: 0.8481 - loss: 0.6277
Test Accuracy: 0.8522
Test Loss: 0.6277
```

[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

         baca      1.000     0.833     0.909        12
        bantu      1.000     0.727     0.842        11
        bapak      0.786     0.917     0.846        12
buangairkecil      1.000     1.000     1.000         6
         buat      0.812     1.000     0.897        13
         halo      0.900     1.000     0.947        18
          ibu      1.000     0.750     0.857         4
         kamu      0.682     0.789     0.732        19
         maaf      1.000     1.000     1.000        18
        makan      1.000     0.714     0.833        14
          mau      0.933     0.824     0.875        17
         nama      0.833     0.833     0.833        18
         pagi      0.947     0.900     0.923        20
        paham      0.950     0.950     0.950        20
        sakit      1.000     0.667     0.800         3
    sama-sama      0.885     0.920     0.902        25
         saya      0.600     0.500     0.545         6
      selamat      0.882     0.833     0.857        18
        siapa      1.000     0.750     0.857        12
        tanya      0.867     0.765     0.812        17
       tempat      1.000     0.250     0.400         4
  terima-kasih     0.773     0.944     0.850        18
    terlambat      0.722     1.000     0.839        13
        tidak      0.909     0.714     0.800        14
       tolong      0.500     0.769     0.606        13

     accuracy                          0.852       345
```

macro avg	0.879	0.814	0.829	345
weighted avg	0.872	0.852	0.852	345