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]: 2155

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

[5]: (2155, 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()
```

```
    # 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)
```

```
(1293, 3, 61, 3, 1)
(1293,)
```

[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)
```

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

```
Epoch 1/50
21/21              6s 68ms/step -
accuracy: 0.1066 - loss: 3.1065 - val_accuracy: 0.0464 - val_loss: 3.1254
Epoch 2/50
21/21              1s 46ms/step -
accuracy: 0.2142 - loss: 2.6277 - val_accuracy: 0.0464 - val_loss: 3.1366
Epoch 3/50
21/21              1s 43ms/step -
accuracy: 0.2999 - loss: 2.3772 - val_accuracy: 0.1230 - val_loss: 3.0154
Epoch 4/50
21/21              1s 44ms/step -
accuracy: 0.3226 - loss: 2.2264 - val_accuracy: 0.2158 - val_loss: 2.8296
Epoch 5/50
21/21              1s 43ms/step -
accuracy: 0.3770 - loss: 2.0676 - val_accuracy: 0.2019 - val_loss: 2.7027
```

```
Epoch 6/50
21/21            1s 43ms/step -
accuracy: 0.4285 - loss: 1.9289 - val_accuracy: 0.2854 - val_loss: 2.4966
Epoch 7/50
21/21            1s 55ms/step -
accuracy: 0.4817 - loss: 1.8106 - val_accuracy: 0.3712 - val_loss: 2.3343
Epoch 8/50
21/21            1s 44ms/step -
accuracy: 0.4966 - loss: 1.7196 - val_accuracy: 0.4362 - val_loss: 2.1730
Epoch 9/50
21/21            1s 44ms/step -
accuracy: 0.5453 - loss: 1.6237 - val_accuracy: 0.4687 - val_loss: 2.0802
Epoch 10/50
21/21            1s 44ms/step -
accuracy: 0.5712 - loss: 1.5956 - val_accuracy: 0.4872 - val_loss: 1.9338
Epoch 11/50
21/21            1s 43ms/step -
accuracy: 0.5930 - loss: 1.4754 - val_accuracy: 0.4942 - val_loss: 1.8980
Epoch 12/50
21/21            1s 61ms/step -
accuracy: 0.6227 - loss: 1.4098 - val_accuracy: 0.5220 - val_loss: 1.7979
Epoch 13/50
21/21            1s 49ms/step -
accuracy: 0.6485 - loss: 1.3376 - val_accuracy: 0.5244 - val_loss: 1.7424
Epoch 14/50
21/21            1s 48ms/step -
accuracy: 0.6548 - loss: 1.2770 - val_accuracy: 0.5383 - val_loss: 1.6904
Epoch 15/50
21/21            1s 43ms/step -
accuracy: 0.7103 - loss: 1.1797 - val_accuracy: 0.5940 - val_loss: 1.6094
Epoch 16/50
21/21            1s 42ms/step -
accuracy: 0.7115 - loss: 1.1876 - val_accuracy: 0.5824 - val_loss: 1.5863
Epoch 17/50
21/21            1s 57ms/step -
accuracy: 0.6915 - loss: 1.1704 - val_accuracy: 0.6589 - val_loss: 1.4618
Epoch 18/50
21/21            1s 54ms/step -
accuracy: 0.6988 - loss: 1.1164 - val_accuracy: 0.6450 - val_loss: 1.4769
Epoch 19/50
21/21            1s 48ms/step -
accuracy: 0.7325 - loss: 1.0445 - val_accuracy: 0.6659 - val_loss: 1.3893
Epoch 20/50
21/21            1s 47ms/step -
accuracy: 0.7659 - loss: 0.9465 - val_accuracy: 0.6705 - val_loss: 1.3289
Epoch 21/50
21/21            1s 51ms/step -
accuracy: 0.7902 - loss: 0.9112 - val_accuracy: 0.6891 - val_loss: 1.3146
```

```
Epoch 22/50
21/21            1s 65ms/step -
accuracy: 0.7614 - loss: 0.9063 - val_accuracy: 0.7007 - val_loss: 1.2697
Epoch 23/50
21/21            1s 49ms/step -
accuracy: 0.7620 - loss: 0.9016 - val_accuracy: 0.7100 - val_loss: 1.2721
Epoch 24/50
21/21            1s 47ms/step -
accuracy: 0.7773 - loss: 0.8694 - val_accuracy: 0.7030 - val_loss: 1.2506
Epoch 25/50
21/21            1s 44ms/step -
accuracy: 0.7913 - loss: 0.8297 - val_accuracy: 0.7123 - val_loss: 1.1681
Epoch 26/50
21/21            1s 46ms/step -
accuracy: 0.8036 - loss: 0.7996 - val_accuracy: 0.7169 - val_loss: 1.1608
Epoch 27/50
21/21            1s 45ms/step -
accuracy: 0.8100 - loss: 0.7757 - val_accuracy: 0.7216 - val_loss: 1.1485
Epoch 28/50
21/21            1s 45ms/step -
accuracy: 0.8071 - loss: 0.7417 - val_accuracy: 0.7471 - val_loss: 1.0838
Epoch 29/50
21/21            1s 44ms/step -
accuracy: 0.8169 - loss: 0.7273 - val_accuracy: 0.7262 - val_loss: 1.1255
Epoch 30/50
21/21            1s 43ms/step -
accuracy: 0.8228 - loss: 0.7218 - val_accuracy: 0.7355 - val_loss: 1.1367
Epoch 31/50
21/21            1s 43ms/step -
accuracy: 0.8271 - loss: 0.6905 - val_accuracy: 0.7309 - val_loss: 1.0870
Epoch 32/50
21/21            1s 43ms/step -
accuracy: 0.8255 - loss: 0.6749 - val_accuracy: 0.7494 - val_loss: 1.0880
Epoch 33/50
21/21            1s 44ms/step -
accuracy: 0.8142 - loss: 0.7123 - val_accuracy: 0.7633 - val_loss: 1.0228
Epoch 34/50
21/21            1s 44ms/step -
accuracy: 0.8533 - loss: 0.6150 - val_accuracy: 0.7749 - val_loss: 0.9546
Epoch 35/50
21/21            1s 43ms/step -
accuracy: 0.8464 - loss: 0.6444 - val_accuracy: 0.7309 - val_loss: 1.0202
Epoch 36/50
21/21            1s 45ms/step -
accuracy: 0.8526 - loss: 0.5876 - val_accuracy: 0.7378 - val_loss: 1.0135
Epoch 37/50
21/21            1s 43ms/step -
accuracy: 0.8512 - loss: 0.6348 - val_accuracy: 0.7773 - val_loss: 0.9735
```

```
Epoch 38/50
21/21            1s 45ms/step -
accuracy: 0.8453 - loss: 0.5934 - val_accuracy: 0.7749 - val_loss: 0.9703
Epoch 39/50
21/21            1s 44ms/step -
accuracy: 0.8602 - loss: 0.5667 - val_accuracy: 0.7865 - val_loss: 0.9553
Epoch 40/50
21/21            1s 45ms/step -
accuracy: 0.8859 - loss: 0.5046 - val_accuracy: 0.7749 - val_loss: 0.9311
Epoch 41/50
21/21            1s 55ms/step -
accuracy: 0.8709 - loss: 0.5081 - val_accuracy: 0.7471 - val_loss: 0.9622
Epoch 42/50
21/21            1s 57ms/step -
accuracy: 0.8831 - loss: 0.4775 - val_accuracy: 0.7796 - val_loss: 0.9074
Epoch 43/50
21/21            1s 54ms/step -
accuracy: 0.8782 - loss: 0.4810 - val_accuracy: 0.7796 - val_loss: 0.9248
Epoch 44/50
21/21            1s 49ms/step -
accuracy: 0.8776 - loss: 0.4814 - val_accuracy: 0.7633 - val_loss: 0.8813
Epoch 45/50
21/21            1s 57ms/step -
accuracy: 0.8833 - loss: 0.4590 - val_accuracy: 0.7517 - val_loss: 0.9166
Epoch 46/50
21/21            1s 46ms/step -
accuracy: 0.8846 - loss: 0.4570 - val_accuracy: 0.7749 - val_loss: 0.9062
Epoch 47/50
21/21            1s 44ms/step -
accuracy: 0.8941 - loss: 0.4429 - val_accuracy: 0.7587 - val_loss: 0.9294
Epoch 48/50
21/21            1s 48ms/step -
accuracy: 0.8916 - loss: 0.4564 - val_accuracy: 0.7425 - val_loss: 0.9306
Epoch 49/50
21/21            1s 51ms/step -
accuracy: 0.8940 - loss: 0.4083 - val_accuracy: 0.7773 - val_loss: 0.8769
Epoch 50/50
21/21            1s 45ms/step -
accuracy: 0.8845 - loss: 0.4382 - val_accuracy: 0.7541 - val_loss: 0.9048
```

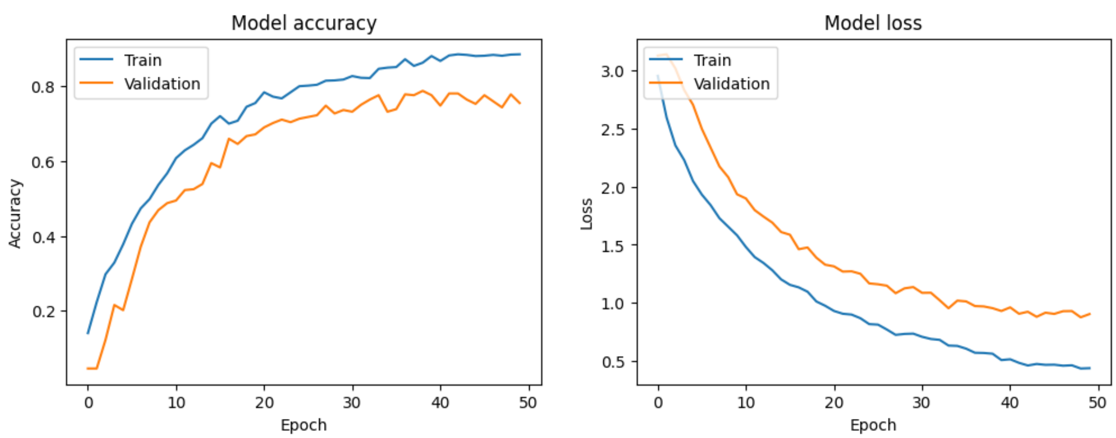
[14]:

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

```
7/7            0s 9ms/step -
accuracy: 0.7872 - loss: 0.8720
Test Accuracy: 0.7981
Test Loss: 0.8176
```

[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

         baca      0.933     0.875     0.903        16
        bantu      0.800     0.923     0.857        13
        bapak      0.778     0.467     0.583        15
buangairkecil      1.000     0.750     0.857         8
         buat      0.944     1.000     0.971        17
         halo      1.000     0.850     0.919        20
          ibu      1.000     0.333     0.500         6
         kamu      0.708     0.773     0.739        22
         maaf      1.000     0.857     0.923        21
        makan      0.923     0.706     0.800        17
          mau      0.913     1.000     0.955        21
         nama      0.700     0.840     0.764        25
         pagi      0.909     0.833     0.870        24
        paham      0.889     0.960     0.923        25
        sakit      1.000     0.750     0.857         4
    sama-sama      1.000     0.857     0.923        28
         saya      0.533     0.615     0.571        13
      selamat      0.833     0.714     0.769        21
        siapa      0.325     0.812     0.464        16
        tanya      0.778     0.700     0.737        20
       tempat      0.500     0.375     0.429         8
  terima-kasih     0.696     0.842     0.762        19
    terlambat      0.722     0.765     0.743        17
        tidak      1.000     0.647     0.786        17
       tolong      1.000     0.889     0.941        18

     accuracy                          0.798       431
```

macro avg	0.835	0.765	0.782	431
weighted avg	0.839	0.798	0.806	431