COMPARISON_MediaPipe+CNN+LSTM

June 21, 2025

[18]:
```
from modules.SignLanguageProcessor import load_and_preprocess_data,parse_frame
import os
```

[19]:
```
ROOT_PATH = ''
sequences,labels,label_map = load_and_preprocess_data(os.path.
 ↪join(ROOT_PATH,'data'))
```

[20]:
```
num_classes = len(label_map)
```

[21]:
```
len(labels)
```

[21]: 3413

[22]:
```
sequences.shape
```

[22]: (3413, 3, 61, 3)

[23]:
```
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
)
```

[24]:
```
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
```

[25]:
```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
```

[26]:
```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)
```

```
(2047, 3, 61, 3, 1)
(2047,)
```

[27]:
```python
input_shape = X_train_cnn.shape[1:]
print(input_shape)
```

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

[28]:
```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)
```

[29]:
```
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'])
```

[30]:
```
history = model.fit(train_ds,validation_data=val_ds, epochs=50, batch_size=64)
```

```
Epoch 1/50
32/32              6s 71ms/step -
accuracy: 0.0712 - loss: 3.6093 - val_accuracy: 0.1069 - val_loss: 3.6128
Epoch 2/50
32/32              2s 54ms/step -
accuracy: 0.1152 - loss: 3.1231 - val_accuracy: 0.1537 - val_loss: 3.3119
Epoch 3/50
32/32              2s 53ms/step -
accuracy: 0.1489 - loss: 2.9579 - val_accuracy: 0.1552 - val_loss: 3.0274
Epoch 4/50
32/32              2s 54ms/step -
accuracy: 0.1756 - loss: 2.8015 - val_accuracy: 0.1933 - val_loss: 2.8193
Epoch 5/50
32/32              2s 52ms/step -
accuracy: 0.2241 - loss: 2.7012 - val_accuracy: 0.2372 - val_loss: 2.7337
```

```
Epoch 6/50
32/32             2s 52ms/step -
accuracy: 0.2613 - loss: 2.5861 - val_accuracy: 0.2972 - val_loss: 2.6101
Epoch 7/50
32/32             2s 48ms/step -
accuracy: 0.2631 - loss: 2.5559 - val_accuracy: 0.3382 - val_loss: 2.5513
Epoch 8/50
32/32             2s 51ms/step -
accuracy: 0.3352 - loss: 2.4166 - val_accuracy: 0.3777 - val_loss: 2.4247
Epoch 9/50
32/32             2s 50ms/step -
accuracy: 0.3638 - loss: 2.2752 - val_accuracy: 0.3441 - val_loss: 2.4053
Epoch 10/50
32/32             2s 48ms/step -
accuracy: 0.3673 - loss: 2.2314 - val_accuracy: 0.3865 - val_loss: 2.3214
Epoch 11/50
32/32             2s 49ms/step -
accuracy: 0.4153 - loss: 2.1264 - val_accuracy: 0.3836 - val_loss: 2.2909
Epoch 12/50
32/32             2s 49ms/step -
accuracy: 0.4595 - loss: 2.0062 - val_accuracy: 0.4100 - val_loss: 2.2592
Epoch 13/50
32/32             2s 49ms/step -
accuracy: 0.4510 - loss: 1.9713 - val_accuracy: 0.4905 - val_loss: 2.1077
Epoch 14/50
32/32             2s 49ms/step -
accuracy: 0.5053 - loss: 1.8851 - val_accuracy: 0.5124 - val_loss: 2.0930
Epoch 15/50
32/32             2s 48ms/step -
accuracy: 0.5136 - loss: 1.7895 - val_accuracy: 0.4846 - val_loss: 2.1556
Epoch 16/50
32/32             2s 49ms/step -
accuracy: 0.5522 - loss: 1.7098 - val_accuracy: 0.5300 - val_loss: 2.0626
Epoch 17/50
32/32             2s 48ms/step -
accuracy: 0.5491 - loss: 1.6804 - val_accuracy: 0.5081 - val_loss: 2.0680
Epoch 18/50
32/32             2s 48ms/step -
accuracy: 0.5857 - loss: 1.5507 - val_accuracy: 0.5447 - val_loss: 1.9993
Epoch 19/50
32/32             2s 48ms/step -
accuracy: 0.6256 - loss: 1.4825 - val_accuracy: 0.6135 - val_loss: 1.8325
Epoch 20/50
32/32             2s 49ms/step -
accuracy: 0.6331 - loss: 1.4369 - val_accuracy: 0.6032 - val_loss: 1.8713
Epoch 21/50
32/32             2s 50ms/step -
accuracy: 0.6275 - loss: 1.4572 - val_accuracy: 0.5944 - val_loss: 1.7713
```

```
Epoch 22/50
32/32            2s 49ms/step -
accuracy: 0.6691 - loss: 1.3031 - val_accuracy: 0.6076 - val_loss: 1.8070
Epoch 23/50
32/32            2s 49ms/step -
accuracy: 0.6976 - loss: 1.2281 - val_accuracy: 0.6457 - val_loss: 1.7261
Epoch 24/50
32/32            2s 49ms/step -
accuracy: 0.6864 - loss: 1.2291 - val_accuracy: 0.6193 - val_loss: 1.7012
Epoch 25/50
32/32            2s 49ms/step -
accuracy: 0.7164 - loss: 1.1419 - val_accuracy: 0.6764 - val_loss: 1.6196
Epoch 26/50
32/32            2s 48ms/step -
accuracy: 0.7444 - loss: 1.0669 - val_accuracy: 0.6603 - val_loss: 1.5814
Epoch 27/50
32/32            2s 49ms/step -
accuracy: 0.7472 - loss: 1.0222 - val_accuracy: 0.6135 - val_loss: 1.6472
Epoch 28/50
32/32            2s 49ms/step -
accuracy: 0.7637 - loss: 1.0055 - val_accuracy: 0.7028 - val_loss: 1.3802
Epoch 29/50
32/32            2s 50ms/step -
accuracy: 0.7529 - loss: 1.0032 - val_accuracy: 0.7013 - val_loss: 1.3752
Epoch 30/50
32/32            2s 48ms/step -
accuracy: 0.7921 - loss: 0.8948 - val_accuracy: 0.7233 - val_loss: 1.3757
Epoch 31/50
32/32            2s 49ms/step -
accuracy: 0.7946 - loss: 0.8668 - val_accuracy: 0.7174 - val_loss: 1.3613
Epoch 32/50
32/32            2s 49ms/step -
accuracy: 0.7878 - loss: 0.8528 - val_accuracy: 0.7482 - val_loss: 1.2329
Epoch 33/50
32/32            2s 49ms/step -
accuracy: 0.8085 - loss: 0.7934 - val_accuracy: 0.6999 - val_loss: 1.4288
Epoch 34/50
32/32            2s 57ms/step -
accuracy: 0.8188 - loss: 0.7503 - val_accuracy: 0.7570 - val_loss: 1.1362
Epoch 35/50
32/32            2s 54ms/step -
accuracy: 0.8125 - loss: 0.7429 - val_accuracy: 0.7160 - val_loss: 1.3110
Epoch 36/50
32/32            2s 54ms/step -
accuracy: 0.8319 - loss: 0.7239 - val_accuracy: 0.7657 - val_loss: 1.2184
Epoch 37/50
32/32            2s 54ms/step -
accuracy: 0.8471 - loss: 0.6726 - val_accuracy: 0.7716 - val_loss: 1.1800
```

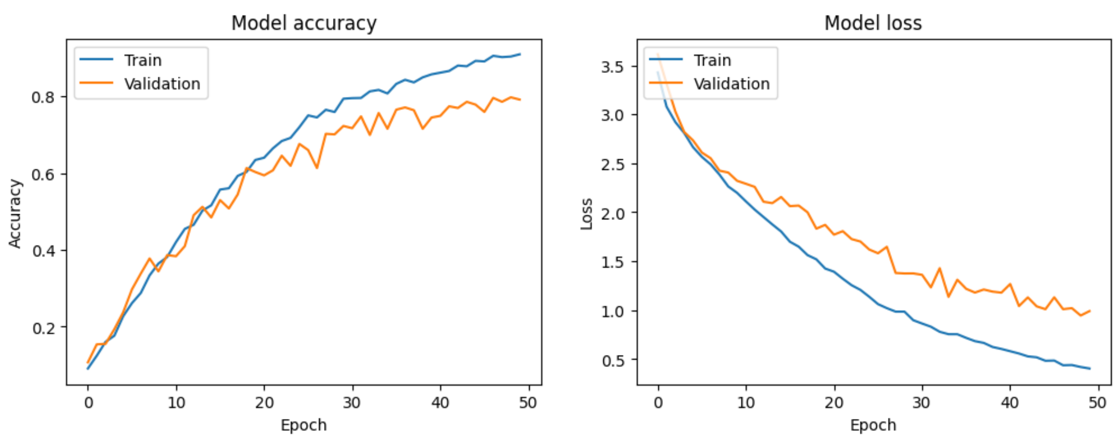
```
Epoch 38/50
32/32              2s 52ms/step -
accuracy: 0.8313 - loss: 0.6721 - val_accuracy: 0.7643 - val_loss: 1.2114
Epoch 39/50
32/32              2s 51ms/step -
accuracy: 0.8464 - loss: 0.6331 - val_accuracy: 0.7160 - val_loss: 1.1898
Epoch 40/50
32/32              2s 53ms/step -
accuracy: 0.8556 - loss: 0.5981 - val_accuracy: 0.7452 - val_loss: 1.1791
Epoch 41/50
32/32              2s 52ms/step -
accuracy: 0.8494 - loss: 0.6114 - val_accuracy: 0.7496 - val_loss: 1.2675
Epoch 42/50
32/32              2s 53ms/step -
accuracy: 0.8714 - loss: 0.5451 - val_accuracy: 0.7745 - val_loss: 1.0424
Epoch 43/50
32/32              2s 52ms/step -
accuracy: 0.8830 - loss: 0.5088 - val_accuracy: 0.7701 - val_loss: 1.1306
Epoch 44/50
32/32              2s 53ms/step -
accuracy: 0.8862 - loss: 0.5115 - val_accuracy: 0.7862 - val_loss: 1.0396
Epoch 45/50
32/32              2s 52ms/step -
accuracy: 0.8987 - loss: 0.4726 - val_accuracy: 0.7789 - val_loss: 1.0102
Epoch 46/50
32/32              2s 52ms/step -
accuracy: 0.8887 - loss: 0.4885 - val_accuracy: 0.7599 - val_loss: 1.1317
Epoch 47/50
32/32              2s 54ms/step -
accuracy: 0.9171 - loss: 0.4109 - val_accuracy: 0.7965 - val_loss: 1.0105
Epoch 48/50
32/32              2s 51ms/step -
accuracy: 0.8995 - loss: 0.4424 - val_accuracy: 0.7862 - val_loss: 1.0209
Epoch 49/50
32/32              2s 52ms/step -
accuracy: 0.8997 - loss: 0.4203 - val_accuracy: 0.7980 - val_loss: 0.9457
Epoch 50/50
32/32              2s 51ms/step -
accuracy: 0.9131 - loss: 0.3951 - val_accuracy: 0.7921 - val_loss: 0.9908
```

[31]:
```
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: 0.7747 - loss: 1.0272
Test Accuracy: 0.7672
Test Loss: 1.0240
```

[32]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
```

[33]:
```
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()
```



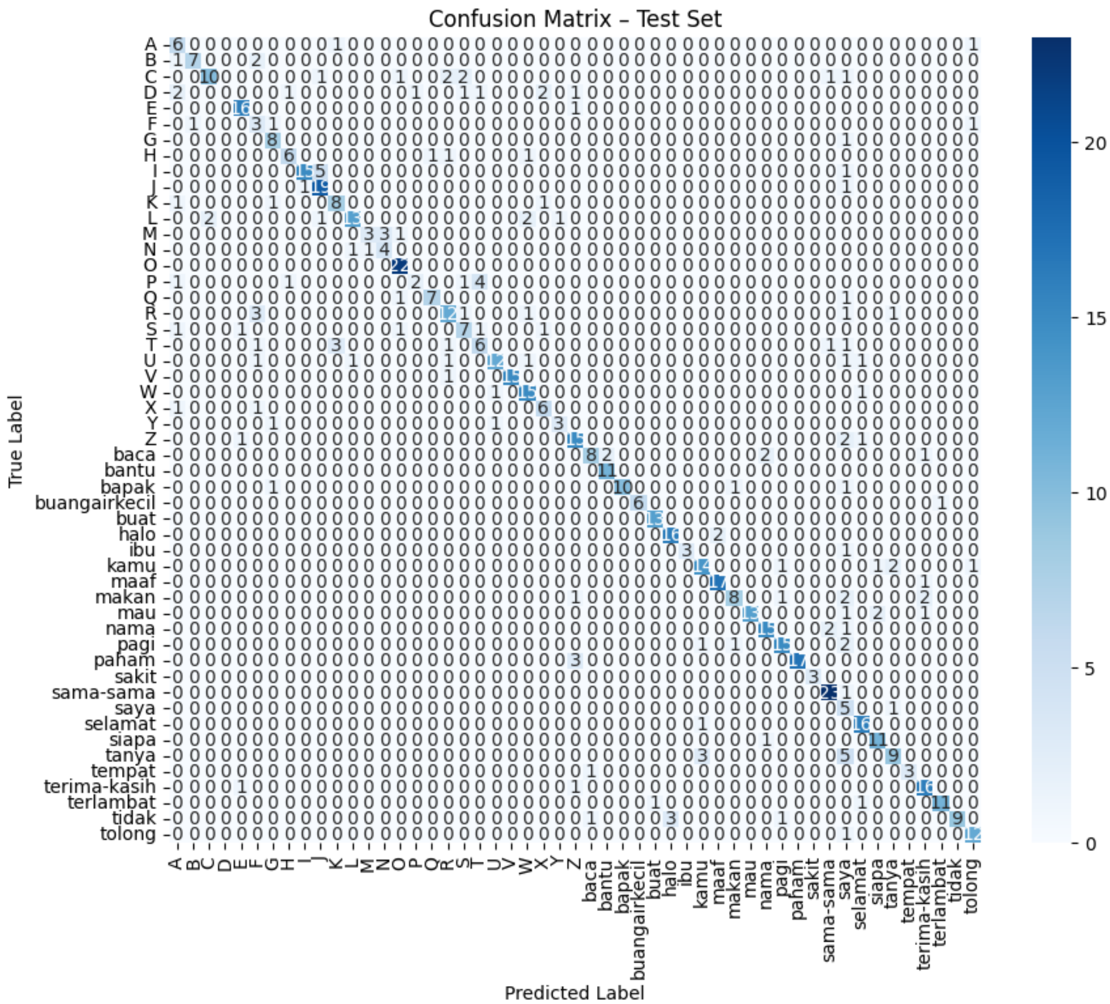
[34]:
```
y_true, y_pred = [], []
target_names = [label_map[i] for i in range(len(label_map))]
for X_batch, y_batch in test_ds:
    y_true.append(y_batch.numpy())

    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()
```

```
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\chris\.conda\envs\ASLR\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

	precision	recall	f1-score	support
A	0.462	0.750	0.571	8
B	0.875	0.700	0.778	10
C	0.833	0.556	0.667	18
D	0.000	0.000	0.000	9
E	0.842	0.941	0.889	17
F	0.273	0.500	0.353	6
G	0.667	0.889	0.762	9
H	0.750	0.667	0.706	9
I	0.938	0.714	0.811	21
J	0.731	0.905	0.809	21
K	0.667	0.727	0.696	11

L	0.867	0.684	0.765	19
M	0.750	0.429	0.545	7
N	0.571	0.667	0.615	6
O	0.846	1.000	0.917	22
P	0.667	0.222	0.333	9
Q	0.875	0.778	0.824	9
R	0.667	0.632	0.649	19
S	0.583	0.583	0.583	12
T	0.500	0.462	0.480	13
U	0.857	0.667	0.750	18
V	1.000	0.938	0.968	16
W	0.750	0.882	0.811	17
X	0.600	0.750	0.667	8
Y	0.750	0.600	0.667	5
Z	0.682	0.789	0.732	19
baca	0.800	0.615	0.696	13
bantu	0.846	1.000	0.917	11
bapak	1.000	0.769	0.870	13
buangairkecil	1.000	0.857	0.923	7
buat	0.929	1.000	0.963	13
halo	0.842	0.889	0.865	18
ibu	1.000	0.750	0.857	4
kamu	0.737	0.737	0.737	19
maaf	0.895	0.944	0.919	18
makan	0.800	0.571	0.667	14
mau	1.000	0.765	0.867	17
nama	0.833	0.833	0.833	18
pagi	0.833	0.789	0.811	19
paham	1.000	0.850	0.919	20
sakit	1.000	1.000	1.000	3
sama-sama	0.852	0.958	0.902	24
saya	0.167	0.833	0.278	6
selamat	0.800	0.941	0.865	17
siapa	0.786	0.917	0.846	12
tanya	0.692	0.529	0.600	17
tempat	1.000	0.750	0.857	4
terima-kasih	0.762	0.889	0.821	18
terlambat	0.917	0.846	0.880	13
tidak	1.000	0.643	0.783	14
tolong	0.800	0.923	0.857	13
accuracy			0.767	683
macro avg	0.770	0.746	0.743	683
weighted avg	0.793	0.767	0.768	683


Confusion Matrix – Test Set
True Label
Predicted Label