COMPARISON_MediaPipe+CNN

June 21, 2025

Paper Reference : https://j-innovative.org/index.php/Innovative/article/download/15199/10372/26113

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

[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):
    """
    Reshape a dataset of (N, F, L, T) into (N*T, L, F, 1) for Conv2D,
    where each frame becomes its own sample.

    Parameters:
    - X: np.ndarray of shape (N, F, L, T)
    - y: np.ndarray of shape (N,)

    Returns:
    - reshaped_X: np.ndarray of shape (N*T, L, F, 1)
    - reshaped_y: np.ndarray of shape (N*T,)
    """
    reshaped_X = []
    reshaped_y = []

    for sample, label in zip(X, y):
        T = sample.shape[-1]
        for t in range(T):
            frame = sample[:, :, t].T[..., np.newaxis]
            reshaped_X.append(frame)
            reshaped_y.append(label)

    reshaped_X = np.array(reshaped_X)
    reshaped_y = np.array(reshaped_y)
    return reshaped_X, reshaped_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)
```

```
(3879, 61, 3, 1)
(3879,)
```

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

```
(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 Conv2D, MaxPooling2D, Dropout, Flatten,␣
 ↪Dense, BatchNormalization,Input

cnn_model = Sequential([
    Input(input_shape),
    Conv2D(32, (3, 2), activation='relu', padding='same'),
    MaxPooling2D((2, 1)),
    Dropout(0.25),
    Conv2D(64, (3, 2), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 1)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(num_classes, activation='softmax')
```

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

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

```
Epoch 1/50
61/61             2s 11ms/step -
accuracy: 0.0944 - loss: 3.1221 - val_accuracy: 0.1462 - val_loss: 2.9218
Epoch 2/50
61/61             1s 8ms/step -
accuracy: 0.1402 - loss: 2.8520 - val_accuracy: 0.1895 - val_loss: 2.6704
Epoch 3/50
61/61             1s 8ms/step -
accuracy: 0.1794 - loss: 2.6196 - val_accuracy: 0.2390 - val_loss: 2.5055
Epoch 4/50
61/61             1s 8ms/step -
accuracy: 0.2369 - loss: 2.4423 - val_accuracy: 0.2962 - val_loss: 2.3516
Epoch 5/50
61/61             1s 8ms/step -
accuracy: 0.2607 - loss: 2.2877 - val_accuracy: 0.3364 - val_loss: 2.2037
Epoch 6/50
61/61             1s 8ms/step -
accuracy: 0.3066 - loss: 2.1756 - val_accuracy: 0.3735 - val_loss: 2.1106
Epoch 7/50
61/61             1s 8ms/step -
accuracy: 0.3574 - loss: 2.0495 - val_accuracy: 0.4269 - val_loss: 1.9843
Epoch 8/50
61/61             1s 8ms/step -
accuracy: 0.3685 - loss: 1.9700 - val_accuracy: 0.4408 - val_loss: 1.9089
Epoch 9/50
61/61             1s 8ms/step -
accuracy: 0.3962 - loss: 1.9152 - val_accuracy: 0.4493 - val_loss: 1.8546
Epoch 10/50
61/61             1s 8ms/step -
accuracy: 0.4177 - loss: 1.8193 - val_accuracy: 0.4671 - val_loss: 1.7835
Epoch 11/50
61/61             1s 8ms/step -
accuracy: 0.4361 - loss: 1.7723 - val_accuracy: 0.4803 - val_loss: 1.7292
Epoch 12/50
61/61             1s 9ms/step -
accuracy: 0.4513 - loss: 1.6990 - val_accuracy: 0.4849 - val_loss: 1.7000
Epoch 13/50
61/61             1s 8ms/step -
accuracy: 0.4745 - loss: 1.6528 - val_accuracy: 0.5012 - val_loss: 1.6480
Epoch 14/50
61/61             1s 8ms/step -
```

```
accuracy: 0.4690 - loss: 1.6392 - val_accuracy: 0.5135 - val_loss: 1.6139
Epoch 15/50
61/61             1s 9ms/step -
accuracy: 0.4832 - loss: 1.6249 - val_accuracy: 0.5189 - val_loss: 1.6078
Epoch 16/50
61/61             1s 8ms/step -
accuracy: 0.4776 - loss: 1.5888 - val_accuracy: 0.5244 - val_loss: 1.5700
Epoch 17/50
61/61             1s 8ms/step -
accuracy: 0.5168 - loss: 1.4998 - val_accuracy: 0.5406 - val_loss: 1.5525
Epoch 18/50
61/61             1s 8ms/step -
accuracy: 0.5208 - loss: 1.4861 - val_accuracy: 0.5375 - val_loss: 1.5221
Epoch 19/50
61/61             1s 8ms/step -
accuracy: 0.5174 - loss: 1.4808 - val_accuracy: 0.5561 - val_loss: 1.5189
Epoch 20/50
61/61             1s 8ms/step -
accuracy: 0.5272 - loss: 1.4402 - val_accuracy: 0.5568 - val_loss: 1.4875
Epoch 21/50
61/61             1s 8ms/step -
accuracy: 0.5294 - loss: 1.4544 - val_accuracy: 0.5398 - val_loss: 1.5042
Epoch 22/50
61/61             1s 8ms/step -
accuracy: 0.5390 - loss: 1.4122 - val_accuracy: 0.5584 - val_loss: 1.4737
Epoch 23/50
61/61             1s 8ms/step -
accuracy: 0.5441 - loss: 1.3911 - val_accuracy: 0.5746 - val_loss: 1.4269
Epoch 24/50
61/61             1s 8ms/step -
accuracy: 0.5494 - loss: 1.3830 - val_accuracy: 0.5692 - val_loss: 1.4396
Epoch 25/50
61/61             1s 8ms/step -
accuracy: 0.5576 - loss: 1.3429 - val_accuracy: 0.5816 - val_loss: 1.4305
Epoch 26/50
61/61             1s 8ms/step -
accuracy: 0.5617 - loss: 1.3373 - val_accuracy: 0.5947 - val_loss: 1.3913
Epoch 27/50
61/61             1s 8ms/step -
accuracy: 0.5638 - loss: 1.3204 - val_accuracy: 0.5831 - val_loss: 1.4120
Epoch 28/50
61/61             1s 9ms/step -
accuracy: 0.5560 - loss: 1.3466 - val_accuracy: 0.5800 - val_loss: 1.3849
Epoch 29/50
61/61             1s 8ms/step -
accuracy: 0.5674 - loss: 1.2905 - val_accuracy: 0.5770 - val_loss: 1.4030
Epoch 30/50
61/61             1s 9ms/step -
```

```
accuracy: 0.5755 - loss: 1.2993 - val_accuracy: 0.5924 - val_loss: 1.3725
Epoch 31/50
61/61             1s 9ms/step -
accuracy: 0.5853 - loss: 1.2582 - val_accuracy: 0.5955 - val_loss: 1.3694
Epoch 32/50
61/61             1s 9ms/step -
accuracy: 0.5836 - loss: 1.2562 - val_accuracy: 0.5932 - val_loss: 1.3765
Epoch 33/50
61/61             1s 10ms/step -
accuracy: 0.5913 - loss: 1.2813 - val_accuracy: 0.5839 - val_loss: 1.3677
Epoch 34/50
61/61             1s 9ms/step -
accuracy: 0.5907 - loss: 1.2530 - val_accuracy: 0.5963 - val_loss: 1.3554
Epoch 35/50
61/61             1s 9ms/step -
accuracy: 0.5895 - loss: 1.2547 - val_accuracy: 0.5978 - val_loss: 1.3670
Epoch 36/50
61/61             1s 9ms/step -
accuracy: 0.6120 - loss: 1.2001 - val_accuracy: 0.5878 - val_loss: 1.3552
Epoch 37/50
61/61             1s 9ms/step -
accuracy: 0.6060 - loss: 1.2010 - val_accuracy: 0.5847 - val_loss: 1.3560
Epoch 38/50
61/61             1s 9ms/step -
accuracy: 0.5953 - loss: 1.2255 - val_accuracy: 0.5963 - val_loss: 1.3282
Epoch 39/50
61/61             1s 9ms/step -
accuracy: 0.5986 - loss: 1.2088 - val_accuracy: 0.6118 - val_loss: 1.3377
Epoch 40/50
61/61             1s 9ms/step -
accuracy: 0.6094 - loss: 1.1641 - val_accuracy: 0.6063 - val_loss: 1.3134
Epoch 41/50
61/61             1s 9ms/step -
accuracy: 0.6177 - loss: 1.1788 - val_accuracy: 0.5978 - val_loss: 1.3250
Epoch 42/50
61/61             1s 10ms/step -
accuracy: 0.6157 - loss: 1.1529 - val_accuracy: 0.6040 - val_loss: 1.3198
Epoch 43/50
61/61             1s 9ms/step -
accuracy: 0.6213 - loss: 1.1615 - val_accuracy: 0.6025 - val_loss: 1.3356
Epoch 44/50
61/61             1s 9ms/step -
accuracy: 0.6175 - loss: 1.1525 - val_accuracy: 0.6071 - val_loss: 1.3095
Epoch 45/50
61/61             1s 9ms/step -
accuracy: 0.6311 - loss: 1.1417 - val_accuracy: 0.6094 - val_loss: 1.3121
Epoch 46/50
61/61             1s 9ms/step -
```

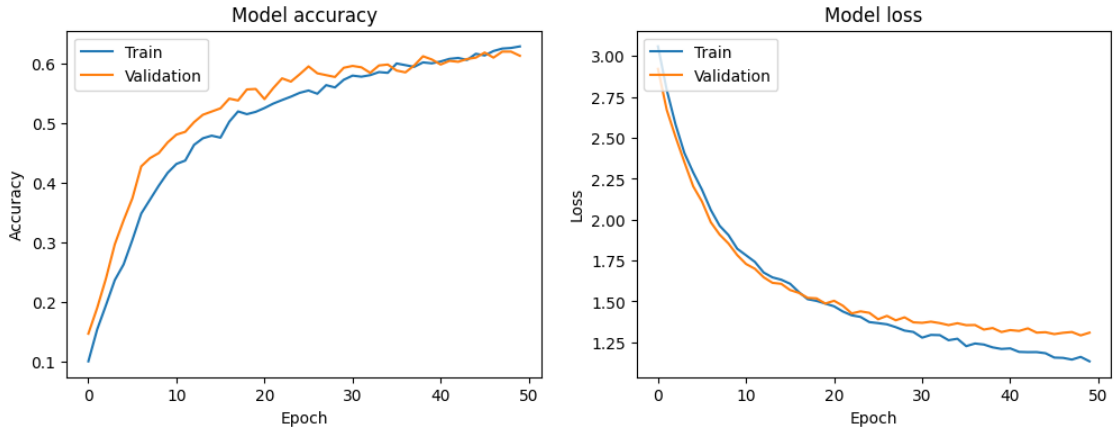
```
accuracy: 0.6288 - loss: 1.1348 - val_accuracy: 0.6179 - val_loss: 1.3005
Epoch 47/50
61/61             1s 8ms/step -
accuracy: 0.6262 - loss: 1.1252 - val_accuracy: 0.6094 - val_loss: 1.3085
Epoch 48/50
61/61             1s 9ms/step -
accuracy: 0.6340 - loss: 1.1144 - val_accuracy: 0.6195 - val_loss: 1.3133
Epoch 49/50
61/61             1s 9ms/step -
accuracy: 0.6289 - loss: 1.1444 - val_accuracy: 0.6195 - val_loss: 1.2926
Epoch 50/50
61/61             1s 9ms/step -
accuracy: 0.6447 - loss: 1.0889 - val_accuracy: 0.6125 - val_loss: 1.3092
```

[14]:
```
test_loss, test_accuracy = cnn_model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
21/21             0s 3ms/step -
accuracy: 0.6347 - loss: 1.4002
Test Accuracy: 0.6326
Test Loss: 1.2936
```

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

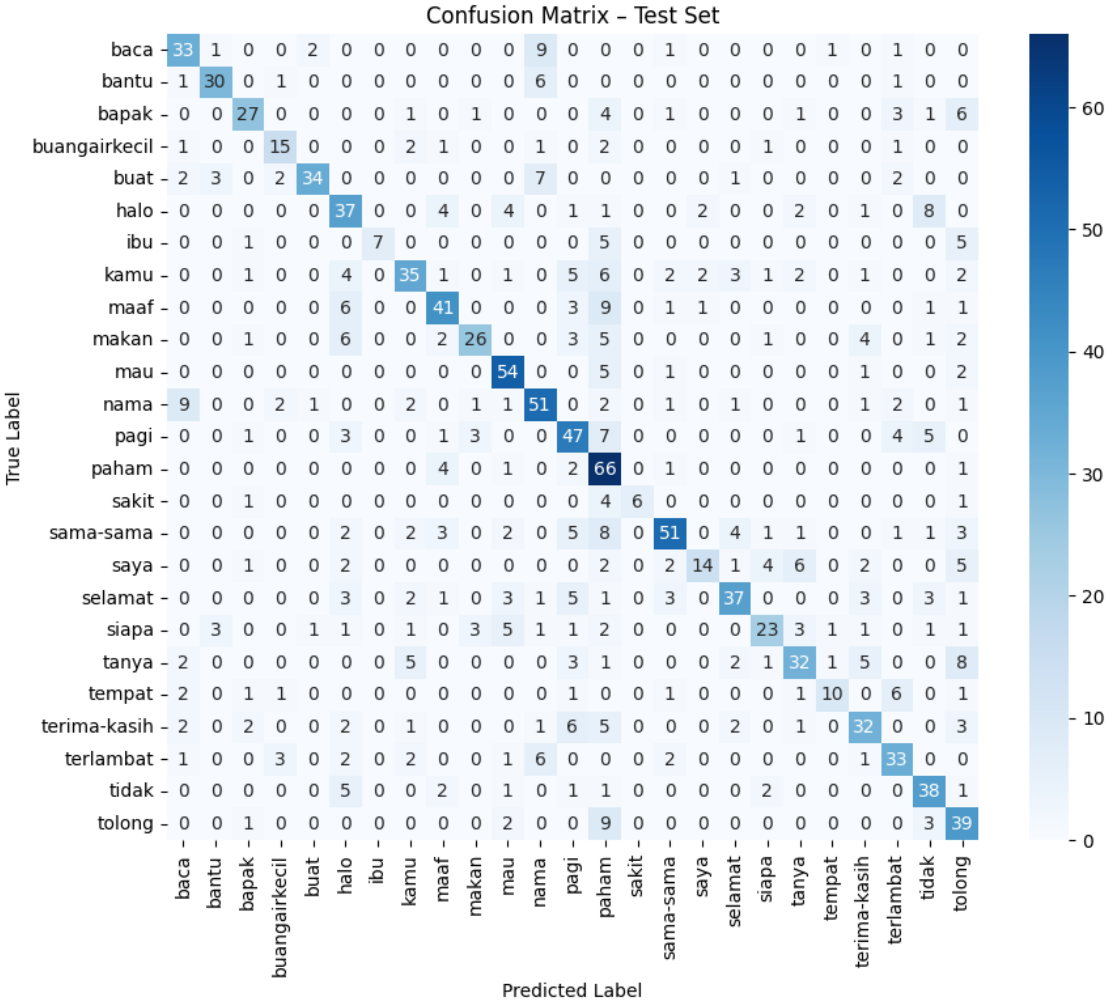
[16]:
```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
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 = cnn_model.predict(X_batch, verbose=0)
    y_pred.append(np.argmax(batch_pred, axis=1))

y_true = np.concatenate(y_true)
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      0.623     0.688     0.653        48
       bantu      0.811     0.769     0.789        39
```

bapak	0.730	0.600	0.659	45
buangairkecil	0.625	0.625	0.625	24
buat	0.895	0.667	0.764	51
halo	0.507	0.617	0.556	60
ibu	1.000	0.389	0.560	18
kamu	0.660	0.530	0.588	66
maaf	0.683	0.651	0.667	63
makan	0.765	0.510	0.612	51
mau	0.720	0.857	0.783	63
nama	0.614	0.680	0.646	75
pagi	0.566	0.653	0.606	72
paham	0.455	0.880	0.600	75
sakit	1.000	0.500	0.667	12
sama-sama	0.761	0.607	0.675	84
saya	0.737	0.359	0.483	39
selamat	0.725	0.587	0.649	63
siapa	0.676	0.479	0.561	48
tanya	0.640	0.533	0.582	60
tempat	0.769	0.417	0.541	24
terima-kasih	0.615	0.561	0.587	57
terlambat	0.611	0.647	0.629	51
tidak	0.613	0.745	0.673	51
tolong	0.470	0.722	0.569	54
accuracy			0.633	1293
macro avg	0.691	0.611	0.629	1293
weighted avg	0.664	0.633	0.632	1293


Confusion Matrix – Test Set
True Label
Predicted Label
baca 33 1 0 0 2 0 0 0 0 0 0 9 0 0 0 1 0 0 0 0 1 0 1 0 0
bantu 1 30 0 1 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 1 0 0
bapak 0 0 27 0 0 0 0 1 0 1 0 0 0 4 0 1 0 0 0 1 0 0 3 1 6
buangairkecil 1 0 0 15 0 0 0 2 1 0 0 1 0 2 0 0 0 0 1 0 0 0 1 0 0
buat 2 3 0 2 34 0 0 0 0 0 0 7 0 0 0 0 0 1 0 0 0 0 2 0 0
halo 0 0 0 0 0 37 0 0 4 0 4 0 1 1 0 0 2 0 0 2 0 1 0 8 0
ibu 0 0 1 0 0 0 7 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 5
kamu 0 0 1 0 0 4 0 35 1 0 1 0 5 6 0 2 2 3 1 2 0 1 0 0 2
maaf 0 0 0 0 0 6 0 0 41 0 0 0 3 9 0 1 1 0 0 0 0 0 0 1 1
makan 0 0 1 0 0 6 0 0 2 26 0 0 3 5 0 0 0 0 1 0 0 4 0 1 2
mau 0 0 0 0 0 0 0 0 0 0 54 0 0 5 0 1 0 0 0 0 0 1 0 0 2
nama 9 0 0 2 1 0 0 2 0 1 1 51 0 2 0 1 0 1 0 0 0 1 2 0 1
pagi 0 0 1 0 0 3 0 0 1 3 0 0 47 7 0 0 0 0 0 1 0 0 4 5 0
paham 0 0 0 0 0 0 0 0 4 0 1 0 2 66 0 1 0 0 0 0 0 0 0 0 1
sakit 0 0 1 0 0 0 0 0 0 0 0 0 0 4 6 0 0 0 0 0 0 0 0 0 1
sama-sama 0 0 0 0 0 2 0 2 3 0 2 0 5 8 0 51 0 4 1 1 0 0 1 1 3
saya 0 0 1 0 0 2 0 0 0 0 0 0 0 2 0 2 14 1 4 6 0 2 0 0 5
selamat 0 0 0 0 0 3 0 2 1 0 3 1 5 1 0 3 0 37 0 0 0 3 0 3 1
siapa 0 3 0 0 1 1 0 1 0 3 5 1 1 2 0 0 0 0 23 3 1 1 0 1 1
tanya 2 0 0 0 0 0 0 5 0 0 0 0 3 1 0 0 0 2 1 32 1 5 0 0 8
tempat 2 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 10 0 6 0 1
terima-kasih 2 0 2 0 0 2 0 1 0 0 0 1 6 5 0 0 0 2 0 1 0 32 0 0 3
terlambat 1 0 0 3 0 2 0 2 0 0 1 6 0 0 0 2 0 0 0 0 0 1 33 0 0
tidak 0 0 0 0 0 5 0 0 2 0 1 0 1 1 0 0 0 0 2 0 0 0 0 38 1
tolong 0 0 1 0 0 0 0 0 0 0 2 0 0 9 0 0 0 0 0 0 0 0 0 3 39
baca bantu bapak buangairkecil buat halo ibu kamu maaf makan mau nama pagi paham sakit sama-sama saya selamat siapa tanya tempat terima-kasih terlambat tidak tolong
60
50
40
30
20
10
0