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]:
```
1722
```

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

[5]:
```
(1722, 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
```

```python
    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]:
```python
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)
```

```
(3099, 61, 3, 1)
(3099,)
```

[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
49/49                2s 11ms/step -
accuracy: 0.0831 - loss: 3.1252 - val_accuracy: 0.1008 - val_loss: 2.9760
Epoch 2/50
49/49                0s 8ms/step -
accuracy: 0.1255 - loss: 2.9029 - val_accuracy: 0.1890 - val_loss: 2.7493
Epoch 3/50
49/49                0s 8ms/step -
accuracy: 0.1708 - loss: 2.6864 - val_accuracy: 0.2306 - val_loss: 2.5316
Epoch 4/50
49/49                0s 8ms/step -
accuracy: 0.2185 - loss: 2.5100 - val_accuracy: 0.2762 - val_loss: 2.3672
Epoch 5/50
49/49                0s 8ms/step -
accuracy: 0.2348 - loss: 2.3699 - val_accuracy: 0.2839 - val_loss: 2.2494
Epoch 6/50
49/49                0s 8ms/step -
accuracy: 0.2779 - loss: 2.2674 - val_accuracy: 0.3566 - val_loss: 2.1118
Epoch 7/50
49/49                0s 8ms/step -
accuracy: 0.3249 - loss: 2.1291 - val_accuracy: 0.3711 - val_loss: 2.0675
Epoch 8/50
49/49                0s 8ms/step -
accuracy: 0.3489 - loss: 2.0297 - val_accuracy: 0.3895 - val_loss: 1.9599
Epoch 9/50
49/49                0s 8ms/step -
accuracy: 0.3637 - loss: 1.9534 - val_accuracy: 0.4157 - val_loss: 1.8916
Epoch 10/50
49/49                0s 8ms/step -
accuracy: 0.3925 - loss: 1.8633 - val_accuracy: 0.4138 - val_loss: 1.8557
Epoch 11/50
49/49                0s 8ms/step -
accuracy: 0.4115 - loss: 1.8127 - val_accuracy: 0.4486 - val_loss: 1.7739
Epoch 12/50
49/49                0s 8ms/step -
accuracy: 0.4364 - loss: 1.7303 - val_accuracy: 0.4535 - val_loss: 1.7131
Epoch 13/50
49/49                0s 8ms/step -
accuracy: 0.4434 - loss: 1.6976 - val_accuracy: 0.4864 - val_loss: 1.6782
Epoch 14/50
49/49                0s 8ms/step -
```

```
accuracy: 0.4609 - loss: 1.6320 - val_accuracy: 0.5000 - val_loss: 1.6316
Epoch 15/50
49/49           0s 8ms/step -
accuracy: 0.4620 - loss: 1.6430 - val_accuracy: 0.5000 - val_loss: 1.6297
Epoch 16/50
49/49           0s 8ms/step -
accuracy: 0.4979 - loss: 1.5494 - val_accuracy: 0.5155 - val_loss: 1.5894
Epoch 17/50
49/49           0s 8ms/step -
accuracy: 0.4751 - loss: 1.5669 - val_accuracy: 0.5329 - val_loss: 1.5680
Epoch 18/50
49/49           0s 9ms/step -
accuracy: 0.4932 - loss: 1.5607 - val_accuracy: 0.5223 - val_loss: 1.5451
Epoch 19/50
49/49           0s 8ms/step -
accuracy: 0.5145 - loss: 1.4812 - val_accuracy: 0.5359 - val_loss: 1.5241
Epoch 20/50
49/49           0s 8ms/step -
accuracy: 0.5183 - loss: 1.4346 - val_accuracy: 0.5310 - val_loss: 1.5146
Epoch 21/50
49/49           0s 8ms/step -
accuracy: 0.5120 - loss: 1.4510 - val_accuracy: 0.5329 - val_loss: 1.4871
Epoch 22/50
49/49           0s 8ms/step -
accuracy: 0.5142 - loss: 1.4614 - val_accuracy: 0.5475 - val_loss: 1.4796
Epoch 23/50
49/49           0s 8ms/step -
accuracy: 0.5235 - loss: 1.4149 - val_accuracy: 0.5552 - val_loss: 1.4837
Epoch 24/50
49/49           0s 8ms/step -
accuracy: 0.5607 - loss: 1.3466 - val_accuracy: 0.5649 - val_loss: 1.4420
Epoch 25/50
49/49           0s 8ms/step -
accuracy: 0.5393 - loss: 1.3676 - val_accuracy: 0.5484 - val_loss: 1.4446
Epoch 26/50
49/49           0s 8ms/step -
accuracy: 0.5399 - loss: 1.3613 - val_accuracy: 0.5552 - val_loss: 1.4496
Epoch 27/50
49/49           0s 8ms/step -
accuracy: 0.5406 - loss: 1.3704 - val_accuracy: 0.5572 - val_loss: 1.4462
Epoch 28/50
49/49           0s 8ms/step -
accuracy: 0.5598 - loss: 1.3179 - val_accuracy: 0.5756 - val_loss: 1.4098
Epoch 29/50
49/49           0s 8ms/step -
accuracy: 0.5686 - loss: 1.2726 - val_accuracy: 0.5533 - val_loss: 1.4287
Epoch 30/50
49/49           0s 8ms/step -
```

```
accuracy: 0.5550 - loss: 1.3386 - val_accuracy: 0.5640 - val_loss: 1.3997
Epoch 31/50
49/49            0s 8ms/step -
accuracy: 0.5884 - loss: 1.2403 - val_accuracy: 0.5698 - val_loss: 1.4013
Epoch 32/50
49/49            0s 8ms/step -
accuracy: 0.5824 - loss: 1.2521 - val_accuracy: 0.5824 - val_loss: 1.3944
Epoch 33/50
49/49            0s 8ms/step -
accuracy: 0.5900 - loss: 1.2386 - val_accuracy: 0.5862 - val_loss: 1.3800
Epoch 34/50
49/49            0s 8ms/step -
accuracy: 0.5832 - loss: 1.2418 - val_accuracy: 0.5824 - val_loss: 1.3668
Epoch 35/50
49/49            0s 8ms/step -
accuracy: 0.5867 - loss: 1.2517 - val_accuracy: 0.5862 - val_loss: 1.3983
Epoch 36/50
49/49            0s 8ms/step -
accuracy: 0.6056 - loss: 1.1933 - val_accuracy: 0.5891 - val_loss: 1.3747
Epoch 37/50
49/49            0s 8ms/step -
accuracy: 0.5939 - loss: 1.2128 - val_accuracy: 0.5814 - val_loss: 1.3866
Epoch 38/50
49/49            0s 8ms/step -
accuracy: 0.5861 - loss: 1.2218 - val_accuracy: 0.5930 - val_loss: 1.3757
Epoch 39/50
49/49            0s 9ms/step -
accuracy: 0.6008 - loss: 1.2018 - val_accuracy: 0.5988 - val_loss: 1.3631
Epoch 40/50
49/49            0s 8ms/step -
accuracy: 0.6116 - loss: 1.1761 - val_accuracy: 0.5882 - val_loss: 1.3807
Epoch 41/50
49/49            0s 8ms/step -
accuracy: 0.6083 - loss: 1.1637 - val_accuracy: 0.5882 - val_loss: 1.3638
Epoch 42/50
49/49            0s 8ms/step -
accuracy: 0.6149 - loss: 1.1605 - val_accuracy: 0.6047 - val_loss: 1.3604
Epoch 43/50
49/49            0s 8ms/step -
accuracy: 0.6054 - loss: 1.1343 - val_accuracy: 0.5969 - val_loss: 1.3539
Epoch 44/50
49/49            0s 8ms/step -
accuracy: 0.6008 - loss: 1.1551 - val_accuracy: 0.6095 - val_loss: 1.3387
Epoch 45/50
49/49            0s 8ms/step -
accuracy: 0.6091 - loss: 1.1781 - val_accuracy: 0.5979 - val_loss: 1.3394
Epoch 46/50
49/49            0s 8ms/step -
```

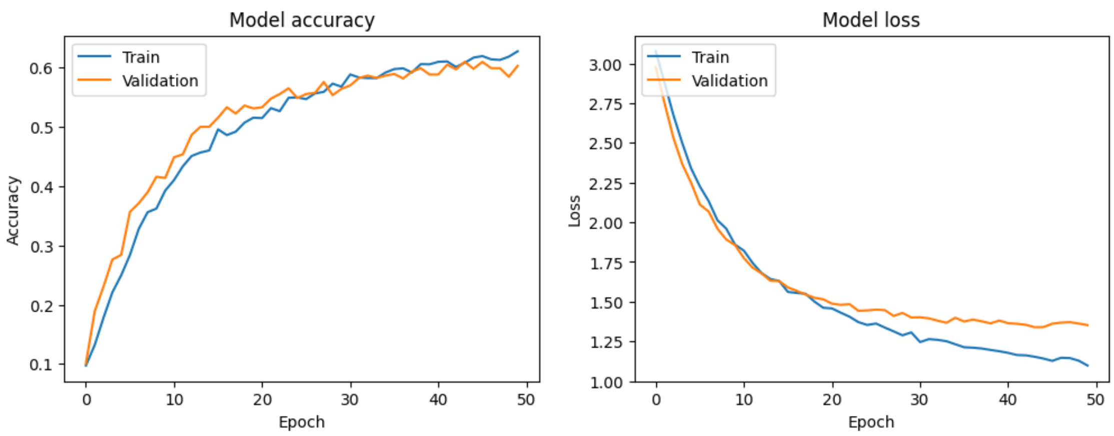
```
accuracy: 0.6098 - loss: 1.1331 - val_accuracy: 0.6095 - val_loss: 1.3609
Epoch 47/50
49/49            0s 8ms/step -
accuracy: 0.5954 - loss: 1.1665 - val_accuracy: 0.5988 - val_loss: 1.3677
Epoch 48/50
49/49            0s 8ms/step -
accuracy: 0.6131 - loss: 1.1347 - val_accuracy: 0.5988 - val_loss: 1.3708
Epoch 49/50
49/49            0s 8ms/step -
accuracy: 0.6061 - loss: 1.1569 - val_accuracy: 0.5843 - val_loss: 1.3617
Epoch 50/50
49/49            0s 8ms/step -
accuracy: 0.6306 - loss: 1.0860 - val_accuracy: 0.6027 - val_loss: 1.3520
```

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

```
17/17            0s 3ms/step -
accuracy: 0.6159 - loss: 1.2009
Test Accuracy: 0.6145
Test Loss: 1.2285
```

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

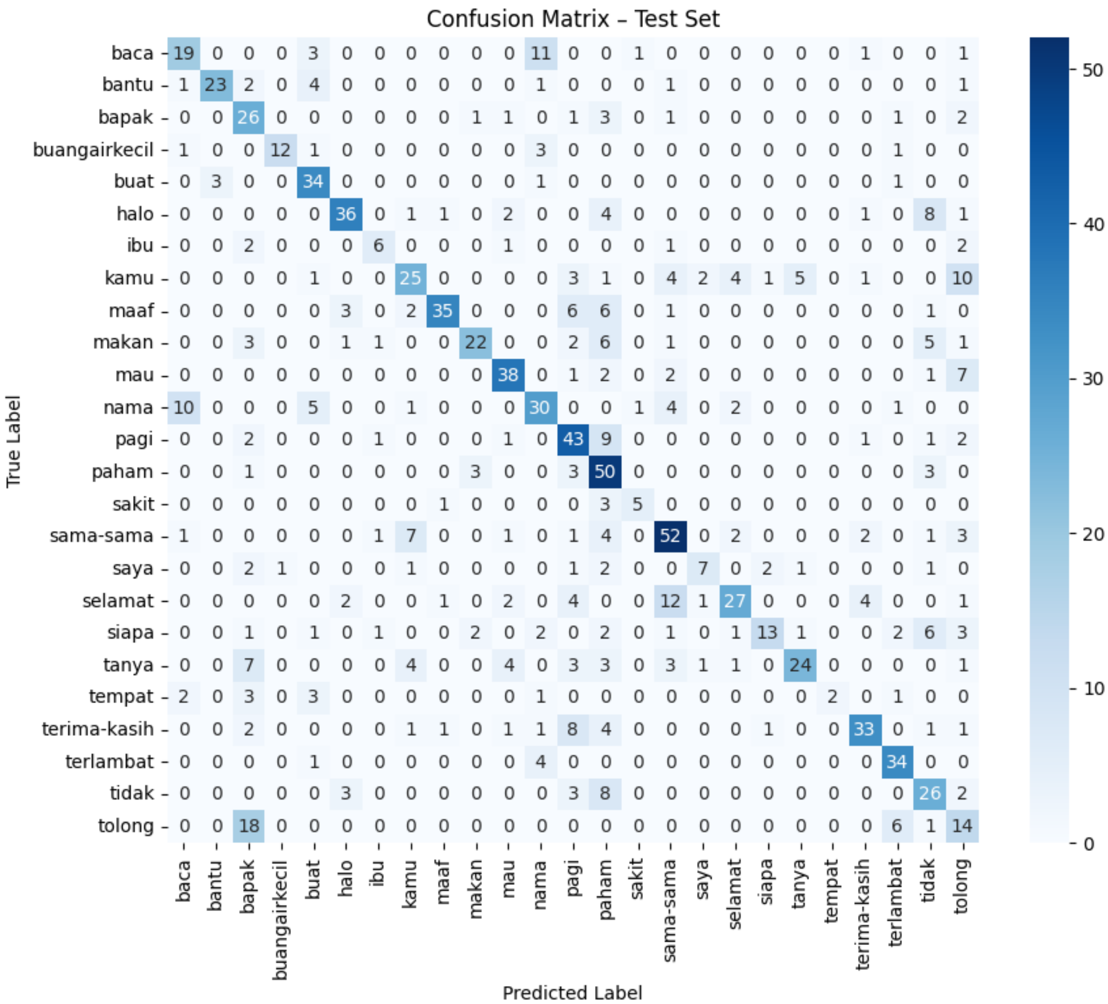
[16]:
```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
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      0.559     0.528     0.543        36
       bantu      0.885     0.697     0.780        33
```

bapak	0.377	0.722	0.495	36
buangairkecil	0.923	0.667	0.774	18
buat	0.642	0.872	0.739	39
halo	0.800	0.667	0.727	54
ibu	0.600	0.500	0.545	12
kamu	0.595	0.439	0.505	57
maaf	0.897	0.648	0.753	54
makan	0.786	0.524	0.629	42
mau	0.745	0.745	0.745	51
nama	0.556	0.556	0.556	54
pagi	0.544	0.717	0.619	60
paham	0.467	0.833	0.599	60
sakit	0.714	0.556	0.625	9
sama-sama	0.627	0.693	0.658	75
saya	0.636	0.389	0.483	18
selamat	0.730	0.500	0.593	54
siapa	0.765	0.361	0.491	36
tanya	0.774	0.471	0.585	51
tempat	1.000	0.167	0.286	12
terima-kasih	0.767	0.611	0.680	54
terlambat	0.723	0.872	0.791	39
tidak	0.473	0.619	0.536	42
tolong	0.269	0.359	0.308	39
accuracy			0.614	1035
macro avg	0.674	0.588	0.602	1035
weighted avg	0.659	0.614	0.616	1035


Confusion Matrix – Test Set
True Label
Predicted Label
baca - 19 0 0 0 3 0 0 0 0 0 0 11 0 0 1 0 0 0 0 0 0 1 0 0 1
bantu - 1 23 2 0 4 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1
bapak - 0 0 26 0 0 0 0 0 0 1 1 0 1 3 0 1 0 0 0 0 0 0 1 0 2
buangairkecil - 1 0 0 12 1 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 1 0 0
buat - 0 3 0 0 34 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
halo - 0 0 0 0 0 36 0 1 1 0 2 0 0 4 0 0 0 0 0 0 0 1 0 8 1
ibu - 0 0 2 0 0 0 6 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 2
kamu - 0 0 0 0 1 0 0 25 0 0 0 0 3 1 0 4 2 4 1 5 0 1 0 0 10
maaf - 0 0 0 0 0 3 0 2 35 0 0 0 6 6 0 1 0 0 0 0 0 0 0 1 0
makan - 0 0 3 0 0 1 1 0 0 22 0 0 2 6 0 1 0 0 0 0 0 0 0 5 1
mau - 0 0 0 0 0 0 0 0 0 0 38 0 1 2 0 2 0 0 0 0 0 0 0 1 7
nama - 10 0 0 0 5 0 0 1 0 0 0 30 0 0 1 4 0 2 0 0 0 0 1 0 0
pagi - 0 0 2 0 0 0 1 0 0 0 1 0 43 9 0 0 0 0 0 0 0 1 0 1 2
paham - 0 0 1 0 0 0 0 0 0 3 0 0 3 50 0 0 0 0 0 0 0 0 0 3 0
sakit - 0 0 0 0 0 0 0 0 1 0 0 0 0 3 5 0 0 0 0 0 0 0 0 0 0
sama-sama - 1 0 0 0 0 0 1 7 0 0 1 0 1 4 0 52 0 2 0 0 0 2 0 1 3
saya - 0 0 2 1 0 0 0 1 0 0 0 0 1 2 0 0 7 0 2 1 0 0 0 1 0
selamat - 0 0 0 0 0 2 0 0 1 0 2 0 4 0 0 12 1 27 0 0 0 4 0 0 1
siapa - 0 0 1 0 1 0 1 0 0 2 0 2 0 2 0 1 0 1 13 1 0 0 2 6 3
tanya - 0 0 7 0 0 0 0 4 0 0 4 0 3 3 0 3 1 1 0 24 0 0 0 0 1
tempat - 2 0 3 0 3 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 2 0 1 0 0
terima-kasih - 0 0 2 0 0 0 0 1 1 0 1 1 8 4 0 0 0 0 1 0 0 33 0 1 1
terlambat - 0 0 0 0 1 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 34 0 0
tidak - 0 0 0 0 0 3 0 0 0 0 0 0 3 8 0 0 0 0 0 0 0 0 0 26 2
tolong - 0 0 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 1 14
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
50 40 30 20 10 0