การจำแนกการออกเสียงภาษาไทยของชาวต่างชาติ

Thai Pronunciation of foreigners Classification

Introduction

01

Data Collected

YouTube
TH
JP
CN
KR
EN

02

Data Cleansing

spleeter

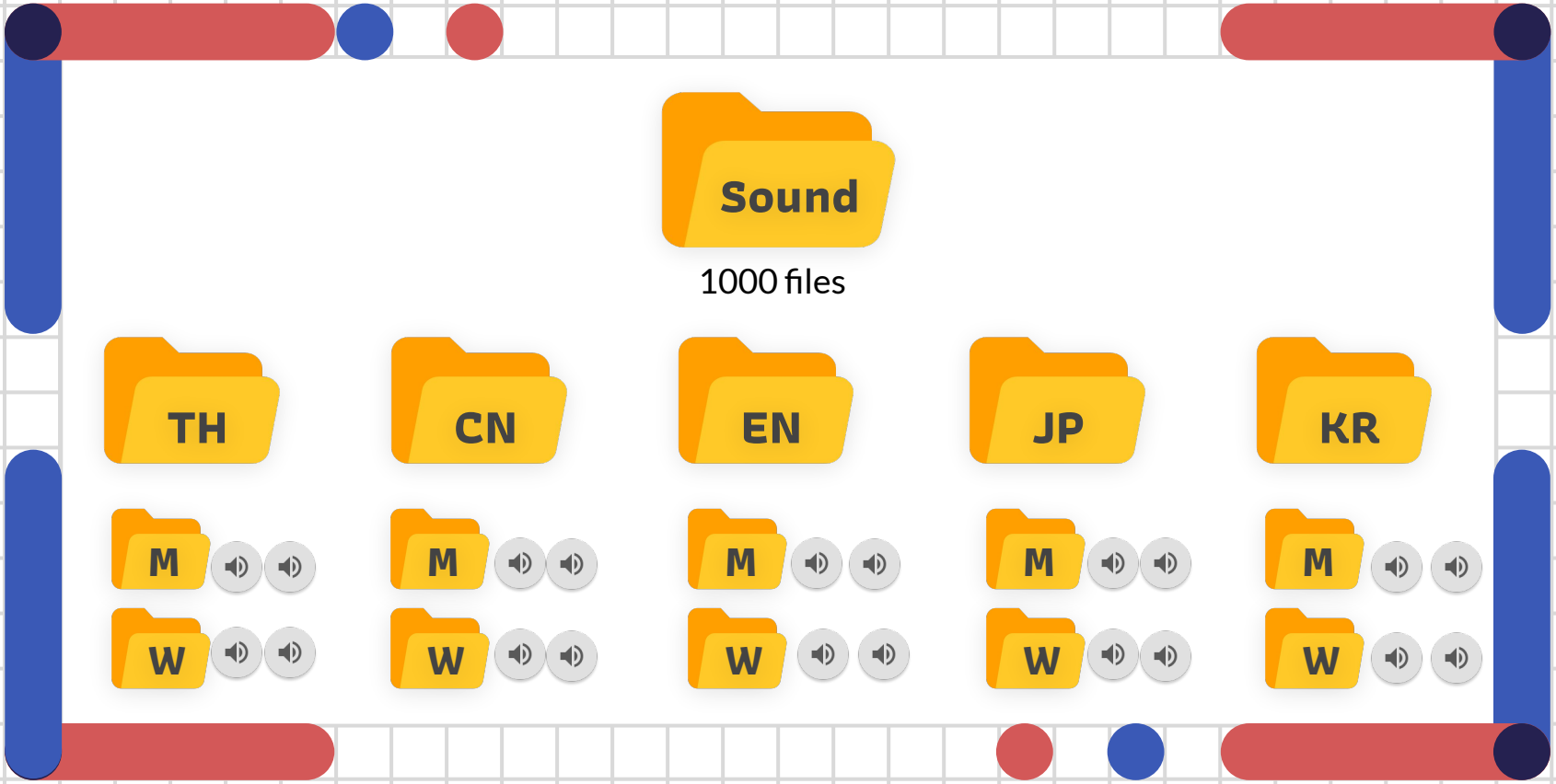
Windows 10

circleci passing | python 3.6 | 3.7 | pypi package 1.5.2 | conda-forge v1.5.2

JOSS Under Review

- Vocals (singing voice) / accompaniment separation (2 stems)
- Vocals / drums / bass / other separation (4 stems)
- Vocals / drums / bass / piano / other separation (5 stems)

Au


Sound
1000 files
TH
CN
EN
JP
KR
M
W
M
W
M
W
M
W
M
W

03

Data Preparation

Sound

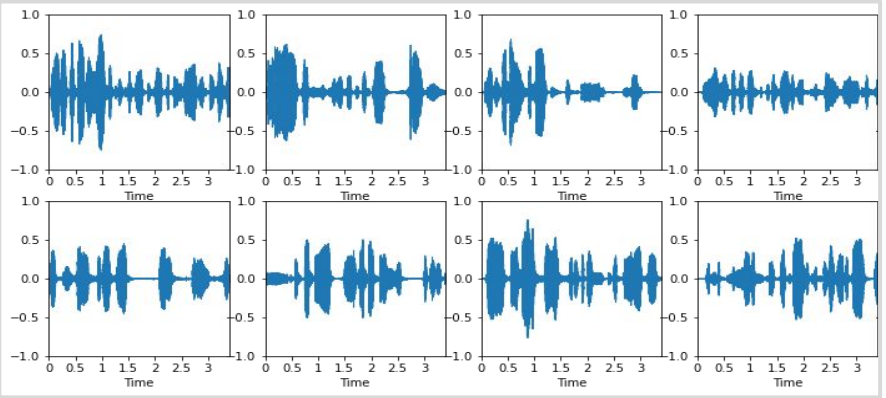
```
dataset = data_path + "sound/"
# os.listdir(dataset)
```

```
import librosa
import librosa.display
```

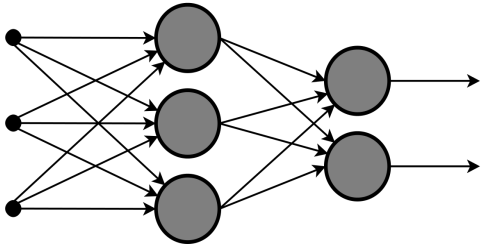
```
th = dataset + "TH_* (*).wav"
cn = dataset + "CN_* (*).wav"
en = dataset + "EN_* (*).wav"
jp = dataset + "JP_* (*).wav"
kr = dataset + "KR_* (*).wav"
```

librosa

```
th_signals = [librosa.load(p)[0] for p in Path().glob(th)]
cn_signals = [librosa.load(p)[0] for p in Path().glob(cn)]
en_signals = [librosa.load(p)[0] for p in Path().glob(en)]
jp_signals = [librosa.load(p)[0] for p in Path().glob(jp)]
kr_signals = [librosa.load(p)[0] for p in Path().glob(kr)]
```

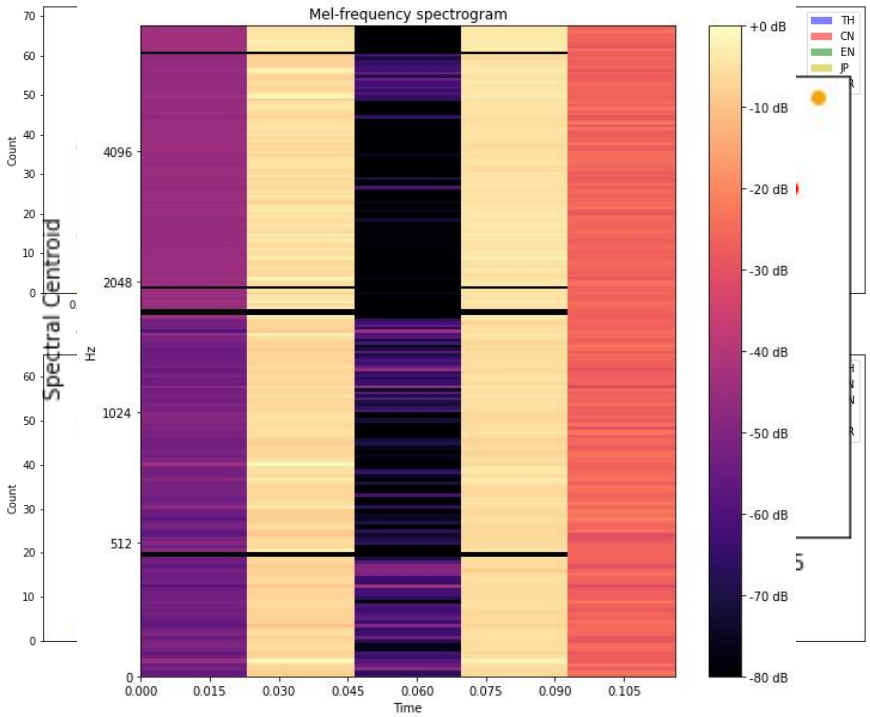


Feature Extraction



```python
def extract_features(signal):
    return [
        librosa.feature.zero_crossing_rate(signal)[0, 0],
        librosa.feature.spectral_centroid(signal)[0, 0],
        librosa.feature.mfcc(signal)[0, 0],
        librosa.feature.rms(signal)[0, 0],
        librosa.feature.spectral_contrast(signal)[0, 0],
    ]
```

```python
th_features = np.array([extract_features(x) for x in th_signals])
cn_features = np.array([extract_features(x) for x in cn_signals])
en_features = np.array([extract_features(x) for x in en_signals])
jp_features = np.array([extract_features(x) for x in jp_signals])
kr_features = np.array([extract_features(x) for x in kr_signals])
```



Split Train Test data

```
train_th, test_th, yth_train, yth_test = train_test_split(th_features, yth, test_size=0.2, shuffle=True,random_state=123)
train_cn, test_cn, ycn_train, ycn_test = train_test_split(cn_features, ycn, test_size=0.2, shuffle=True,random_state=123)
train_en, test_en, yen_train, yen_test = train_test_split(en_features, yen, test_size=0.2, shuffle=True,random_state=123)
train_jp, test_jp, yjp_train, yjp_test = train_test_split(jp_features, yjp, test_size=0.2, shuffle=True,random_state=123)
train_kr, test_kr, ykr_train, ykr_test = train_test_split(kr_features, ykr, test_size=0.2, shuffle=True,random_state=123)
```

```
train_data = np.vstack((train_th,train_cn,train_en,train_jp,train_kr))
y_train = yth_train+ycn_train+yen_train+yjp_train+ykr_train
y_train = to_categorical(y_train)
print("Train dataset :",train_data.shape)
print("Y-Train dataset :",y_train.shape)
```

```
Train dataset : (800, 5)
Y-Train dataset : (800, 5)
```

```
test_data = np.vstack((test_th,test_cn,test_en,test_jp,test_kr))
y_test = yth_test+ycn_test+yen_test+yjp_test+ykr_test
y_test = to_categorical(y_test)
print("Test dataset :",test_data.shape)
print("Y-Test dataset :",y_test.shape)
```

```
Test dataset : (200, 5)
Y-Test dataset : (200, 5)
```

Split Train Valid data

```
x_train, x_val, y_train, y_val = train_test_split(train_data, y_train, test_size=0.2, shuffle=True,random_state=123)
```

```
x_train.shape, x_val.shape, y_train.shape, y_val.shape
```

```
((640, 5), (160, 5), (640, 5), (160, 5))
```

Train	Validation	Test
(640,5)	(160,5)	(200,5)

04

Models

Baseline Models

```python
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

```python
def create_model():
    model = Sequential()
    model.add(Dense(8, input_shape=(5,)))
    model.add(LeakyReLU(alpha=0.1))
    model.add(Dropout(0.1))
    model.add(Dense(5, activation='softmax'))
    return model
```

```python
model = create_model()
model.summary()
adam_optim = Adam(learning_rate = 0.0001)
model.compile(optimizer=adam_optim, loss='categorical_crossentropy', metrics=['accuracy'])
```

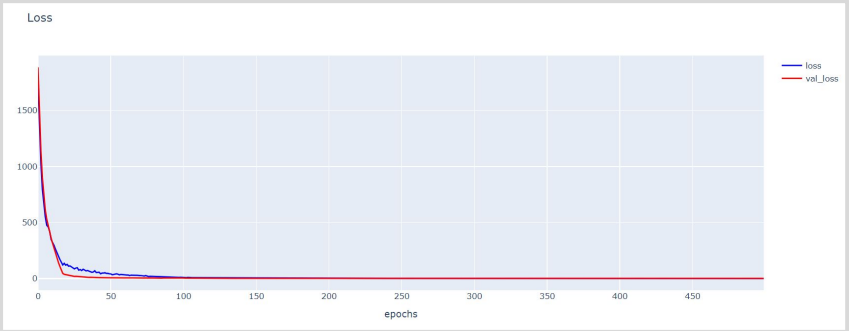
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 48

 leaky_re_lu (LeakyReLU)     (None, 8)                 0

 dropout (Dropout)           (None, 8)                 0

 dense_1 (Dense)             (None, 5)                 45

=================================================================
Total params: 93
Trainable params: 93
Non-trainable params: 0
_________________________________________________________________
```

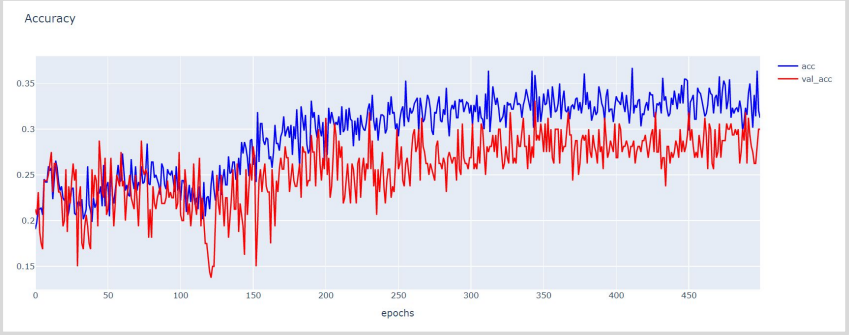
Baseline Models

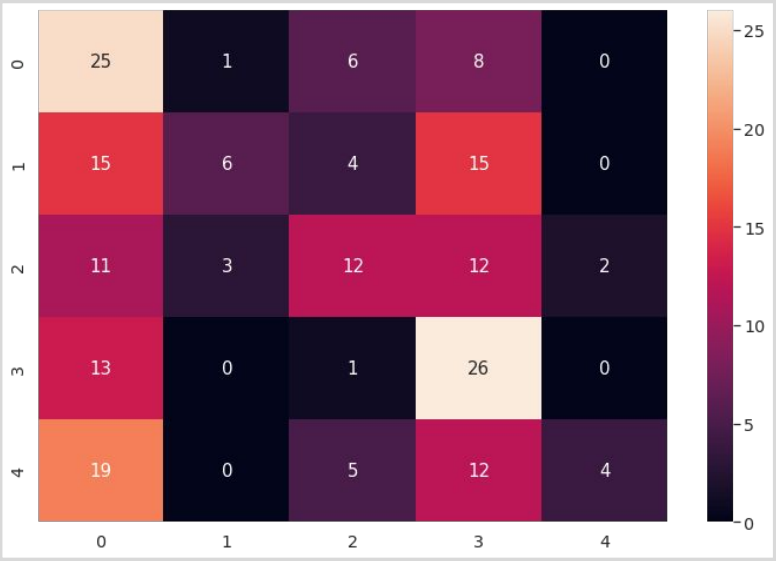
```python
his = model.fit(
    x_train, y_train,
    batch_size=1,
    epochs=500,
    verbose=1,
    validation_data = (x_val, y_val)
)
```

```
Epoch 1/500
640/640 [==============================] - 5s 4ms/step - loss: 1733.1400 - accuracy: 0.1906 - val_loss: 1885.6619 - val_accuracy: 0.2125
Epoch 2/500
640/640 [==============================] - 2s 3ms/step - loss: 1363.1937 - accuracy: 0.1984 - val_loss: 1467.6289 - val_accuracy: 0.2062
Epoch 3/500
640/640 [==============================] - 2s 3ms/step - loss: 1020.6646 - accuracy: 0.2141 - val_loss: 1134.4502 - val_accuracy: 0.2313
Epoch 4/500
640/640 [==============================] - 2s 3ms/step - loss: 792.6118 - accuracy: 0.2125 - val_loss: 900.8021 - val_accuracy: 0.1875
Epoch 5/500
640/640 [==============================] - 2s 3ms/step - loss: 674.7211 - accuracy: 0.2141 - val_loss: 738.8348 - val_accuracy: 0.1750
Epoch 6/500
640/640 [==============================] - 2s 3ms/step - loss: 549.5569 - accuracy: 0.2062 - val_loss: 611.3173 - val_accuracy: 0.1688
Epoch 7/500
640/640 [==============================] - 2s 3ms/step - loss: 475.3826 - accuracy: 0.2453 - val_loss: 530.2086 - val_accuracy: 0.2438
Epoch 8/500
640/640 [==============================] - 3s 4ms/step - loss: 460.6410 - accuracy: 0.2422 - val_loss: 470.4824 - val_accuracy: 0.2438
Epoch 9/500
640/640 [==============================] - 2s 3ms/step - loss: 420.1028 - accuracy: 0.2422 - val_loss: 417.8871 - val_accuracy: 0.2438
Epoch 10/500
640/640 [==============================] - 2s 3ms/step - loss: 349.1039 - accuracy: 0.2594 - val_loss: 363.0685 - val_accuracy: 0.2500
Epoch 11/500
640/640 [==============================] - 2s 4ms/step - loss: 321.4443 - accuracy: 0.2547 - val_loss: 315.3325 - val_accuracy: 0.2625
```

Baseline Models







Baseline Models

```python
report = classification_report(y_true, predicted_classes, target_names=CATEGORIES, digits=4)

print(report)
```

```
              precision    recall  f1-score   support

          TH     0.3012    0.6250    0.4065        40
          CN     0.6000    0.1500    0.2400        40
          EN     0.4286    0.3000    0.3529        40
          JP     0.3562    0.6500    0.4602        40
          KR     0.6667    0.1000    0.1739        40

    accuracy                         0.3650       200
   macro avg     0.4705    0.3650    0.3267       200
weighted avg     0.4705    0.3650    0.3267       200
```

CNN Models

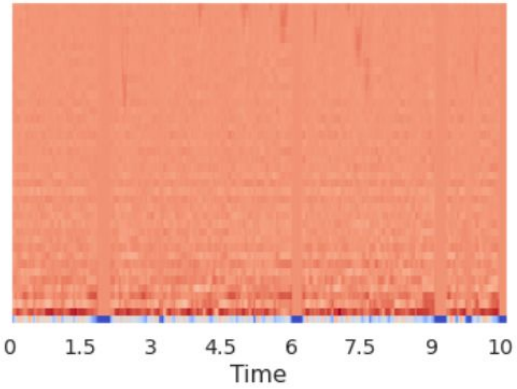
```
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.optimizers import Adam
```

```
file_name = 'drive/My Drive/AUCC/DP/dataset/sound/TH_M (1).wav'
audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')

print(audio.shape, sample_rate)

mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
print(mfccs.shape)
```

```
(233856,) 22050
(40, 457)
```

```
librosa.display.specshow(mfccs, sr=sample_rate, x_axis='time')
```

```
<matplotlib.collections.QuadMesh at 0x7fc64d3ab9d0>
```



CNN Models

```python
max_pad_len = 1000

def extract_features(file_name):
    audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
    pad_width = max_pad_len - mfccs.shape[1]
    mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)), mode='constant')
    return mfccs
```

```python
TH_features = []
for file_name in Path().glob(th):
    data = extract_features(file_name)
    TH_features.append(data)
print(len(TH_features))
TH_features[0].shape
```

```
200
(40, 1000)
```

```python
CN_features = []
for file_name in Path().glob(cn):
    data = extract_features(file_name)
    CN_features.append(data)
print(len(CN_features))
CN_features[0].shape
```

```
200
(40, 1000)
```

```python
EN_features = []
for file_name in Path().glob(en):
    data = extract_features(file_name)
    EN_features.append(data)
print(len(EN_features))
EN_features[0].shape
```

```
200
(40, 1000)
```

```python
JP_features = []
for file_name in Path().glob(jp):
    data = extract_features(file_name)
    JP_features.append(data)
print(len(JP_features))
JP_features[0].shape
```

```
200
(40, 1000)
```

```python
KR_features = []
for file_name in Path().glob(kr):
    data = extract_features(file_name)
    KR_features.append(data)
print(len(KR_features))
KR_features[0].shape
```

```
200
(40, 1000)
```

CNN Models

```python
num_rows = 40
num_columns = 1000
num_channels = 1
num_batch_size = 64
epochs = 500
```

```python
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows, num_columns, num_channels), activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(GlobalAveragePooling2D())

model.add(Dense(5, activation='softmax'))
```

```python
model.summary()
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```

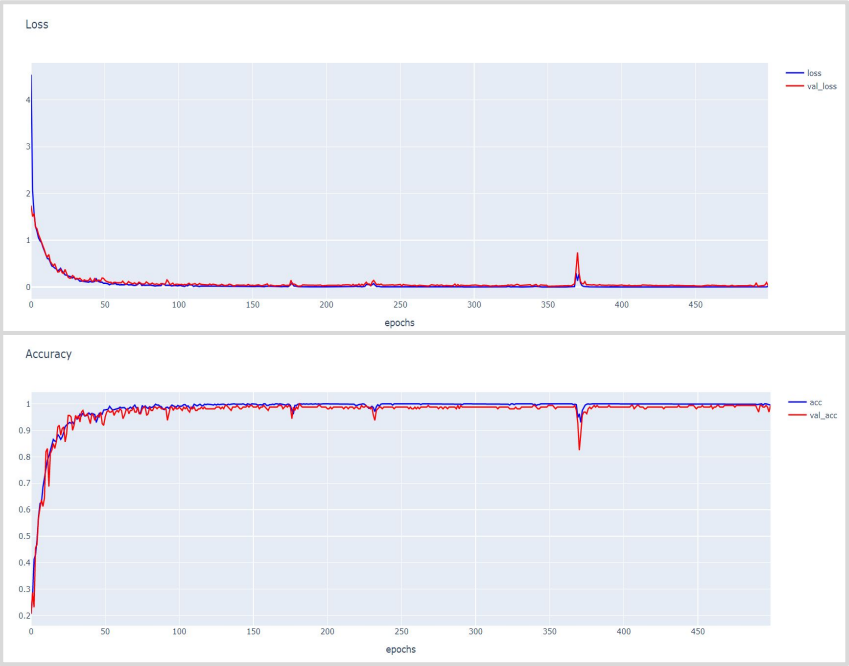
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 39, 999, 16)       80

 max_pooling2d (MaxPooling2D  (None, 19, 499, 16)      0
 )

 dropout (Dropout)           (None, 19, 499, 16)       0

 conv2d_1 (Conv2D)           (None, 18, 498, 32)       2080

 max_pooling2d_1 (MaxPooling  (None, 9, 249, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 9, 249, 32)        0

 conv2d_2 (Conv2D)           (None, 8, 248, 64)        8256

 max_pooling2d_2 (MaxPooling  (None, 4, 124, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 4, 124, 64)        0

 conv2d_3 (Conv2D)           (None, 3, 123, 128)       32896

 max_pooling2d_3 (MaxPooling  (None, 1, 61, 128)       0
 2D)

 dropout_3 (Dropout)         (None, 1, 61, 128)        0

 global_average_pooling2d (G  (None, 128)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 5)                 645

=================================================================
Total params: 43,957
Trainable params: 43,957
Non-trainable params: 0
_________________________________________________________________
```

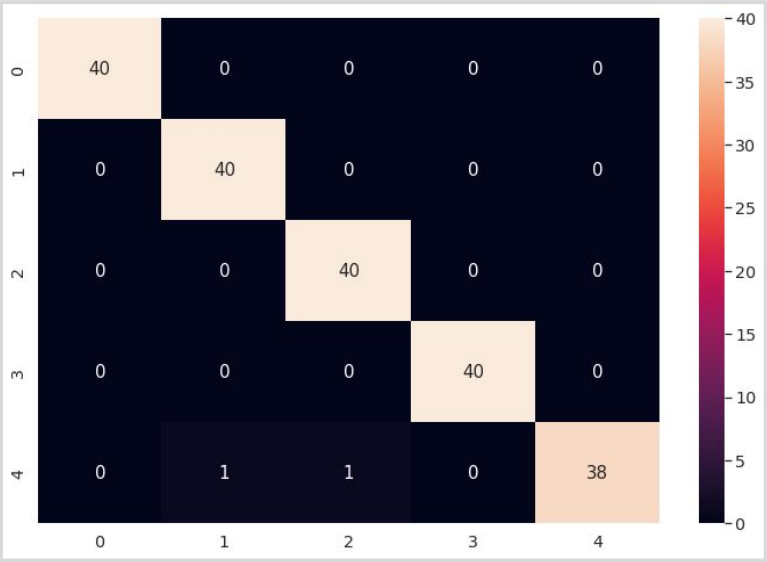
CNN Models

```
his = model.fit(x_train, y_train,
                batch_size=num_batch_size,
                epochs=epochs,
                validation_data=(x_val, y_val),
                verbose=1)
```

```
Epoch 1/500
10/10 [==============================] - 10s 133ms/step - loss: 4.5420 - accuracy: 0.2078 - val_loss: 1.7414 - val_accuracy: 0.2062
Epoch 2/500
10/10 [==============================] - 1s 89ms/step - loss: 2.0627 - accuracy: 0.2766 - val_loss: 1.5087 - val_accuracy: 0.2875
Epoch 3/500
10/10 [==============================] - 1s 90ms/step - loss: 1.5829 - accuracy: 0.4125 - val_loss: 1.5723 - val_accuracy: 0.2313
Epoch 4/500
10/10 [==============================] - 1s 79ms/step - loss: 1.2940 - accuracy: 0.4297 - val_loss: 1.2806 - val_accuracy: 0.4563
Epoch 5/500
10/10 [==============================] - 1s 80ms/step - loss: 1.1900 - accuracy: 0.4859 - val_loss: 1.2474 - val_accuracy: 0.4688
Epoch 6/500
10/10 [==============================] - 1s 79ms/step - loss: 1.0540 - accuracy: 0.5734 - val_loss: 1.1262 - val_accuracy: 0.5688
Epoch 7/500
10/10 [==============================] - 1s 80ms/step - loss: 0.9888 - accuracy: 0.6234 - val_loss: 1.0489 - val_accuracy: 0.6062
Epoch 8/500
10/10 [==============================] - 1s 90ms/step - loss: 0.9526 - accuracy: 0.6281 - val_loss: 0.9537 - val_accuracy: 0.6375
Epoch 9/500
10/10 [==============================] - 1s 80ms/step - loss: 0.8550 - accuracy: 0.6859 - val_loss: 0.8805 - val_accuracy: 0.6125
Epoch 10/500
10/10 [==============================] - 1s 81ms/step - loss: 0.7669 - accuracy: 0.7188 - val_loss: 0.7960 - val_accuracy: 0.6438
Epoch 11/500
10/10 [==============================] - 1s 82ms/step - loss: 0.7065 - accuracy: 0.7453 - val_loss: 0.6872 - val_accuracy: 0.8188
Epoch 12/500
10/10 [==============================] - 1s 80ms/step - loss: 0.6102 - accuracy: 0.7875 - val_loss: 0.6426 - val_accuracy: 0.8313
```

CNN Models


Loss
loss
val_loss
epochs
Accuracy
acc
val_acc
epochs


40 0 0 0 0
0 40 0 0 0
0 0 40 0 0
0 0 0 40 0
0 1 1 0 38
0 1 2 3 4

CNN Models

```
report = classification_report(y_true, predicted_classes, target_names=CATEGORIES, digits=4)

print(report)
```

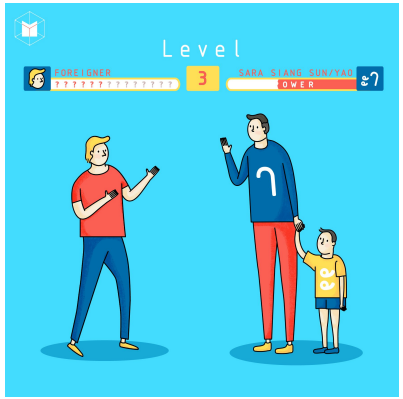
```
              precision    recall  f1-score   support

          TH     1.0000    1.0000    1.0000        40
          CN     0.9756    1.0000    0.9877        40
          EN     0.9756    1.0000    0.9877        40
          JP     1.0000    1.0000    1.0000        40
          KR     1.0000    0.9500    0.9744        40

    accuracy                         0.9900       200
   macro avg     0.9902    0.9900    0.9899       200
weighted avg     0.9902    0.9900    0.9899       200
```

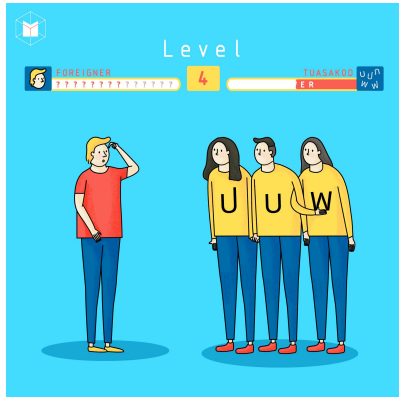
Thanks

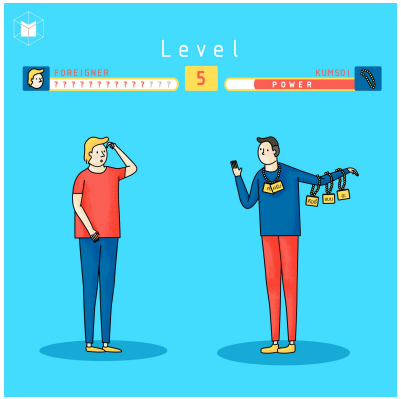
620710405 นางสาวณัฐริดา ลาภธนชัย

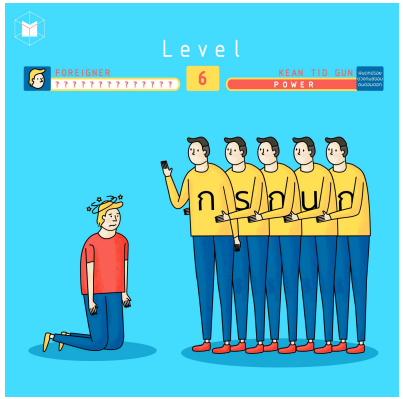
620710745 นางสาวอาทิตยา ชมทอง

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics & images by **Freepik**









https://thematter.co/social/thai-language-is-not-that-easy/29010