Thai Dessert Classification

จัดทำโดย

นายกันตณัฐ	นาคสมบูรณ์	6013110
นายกิตติภูมิ	สุทธินันทกร	6013111
นายรัตนพงษ์	ชุนไธสง	6013122
นายจีรเดช	สวัสดิรักษา	6013287
นายธงชัย	แย้มสุข	6013291

เสนอ

ผศ.ดร. นริศ หนูหอม

รายงานนี้เป็นส่วนหนึ่งของรายวิชา Image Processing (EGCO 486)

คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยมหิดล

คำนำ

รายงานเล่มนี้ได้นำทฤษฎีเบื้องต้นของการทำ Thai Dessert Classification หรือการจำแนกประเภทขนมไทย โดยทำการ Deep learning เป็นการทำ Machine learning ที่ใช้การ Convolution Neural Network (CNN) InceptionV3 Xception และMobileNetV2 รวมถึงอธิบายขั้นตอนการทำงานและผลลัพธ์หลังจากการทำ Deep learning และ Neural Network และทำการเปรียบเทียบผลลัพธ์ จากการทำงานของทั้ง 3 โมเดล

คณะผู้จัดทำ

สารบัญ

Dataset Description

ชุดข้อมูลขนมไทยประกอบไปด้วยขนมไทย 10 ชนิด

- ทองหยอด จำนวน 200 รูป
- ทองหยิบ จำนวน 200 รูป
- สังขยาฟักทอง จำนวน 225 รูป
- ซาหริ่ม จำนวน 200 รูป
- ฝอยทอง จำนวน 217 รูป
- ขนมครก จำนวน 207 รูป
- ไข่เต่า จำนวน 260 รูป
- ดอกจอก จำนวน 200 รูป
- ขนมชั้น จำนวน 217 รูป
- บัวลอย จำนวน 263 รูป

รวมทั้งหมด 2189 รูป

ตัวอย่างรูปในแต่ละคลาส

- ทองหยอด

- ทองหยิบ

- สังขยาฟักทอง

- ซาหริ่ม

- ฝอยทอง

- ขนมครก

- ไข่เต่า

- ดอกจอก

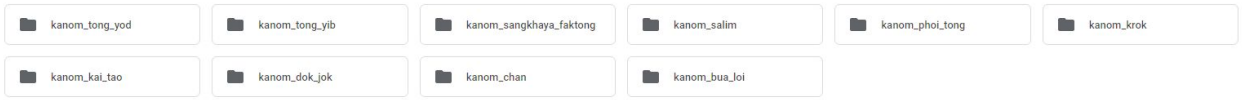
- ขนมชั้น

- บัวลอย

Data Preparation

1. ดาวน์โหลดรูปภาพจากอินเทอร์เน็ต โดยใช้ Chrome Extension Fatkun Batch Download Image โดยหลังจากการคัดภาพแล้วได้รวม 2189 ภาพ

2. เก็บรูปในแต่ละคลาสไว้ในโฟลเดอร์ Thai Dessert Dataset ดังภาพ



3. ทำกระบวนการ Augmentation หมุนรูป 90˚, 180˚, 270˚ แล้วเก็บไว้ในโฟลเดอร์ Thai Dessert Dataset Augmented

4. แยกรูปออกจากแต่ละคลาส 50 รูปเพื่อเก็บไว้เป็น Test เก็บไว้ในโฟลเดอร์ Test และส่วนที่เหลือเก็บไว้ในโฟลเดอร์ Train

5. ในโฟลเดอร์ Train แบ่งรูปภาพเป็น Train กับ Validation โดยแบ่งเป็น Train 80% Validation 20% โดยใช้ฟังก์ชัน tf.keras.preprocessing.image_dataset_from_directory() ฟังก์ชันนี้จะโหลดชุดข้อมูลตามตำแหน่งที่เราระบุและแบ่งชื่อคลาสตามชื่อโฟลเดอร์ที่เรากำหนด โดยพารามิเตอร์ดังรูปด้านล่าง พารามิเตอร์ที่สำคัญ

    • validation_split = 0.2 แบ่ง Train/Validate 80/20

    • image_size = (img_height, img_witdh) รูปถูก resize ให้เป็นขนาด 160 x 160 ให้สอดคล้องกับความต้องการของ Pre-Trained Model

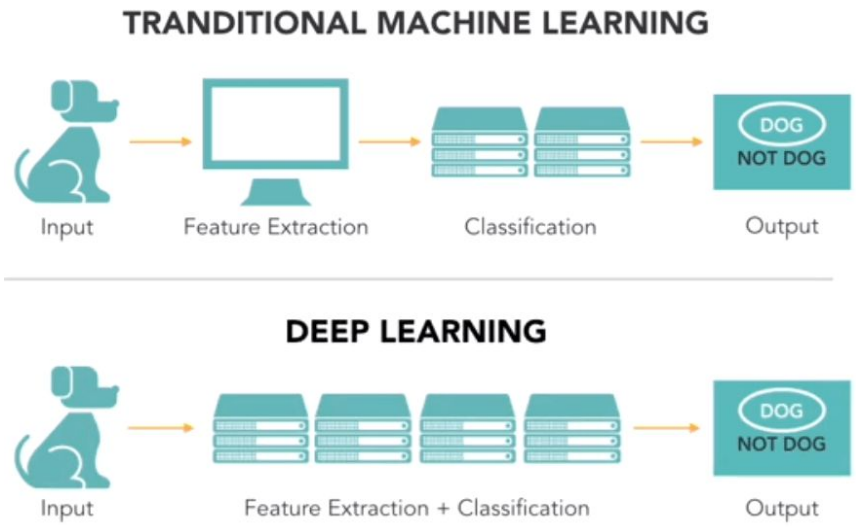
```
BATCH_SIZE = 32
SHUFFLE_BUFFER_SIZE = 1000
img_height = 160
img_width = 160

train_batches = tf.keras.preprocessing.image_dataset_from_directory(
  dataset_path,
  validation_split=0.2,
  subset="training",
  seed=123,
  shuffle = True,
  image_size=(img_height, img_width),
  batch_size=BATCH_SIZE,
  label_mode='categorical')
```

```
validation_batches = tf.keras.preprocessing.image_dataset_from_directory(
  dataset_path,
  validation_split=0.2,
  subset="validation",
  seed=123,
  shuffle = True,
  image_size=(img_height, img_width),
  batch_size=BATCH_SIZE,
  label_mode='categorical')
```

Methods

Convolutional Neural Network (CNN) คือโครงข่ายปราสาทเทียมชนิดหนึ่งที่จำลองการมองเห็นของมนุษย์ที่มองส่วนประกอบย่อย ๆ ของภาพ และนำกลุ่มภาพย่อย ๆ นั้นมาผสานกัน โดยการมองภาพในส่วนประกอบย่อยของมนุษย์นั้นจะมีการแยกคุณลักษณะ (feature) ของภาพส่วนย่อย ๆ นั้น เช่น ลายเส้น และการตัดกันของสี ซึ่งการที่มนุษย์รู้ว่าพื้นที่ตรงนี้เป็นเส้นตรงหรือสีตัดกัน เพราะมนุษย์ดูทั้งจุดที่สนใจและบริเวณรอบ ๆ ประกอบกัน

ซึ่งการประยุกต์ใช้ deep learning ทำให้อัลกอริทึมสามารถทำงานที่ซับซ้อนได้ดีขึ้น อีกทั้งยังลดขั้นตอนการทำ Feature extraction ได้ด้วยตัวเอง



ส่วนประกอบที่สำคัญ

1. Input Layer

มีหน้าที่ในการรับข้อมูลเข้ามาในโครงข่ายประสาทโดย Input Layer จะมีเพียงชั้นเดียวเท่านั้นและมีหน้าส่งข้อมูลไปยังชั้นถัดไป (Hidden Layer)

2. Hidden Layer

มีหน้าที่รับข้อมูลจาก Layer ก่อนหน้า จะสังเกตุว่า Hidden Layer สามารถมีจำนวนมากกว่า 1 Layer ได้ และโดยพื้นฐาน การเพิ่มจำนวนชั้นของ Hidden Layer และจำนวน Neurons อาจเพิ่มความแม่นยำที่มากขึ้น Convolution Layer (ConvLayer) มักถูกนำมาใช้ทำหน้าที่สกัดเอา Feature ที่สำคัญจากรูปภาพ

ConvLayer มีความพิเศษตรงที่ คงความสัมพันธ์ของ Pixel ที่อยู่บริเวณพื้นที่ใกล้เคียงกันเอาไว้ด้วย การทำ Convolution รูปด้วย Filter ที่แตกต่างกัน จะได้ความหมายที่แตกต่างกันไป เช่น หาขอบรูป, หาความเบลอ, หาความคม

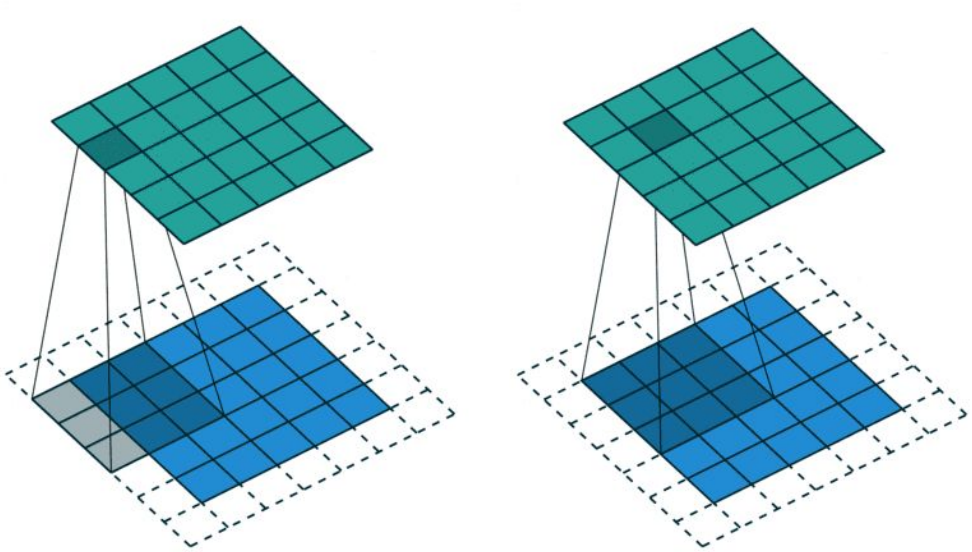
2.1 Filter

สำหรับ Filter ของภาพดิจิตอลนั้นโดยปกติแล้วจะเป็นตารางสองมิติ ที่มีขนาดตามพื้นที่ย่อย ๆ ที่เราอยากพิจารณาซึ่งตัวกรองจะถูกทาบลงใน Pixel แรกของภาพข้อมูลเข้า จากนั้นจะถูกเลื่อนไปทาบลงบน Pixel อื่นในภาพ ทีละ Pixel จนครบทุก Pixel ในภาพและได้สิ่งที่เรียกว่า feature map

2.2 Stride และ Padding

Stride เป็นตัวกำหนดว่าเราจะเลื่อนตัวกรอง (filter) ไปด้วย Step เท่าไร การกำหนดค่าของ Stride ให้มากขึ้นจะทำให้การคำนวณหาคุณลักษณะมีพื้นที่ทับซ้อนกันน้อยลง และการกำหนดค่าของ Stride ที่มากขึ้นจะทำให้เราได้ feature map ที่มีขนาดเล็กลง

Padding คือการเติมขอบให้ภาพเพราะในบางปัญหา Input ที่อยู่ตามขอบภาพอาจมีควาวมสำคัญที่ส่งผลต่อการตัดสินใจบางอย่าง เราจึงจำเป็นต้องเก็บคุณลักษณะตามขอบของรูปภาพไว้ด้วยการเติม Padding

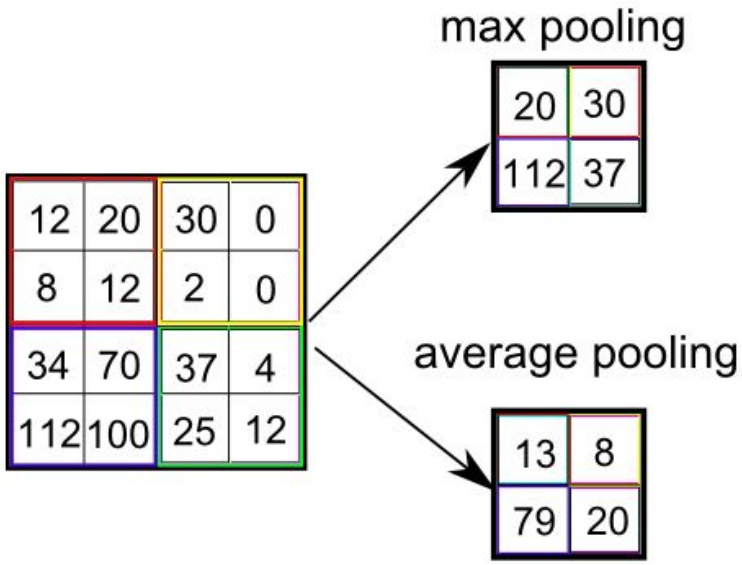


2.3 Pooling

โดยปกติแล้วแม้รูปภาพมีขนาดสเกลที่เล็กลง แต่มนุษย์ก็ยังสามารถบอกได้ว่าสิ่งนั้นคืออะไร แสดงว่าจำแนกภาพต่าง ๆ ออกจากกันสามารถทำได้แม้มีความละเอียดต่ำโดยอาศัยการดูที่รายละเอียดเล็ก ๆ และการดูแบบคร่าว ๆ บนพื้นที่ใหญ่ แต่เมื่อ filter มีขนาดมีขนาดเท่าเดิมการทำให้มีขนาดเล็กลง ส่งผลให้ filter ครอบคลุมพื้นที่วัตถุเดิมมากขึ้น โดย Pooling คือความสามารถในการย่อรูปแบบหนึ่ง ซึ่งมีสองประเภทหลัก ที่นิยมกันคือ Max pooling และ Average pooling

Average Pooling เป็นการลดขนาดโดยหาค่าเฉลี่ยจาก filter ที่วางอยู่บน feature map โดยเราจะเตรียมตัวกรองในลักษณะเดียวกับการทำ Feature Extraction ของ CNN

Max Pooling เป็นการลดขนาดโดยหาค่าสูงสุดจาก filter ที่วางอยู่บน feature map โดยเราจะเตรียมตัวกรองในลักษณะเดียวกับการทำ Feature Extraction ของ CNN



2.4 Activation function

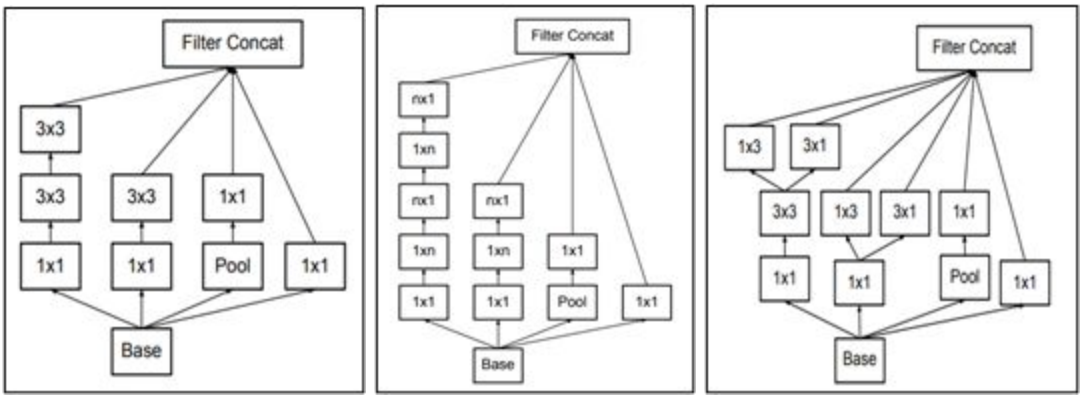
ทำหน้าที่รับ output จาก hidden layer แปลงให้อยู่ในรูปของ nonlinear เพื่อความง่ายในการคำนวณ และสร้างเป็น information ส่งไปให้ layer ถัดไป

Relu function คือฟังก์ชั่นเชิงเส้นที่แสดงผลลัพธ์ในช่วงที่มีค่าตั้งแต่ บวกจนถึงศูนย์

Softmax เป็นฟังก์ชั่นที่คำนวณผลลัพธ์ probability ของแต่ละคลาส ที่มี ค่าระหว่าง [0,1]

Method 1 : InceptionV3

model InceptionV3 เป็น model ที่ถูกพัฒนาโดย Google โดยพัฒนามาจาก InceptionV1,V2 ซึ่ง model นี้แต่เดิมถูกสร้างมาเพื่อให้สามารถ classify ได้ถึง 1000 classes



การทำงานของ inceptionV3 เป็น model ที่มุ่งเน้นด้านความลึกของ network ใช้ Convolutions layer ที่มีขนาดเล็กและการลดจำนวนการเชื่อมต่อของ parameter ลง อีกทั้งมีการเพิ่มตัวช่วยตัดสินใจเข้ามาร่วมด้วย

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

ขั้นตอนการ Train

1. เตรียม InceptionV3 pre-trained model โดยกำหนด input_shape ให้ตรงกับ dataset ที่เตรียมไว้ กำหนด include_top = false เพื่อไม่ให้ทำในส่วนของ Classification แต่จะนำ feature ที่ได้นั้นมาเข้า Neural Network กำหนด weights = ‘imagenet’ คือการนำ weight ที่ทาง Google ได้ pretrain ไว้ classifier_activation คือให้ layer สุดท้ายเป็น Softmax ก่อนแล้วมาใช้ base_model.trainable = False เพื่อไม่ให้ weight ของ pre-trained model เกิดการเปลี่ยนแปลงในการ train ช่วงแรก

```
IMG_SHAPE = (160, 160, 3)

# Create the base model from the pre-trained model MobileNet V2
base_model = tf.keras.applications.InceptionV3(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet',
                                               classifier_activation="softmax")
base_model.trainable = False

# Let's take a look at the base model architecture
base_model.summary()
```

2. เตรียม GlobalAveragePooling2D layer และ output layer โดยเป็นลักษณะของ Dense และมีจำนวนโหนดเท่ากับจำนวน Class ที่ต้องการทำ Classification และมี Activation Function คือ Softmax Function เพื่อให้แสดงผลลัพธ์เป็นค่า Probability ของแต่ละ Class

```
global_average_layer = GlobalAveragePooling2D()
preds = Dense(10,activation='softmax')#final layer with softmax activation for N classes
```

3. นำทั้ง 3 ส่วนที่เตรียมไว้ด้านบนมารวมกันเป็น model ที่เราจะใช้งาน

```
model = Sequential([
  base_model,
  global_average_layer,
  preds
])
```

4. Compile model โดยปรับค่า Loss = ‘categorical_crossentropy’ ปรับให้มีการคำนวณลักษณะ ‘categorical’ และ Optimize = tf.keras.optimizers.RMSprop(lr=base_learning_rate) และกำหนด base_learning_rate ไว้ที่ 0.0001

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate),
              loss='categorical_crossentropy',
              metrics=['accuracy',f1_m,precision_m, recall_m])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 3, 3, 2048)        21802784
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                20490
=================================================================
Total params: 21,823,274
Trainable params: 20,490
Non-trainable params: 21,802,784
_________________________________________________________________
```

5. การ train ในช่วงแรกได้ทำการปรับค่า epochs = 50

```
initial_epochs = 50
validation_steps = 20

loss, accuracy, f1_score, precision, recall  = model.evaluate(validation_batches, steps = validation_steps)
```

```
20 [==============================] - 2s 110ms/step - loss: 63.0077 - accuracy: 0.1344 - f1_m: 0.1329 - prec
```

```
history = model.fit(train_batches,
                    epochs=initial_epochs,
                    validation_data=validation_batches
                    )
```

6. Fine tuning หลังจากการ train ในช่วงแรกเสร็จแล้ว จะทำการ fine tuning โดย ปรับให้ layer ลำดับที่ 306 ขึ้นไปสามารถ train ต่อได้

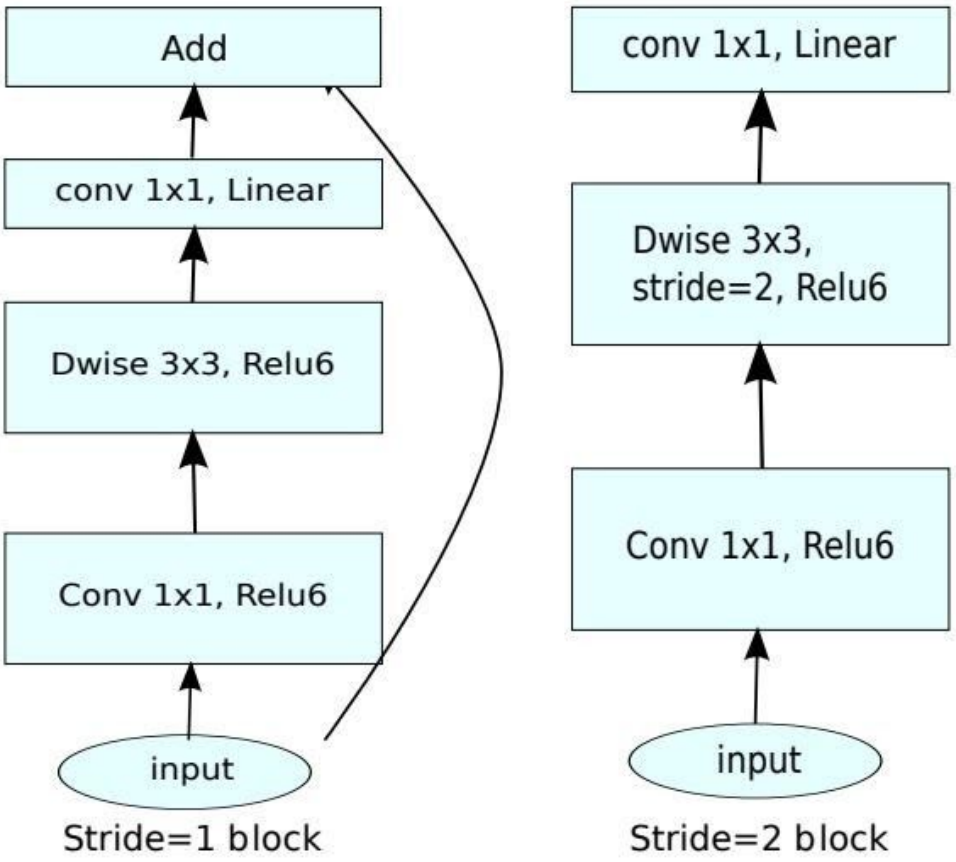
```
fine_tune_at = 306

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

7. ในการทำ fine tuning จะปรับให้ train ต่อจากช่วงแรก epochs = 20 และ initial_epoch = history.epoch[-1] และบันทึกโมเดลเมื่อค่า loss ลดลง โดยทำการบันทึกลงในโฟลเดอร์ Thai Dessert Saved Models/model_inceptionv3_tuned_v1.h5

```
fine_tune_epochs = 30
total_epochs =  initial_epochs + fine_tune_epochs

filepath = saved_path+"model_inceptionv3_tuned_v1.h5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]

history_fine = model.fit(train_batches,
                         epochs=total_epochs,
                         initial_epoch =  history.epoch[-1],
                         validation_data=validation_batches,
                         callbacks=callbacks_list)
```

Method 2 : MobileNetV2

MobileNet คือ โมเดลขนาดเล็ก ที่ทำงานได้เร็ว Latency ต่ำ ใช้พลังงานในการประมวลผลไม่มาก ถูกออกแบบมาสำหรับงานที่มีทรัพยากรจำกัด



ขั้นตอนการ Train

1. เตรียม MobileNetV2 pre-trained model โดยกำหนด input_shape ให้ตรงกับ dataset ที่เตรียมไว้ กำหนด include_top = false เพื่อไม่ให้ทำในส่วนของ Classification แต่จะนำ feature ที่ได้นั้นมาเข้า Neural Network กำหนด weights= ‘imagenet’ คือการนำ weight ที่ทาง Google ได้ pretrain ไว้ก่อนแล้วมาใช้ base_model.trainable = False เพื่อไม่ให้ weight ของ pre-trained model เกิดการเปลี่ยนแปลงในการ train ช่วงแรก

```
IMG_SHAPE = (160, 160, 3)

# Create the base model from the pre-trained model MobileNet V2
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
base_model.trainable = False
```

2. เตรียม GlobalAveragePooling2D layer และ output layer โดยเป็นลักษณะของ Dense และมีจำนวนโหนดเท่ากับจำนวน Class ที่ต้องการทำ Classification และมี Activation Function คือ Softmax Function เพื่อให้แสดงผลลัพธ์เป็นค่า Probability ของแต่ละ Class

```
global_average_layer = GlobalAveragePooling2D()
preds = Dense(10,activation='softmax')#final layer with softmax activation for N classes
```

3. นำทั้ง 3 ส่วนที่เตรียมไว้ด้านบนมารวมกันเป็น model ที่เราจะใช้งาน

```
model = Sequential([
  base_model,
  global_average_layer,
  preds
])
```

4. Compile model โดยปรับค่า Loss = ‘categorical_crossentropy’ ปรับให้มีการคำนวณลักษณะ ‘categorical’ และ Optimize = tf.keras.optimizers.RMSprop(lr=base_learning_rate) และกำหนด base_learning_rate ไว้ที่ 0.0001

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate),
              loss='categorical_crossentropy',
              metrics=['accuracy',f1_m,precision_m, recall_m])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_160 (Functi (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 10)                12810
=================================================================
Total params: 2,270,794
Trainable params: 12,810
Non-trainable params: 2,257,984
_________________________________________________________________
```

5. การ train ในช่วงแรกได้ทำการปรับค่า epochs = 50

```
initial_epochs = 50
validation_steps = 20

loss, accuracy, f1_score, precision, recall  = model.evaluate(validation_batches, steps = validation_steps)
```

```
/20 [==============================] - 273s 14s/step - loss: 2.9440 - accuracy: 0.0625 - f1_m: 0.0107 - precis
```

```
history = model.fit(train_batches,
                    epochs=initial_epochs,
                    validation_data=validation_batches
                    )
```

6. Fine tuning หลังจากการ train ในช่วงแรกเสร็จแล้ว จะทำการ fine tuning โดย ปรับให้ layer ลำดับที่ 152 ขึ้นไปสามารถ train ต่อได้

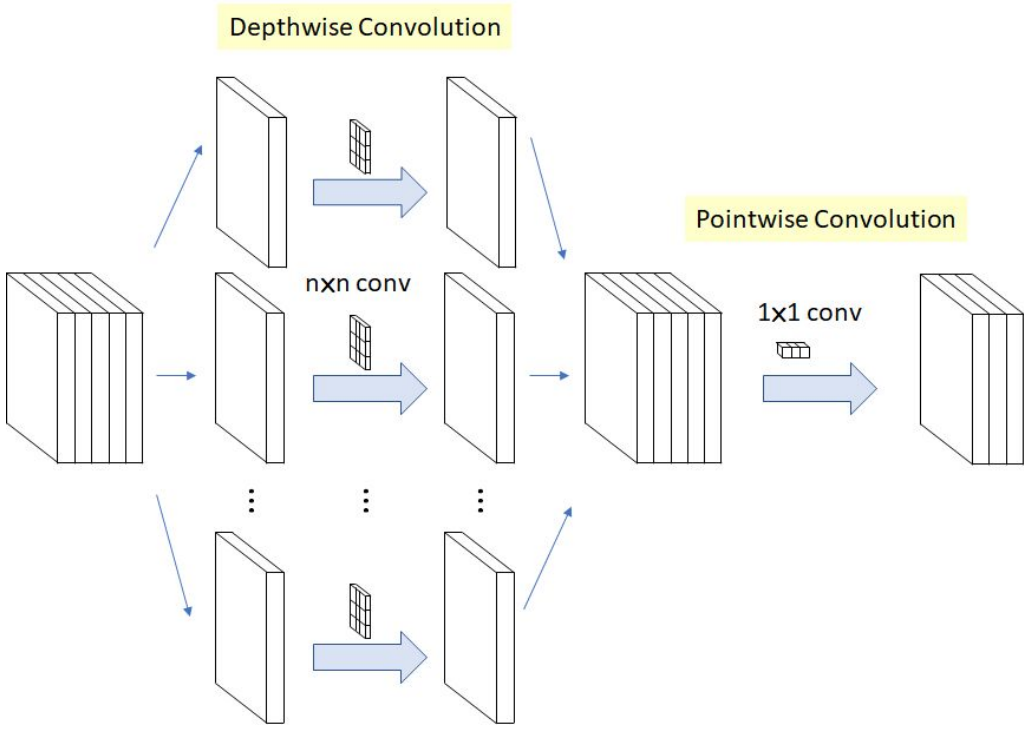
```
fine_tune_at = 152

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

7. ในการทำ fine tuning จะปรับให้ train ต่อจากช่วงแรก epochs = 20 และ initial_epoch = history.epoch[-1] และบันทึกโมเดลเมื่อค่า loss ลดลง โดยทำการบันทึกลงในโฟลเดอร์ Thai Dessert Saved Models/model_imobilenet_tuned_v1.h5

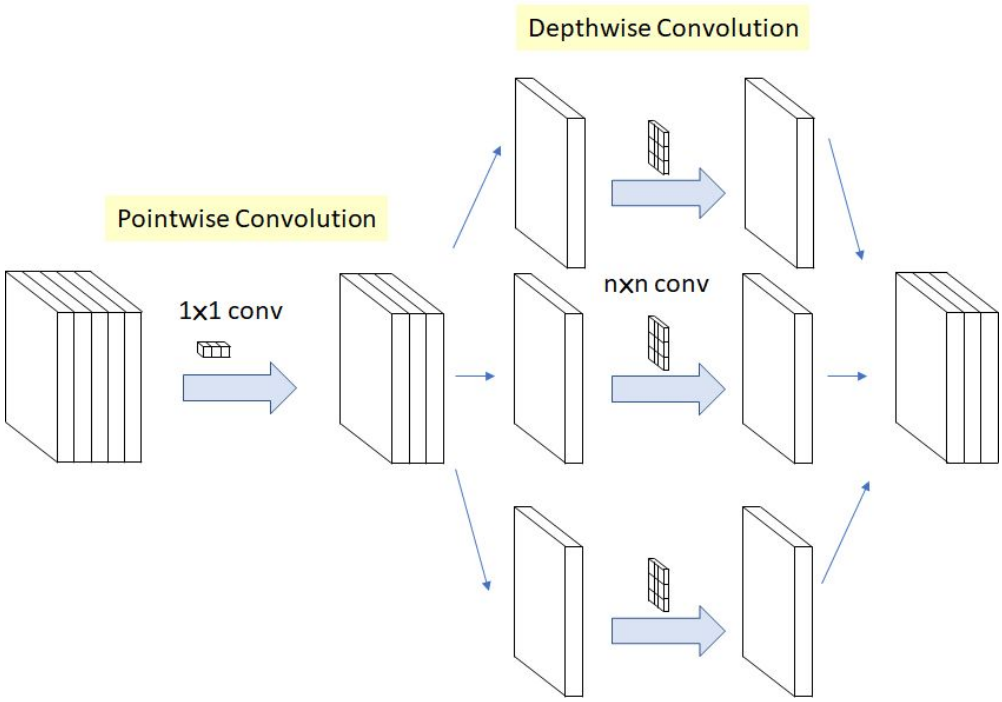
```
fine_tune_epochs = 20
total_epochs =  initial_epochs + fine_tune_epochs

filepath = saved_path+"model_moblienet_tuned_v1.h5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]

history_fine = model.fit(train_batches,
                         epochs=total_epochs,
                         initial_epoch =  history.epoch[-1],
                         validation_data=validation_batches,
                         callbacks=callbacks_list)
```

Method 3 : Xception

Xception ถูกพัฒนาโดย Google เป็น Convolutional neural network ที่มีจำนวน 71 layer Xception ได้ต่อยอดจาก InceptionV3 จากความคิดที่ใช้ Convolution ขนาด 1x1 และต่อด้วย Convolution ขนาด NxN Xception ใช้แนวคิดเดียวกันกับ depthwise separable convolution จากเดิมจะเป็นไปตามรูปที่ 1 เปลี่ยนเป็นดังรูปที่ 2



รูปที่ 1 depthwise separable convolution



รูปที่ 2 modified depthwise separable convolution in Xception

ขั้นตอนการ Train

1. เตรียม Xception pre-trained model โดยกำหนด input_shape ให้ตรงกับ dataset ที่เตรียมไว้ กำหนด include_top = false เพื่อไม่ให้ทำในส่วนของ Classification แต่จะนำ feature ที่ได้นั้นมาเข้า Neural Network กำหนด weights= ‘imagenet’ คือการนำ weight ที่ทาง Google ได้ pretrain ไว้ classifier_activation คือให้ layer สุดท้ายเป็น Softmax ก่อนแล้วมาใช้ base_model.trainable = False เพื่อไม่ให้ weight ของ pre-trained model เกิดการเปลี่ยนแปลงในการ train ช่วงแรก

```
IMG_SHAPE = (160, 160, 3)

# Create the base model from the pre-trained model MobileNet V2
base_model = tf.keras.applications.Xception(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet',
                                                classifier_activation="softmax")
base_model.trainable = False

# Let's take a look at the base model architecture
base_model.summary()
```

2. เตรียม GlobalAveragePooling2D layer และ output layer โดยเป็นลักษณะของ Dense และมีจำนวนโหนดเท่ากับจำนวน Class ที่ต้องการทำ Classification และมี Activation Function คือ Softmax Function เพื่อให้แสดงผลลัพธ์เป็นค่า Probability ของแต่ละ Class

```
global_average_layer = GlobalAveragePooling2D()
preds = Dense(10,activation='softmax')#final layer with softmax activation for N classes
```

3. นำทั้ง 3 ส่วนที่เตรียมไว้ด้านบนมารวมกันเป็น model ที่เราจะใช้งาน

```
modelXecption = Sequential([
  base_model,
  global_average_layer,
  preds
])
```

4. Compile model โดยปรับค่า Loss = ‘categorical_crossentropy’ ปรับให้มีการคำนวณลักษณะ ‘categorical’ และ Optimize = tf.keras.optimizers.RMSprop(lr=base_learning_rate) และกำหนด base_learning_rate ไว้ที่ 0.0001

```
base_learning_rate = 0.0001
modelXecption.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate),
              loss='categorical_crossentropy',
              metrics=['accuracy',f1_m,precision_m, recall_m])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Functional)        (None, 5, 5, 2048)        20861480
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                20490
=================================================================
Total params: 20,881,970
Trainable params: 20,490
Non-trainable params: 20,861,480
_________________________________________________________________
```

5. การ train ในช่วงแรกได้ทำการปรับค่า epochs = 50

```
initial_epochs = 50
validation_steps = 20

loss, accuracy, f1_score, precision, recall  = modelXecption.evaluate(validation_batches, steps = validation_steps)
```

```
/20 [==============================] - 3s 127ms/step - loss: 24.7728 - accuracy: 0.1094 - f1_m: 0.1072 - precision_m:
```

```
history = modelXecption.fit(train_batches,
                    epochs=initial_epochs,
                    validation_data=validation_batches
                    )
```

6. Fine tuning หลังจากการ train ในช่วงแรกเสร็จแล้ว จะทำการ fine tuning โดย ปรับให้ layer ลำดับที่ 129 ขึ้นไปสามารถ train ต่อได้

```
fine_tune_at = 129

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

7. ในการทำ fine tuning จะปรับให้ train ต่อจากช่วงแรก epochs = 20 และ initial_epoch = history.epoch[-1] และบันทึกโมเดลเมื่อค่า loss ลดลง โดยทำการบันทึกลงในโฟลเดอร์ Thai Dessert Saved Models/model_xception_tuned_v3.h5

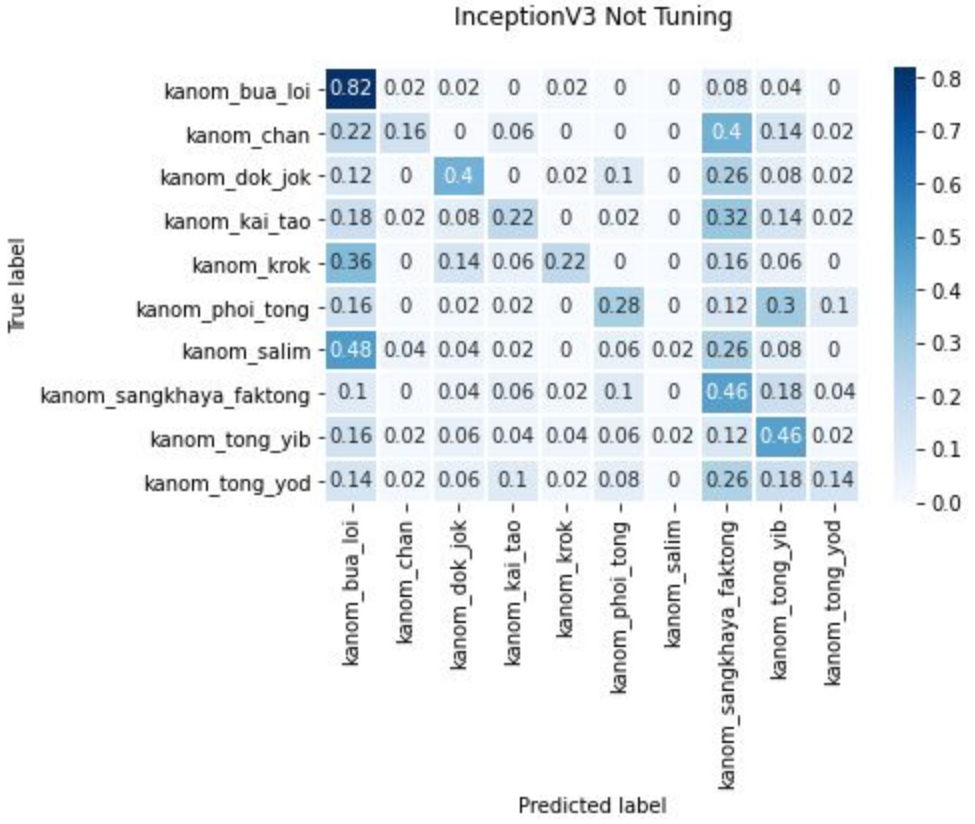
```
fine_tune_epochs = 30
total_epochs =  initial_epochs + fine_tune_epochs

filepath = saved_path+"model_xception_tuned_v3.h5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]

history_fine = modelXecption.fit(train_batches,
                         epochs=total_epochs,
                         initial_epoch =  history.epoch[-1],
                         validation_data=validation_batches,
                         callbacks=callbacks_list)
```

Result

InceptionV3

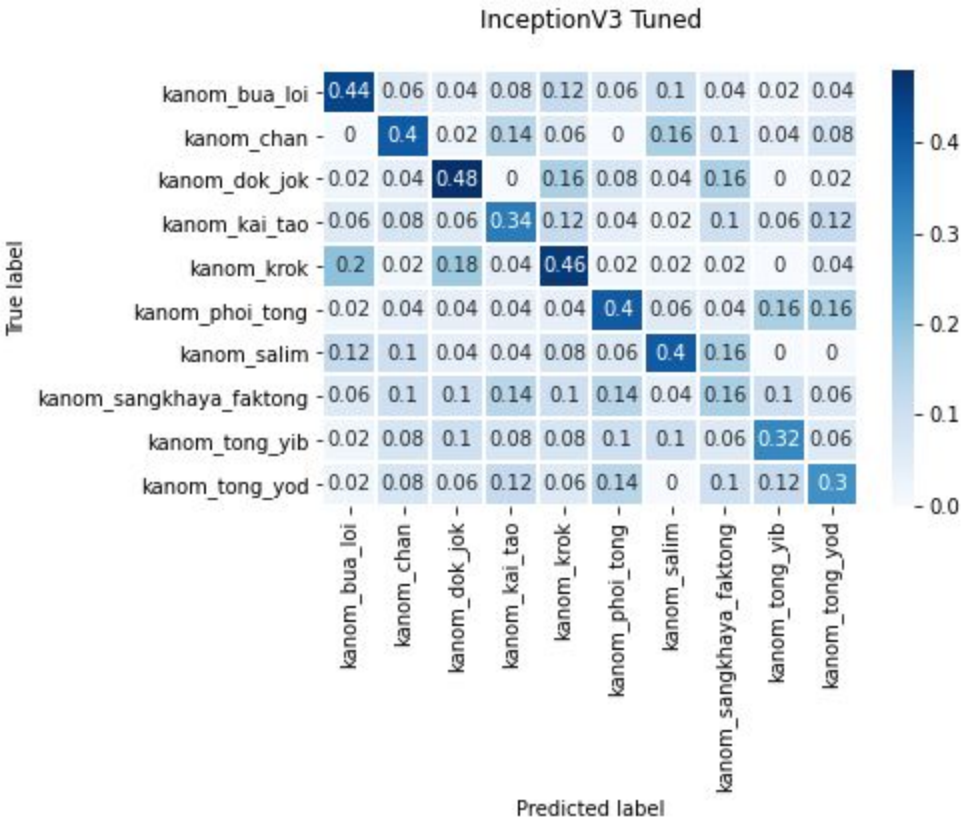
InceptionV3 not tune

	precision	recall	f1-score	support
kanom_bua_loi	0.30	0.82	0.44	50
kanom_chan	0.57	0.16	0.25	50
kanom_dok_jok	0.47	0.40	0.43	50
kanom_kai_tao	0.38	0.22	0.28	50
kanom_krok	0.65	0.22	0.33	50
kanom_phoi_tong	0.40	0.28	0.33	50
kanom_salim	0.50	0.02	0.04	50
kanom_sangkhaya_faktong	0.19	0.46	0.27	50
kanom_tong_yib	0.28	0.46	0.35	50
kanom_tong_yod	0.39	0.14	0.21	50
accuracy			0.32	500
macro avg	0.41	0.32	0.29	500
weighted avg	0.41	0.32	0.29	500



InceptionV3 tuned

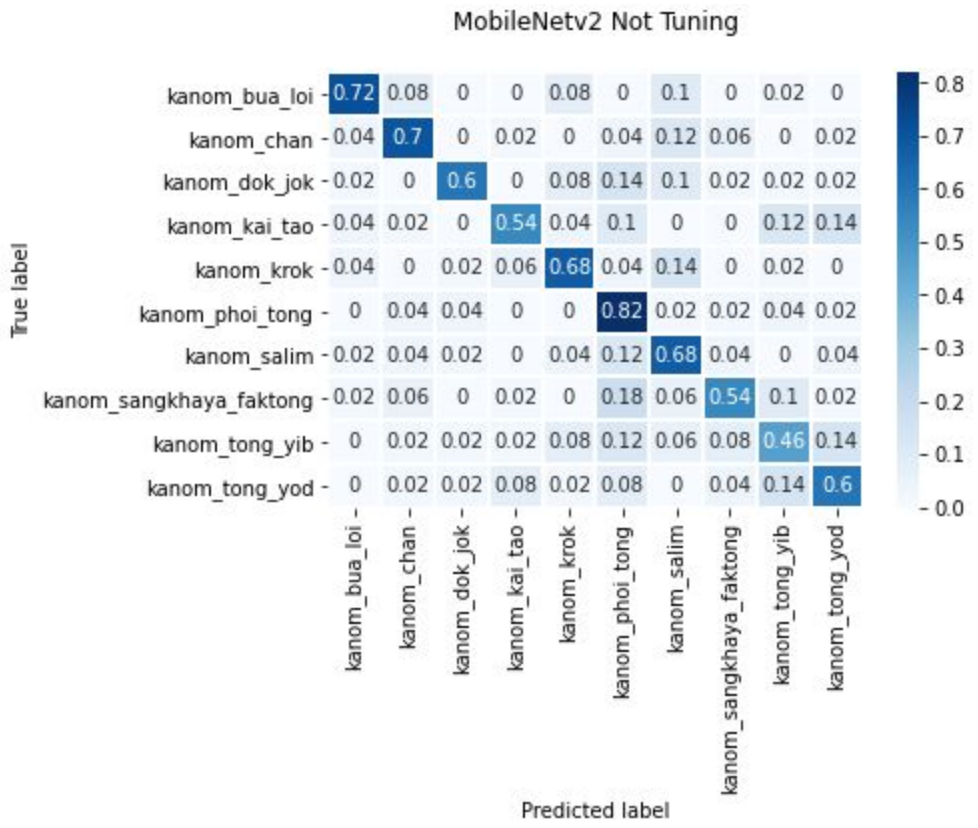
	precision	recall	f1-score	support
kanom_bua_loi	0.46	0.44	0.45	50
kanom_chan	0.40	0.40	0.40	50
kanom_dok_jok	0.43	0.48	0.45	50
kanom_kai_tao	0.33	0.34	0.34	50
kanom_krok	0.36	0.46	0.40	50
kanom_phoi_tong	0.38	0.40	0.39	50
kanom_salim	0.43	0.40	0.41	50
kanom_sangkhaya_faktong	0.17	0.16	0.16	50
kanom_tong_yib	0.39	0.32	0.35	50
kanom_tong_yod	0.34	0.30	0.32	50
accuracy			0.37	500
macro avg	0.37	0.37	0.37	500
weighted avg	0.37	0.37	0.37	500



MobileNetV2

MobileNetV2 not tune

	precision	recall	f1-score	support
kanom_bua_loi	0.80	0.72	0.76	50
kanom_chan	0.71	0.70	0.71	50
kanom_dok_jok	0.83	0.60	0.70	50
kanom_kai_tao	0.73	0.54	0.62	50
kanom_krok	0.67	0.68	0.67	50
kanom_phoi_tong	0.50	0.82	0.62	50
kanom_salim	0.53	0.68	0.60	50
kanom_sangkhaya_faktong	0.68	0.54	0.60	50
kanom_tong_yib	0.50	0.46	0.48	50
kanom_tong_yod	0.60	0.60	0.60	50
accuracy			0.63	500
macro avg	0.66	0.63	0.64	500
weighted avg	0.66	0.63	0.64	500



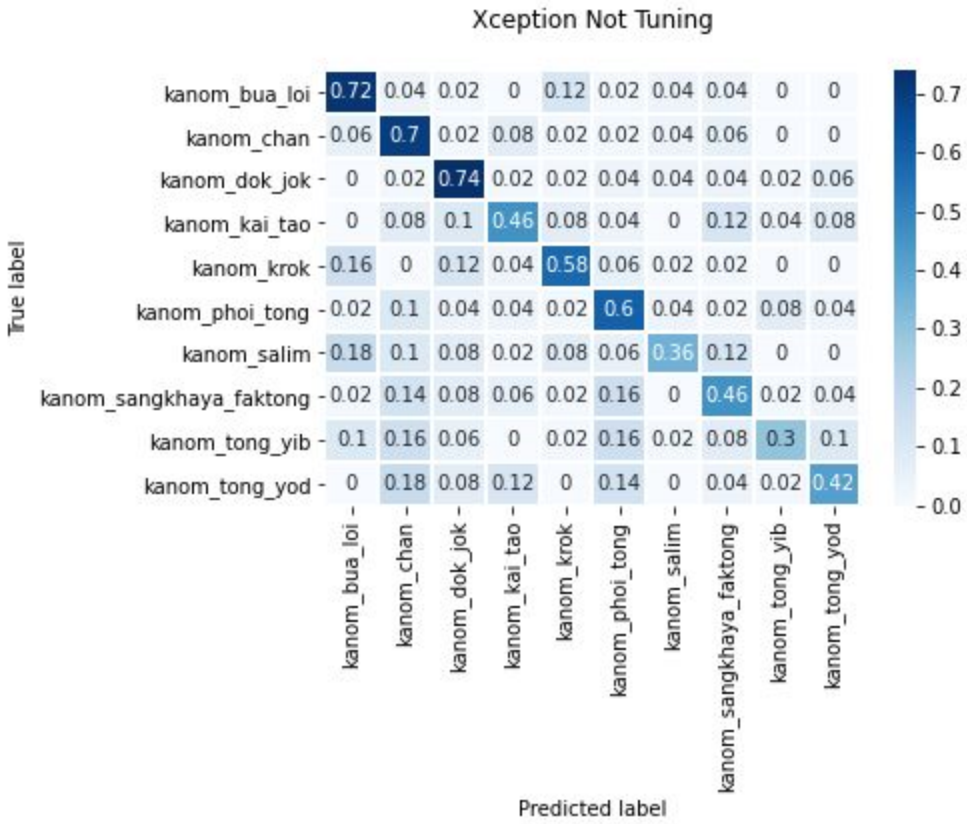
MobileNetV2 Tuned

	precision	recall	f1-score	support
kanom_bua_loi	0.77	0.82	0.80	50
kanom_chan	0.73	0.72	0.73	50
kanom_dok_jok	0.89	0.62	0.73	50
kanom_kai_tao	0.77	0.60	0.67	50
kanom_krok	0.66	0.66	0.66	50
kanom_phoi_tong	0.51	0.86	0.64	50
kanom_salim	0.61	0.76	0.68	50
kanom_sangkhaya_faktong	0.70	0.56	0.62	50
kanom_tong_yib	0.53	0.46	0.49	50
kanom_tong_yod	0.70	0.62	0.66	50
accuracy			0.67	500
macro avg	0.69	0.67	0.67	500
weighted avg	0.69	0.67	0.67	500

Xception

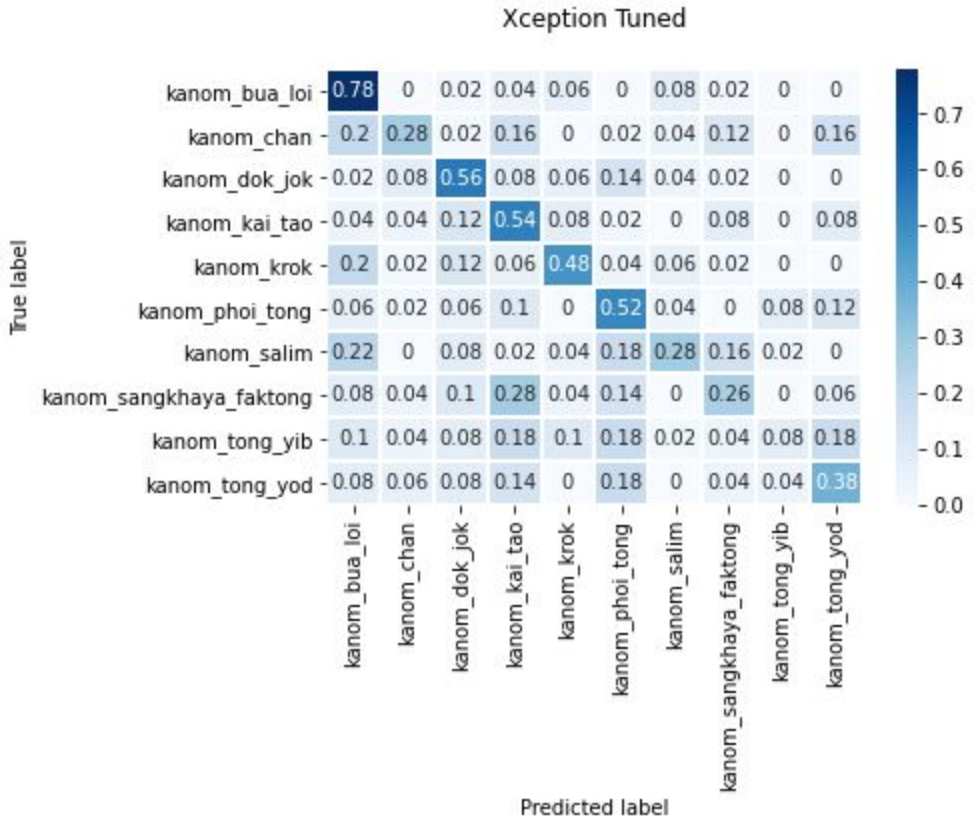
Xception not tune

	precision	recall	f1-score	support
kanom_bua_loi	0.57	0.72	0.64	50
kanom_chan	0.46	0.70	0.56	50
kanom_dok_jok	0.55	0.74	0.63	50
kanom_kai_tao	0.55	0.46	0.50	50
kanom_krok	0.60	0.58	0.59	50
kanom_phoi_tong	0.46	0.60	0.52	50
kanom_salim	0.64	0.36	0.46	50
kanom_sangkhaya_faktong	0.46	0.46	0.46	50
kanom_tong_yib	0.62	0.30	0.41	50
kanom_tong_yod	0.57	0.42	0.48	50
accuracy			0.53	500
macro avg	0.55	0.53	0.52	500
weighted avg	0.55	0.53	0.52	500



Xception tuned

	precision	recall	f1-score	support
kanom_bua_loi	0.44	0.78	0.56	50
kanom_chan	0.48	0.28	0.35	50
kanom_dok_jok	0.45	0.56	0.50	50
kanom_kai_tao	0.34	0.54	0.42	50
kanom_krok	0.56	0.48	0.52	50
kanom_phoi_tong	0.37	0.52	0.43	50
kanom_salim	0.50	0.28	0.36	50
kanom_sangkhaya_faktong	0.34	0.26	0.30	50
kanom_tong_yib	0.36	0.08	0.13	50
kanom_tong_yod	0.39	0.38	0.38	50
accuracy			0.42	500
macro avg	0.42	0.42	0.39	500
weighted avg	0.42	0.42	0.39	500



Discussion

Not Tune

	InceptionV3	MobileNetV2	Xception
Accuracy	0.32	0.63	0.53
Precision	0.41	0.66	0.55
Recall	0.32	0.63	0.53
F1-Score	0.29	0.64	0.52

Tuned

	InceptionV3	MobileNetV2	Xception
Accuracy	0.37	0.67	0.42
Precision	0.37	0.69	0.42
Recall	0.37	0.67	0.42
F1-Score	0.37	0.67	0.39

จากผลลัพธ์ของทั้ง 3 โมเดลจะพบว่า MobileNetV2 มีประสิทธิภาพที่เหนือกว่าโมเดลอื่น ๆ มากในทุกรูปแบบของการประเมินผลไม่ว่าจะเป็น Accuracy, Precision, Recall และ F1-Score อาจจะเป็นเพราะพารามิเตอร์ที่เราใช้นั้นใช้รูปแบบเดียวกันทั้งหมด ซึ่งอาจจะไม่เหมาะสมกับโมเดล InceptionV3 และ Xception

อีกจุดที่น่าสนใจคือหลังจากการ Fine Tuning โมเดล InceptionV3 และ MobileNetV3 โดยรวมมีประสิทธิภาพที่สูงขึ้นโดนอิงจากการประเมินผล แต่โมเดล Xecption มีประสิทธิภาพที่ต่ำลง

ในทุก ๆ ด้าน อาจจะเกิดจากการที่การกำหนด Layer ที่จะ Fine Tuning ไม่เหมาะสม ทำการเกิดเหตุการณ์นี้ขึ้น