โครงงาน **Project-Based Learning 1 (PBL1)**

การจำแนกกล้วย 3 ชนิด และ การจำแนก กะเพรา และ โหระพา ด้วย **Machine Learning**

รายวิชา **968-352 Machine Learning**

วิทยาลัยการคอมพิวเตอร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตภูเก็ต

อาจารย์ประจำวิชา : ผศ.ดร. ขวัญกมล ดิฐกัญจน์

ผู้รับผิดชอบโครงการ

นายรัฐภูมิ รอดนิล รหัสนักศึกษา **6630611025**

หลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาการคอมพิวเตอร์

ภาคเรียนที่ 1 ปีการศึกษา **2568**

Table of Contents

บทที่ 1 บทนำ

1.1 ที่มาและความสำคัญของโครงการ

โครงงานนี้เป็นส่วนหนึ่งของการเรียนรู้เชิงปฏิบัติ (Project-Based Learning) ในวิชา Introduction to Machine Learning ซึ่งมุ่งเน้นการประยุกต์ใช้ Machine Learning ในการแก้ปัญหาจริง
โดยโครงการนี้ตั้งเป้าศึกษาการจำแนกประเภทของพืช 2 กลุ่ม ได้แก่ การจำแนกสายพันธุ์ของกล้วยจากภาพผล (3 สายพันธุ์คือ กล้วยหอม กล้วยน้ำว้า และกล้วยไข่) และ การจำแนกชนิดของใบกะเพราและใบโหระพา จากภาพถ่าย
ทั้งสองกรณีนี้มีความสำคัญในทางการเกษตรและอุตสาหกรรมอาหาร
เนื่องจากการระบุสายพันธุ์กล้วยที่ถูกต้องจะช่วยในการคัดเลือกสายพันธุ์สำหรับการเพาะปลูกหรือการค้า
ส่วนการแยกกะเพรากับโหระพาซึ่งใบมีลักษณะคล้ายกันจะช่วยลดความสับสนในการผลิตและจำหน่ายพืชสมุนไพรไทย การใช้ Machine Learning และ Deep Learning
ในการแก้ปัญหานี้จะช่วยเพิ่มความถูกต้องและรวดเร็วในการจำแนกเมื่อเทียบกับวิธีการดั้งเดิมที่ต้องอาศัยผู้เชี่ยวชาญ

1.2 วัตถุประสงค์

1)เพื่อศึกษาการพัฒนาโมเดล Machine Learning

2) เพื่อพัฒนาโมเดล Machine Learning สำหรับการจำแนกสายพันธุ์ของกล้วยสามสายพันธุ์จากภาพถ่ายของผลกล้วย โดยทดลองใช้เทคนิคการสกัดคุณลักษณะต่าง ๆ และวิธีการจำแนกอย่างน้อย 5 วิธี (เช่น Decision Tree, Random Forest, Naïve Bayes, Logistic Regression และ Neural Network)

3) เพื่อพัฒนาโมเดล Convolutional Neural Network (CNN) ด้วยวิธีการ transfer learning หรือ fine-tuning สำหรับการจำแนกชนิดของใบกะเพราและใบโหระพา และเปรียบเทียบประสิทธิภาพของสถาปัตยกรรม CNN อย่างน้อย 5 แบบ

4) เพื่อเปรียบเทียบประสิทธิภาพระหว่างวิธีการเรียนรู้ของเครื่องแบบดั้งเดิมที่ใช้การสกัดคุณลักษณะเชิงวิศวกรรม (feature engineering) กับวิธีการเรียนรู้เชิงลึกที่ใช้การ transfer learning
ในด้านความแม่นยำในการจำแนกและปัญหาที่อาจเกิดขึ้น เช่น การฝึกโมเดลมากเกินไป (overfitting)

5) เพื่อสรุปข้อดีข้อเสียของแต่ละแนวทางและเสนอแนะแนวทางปรับปรุงสำหรับงานวิจัยหรือการประยุกต์ใช้ในอนาคต

1.3 ขอบเขตของงาน

งานนี้มุ่งเน้นที่การจำแนกภาพนิ่งของพืชในสองกรณีดังกล่าวโดยใช้ข้อมูลชุดเล็กที่จัดเตรียมไว้ ได้แก่ ชุดข้อมูลภาพผลกล้วย 3 สายพันธุ์ (ได้มาจากลิงก์ข้อมูลที่กำหนด) และชุดข้อมูลภาพใบกะเพราและใบโหระพาจำนวนรวม 200 ภาพ (อย่างละ 100 ภาพ จากการรวบรวมออนไลน์) การทดลองถูกจำกัดให้อยู่ในขอบเขตของการใช้โมเดลเรียนรู้ของเครื่องพื้นฐานและโมเดล CNN สถาปัตยกรรมที่เป็นที่รู้จัก ไม่ได้ครอบคลุมไปถึงการสร้างสถาปัตยกรรม CNN ขนาดใหญ่ขึ้นเองหรืองานประยุกต์อื่น ๆ นอกจากนี้ การประเมินประสิทธิภาพจะพิจารณาจากความแม่นยำ (Accuracy) และค่าการสูญเสีย (Loss) บนชุดข้อมูลทดสอบ/validation เป็นหลัก โดยไม่ลงลึกถึงการวัดเชิงลึกอื่น เช่น F1-score หรือ AUC เนื่องจากโจทย์มุ่งเน้นที่ Accuracy/Loss เป็นสำคัญ

บทที่ 2 วิธีดำเนินการ

2.1 ข้อมูลและการเตรียมข้อมูล

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=62,
    # stratify=y  # พยายามให้สัดส่วนคลาสใกล้เคียงกัน
)
```

รูปภาพ 2.1.1 แสดงถึงการ แบ่ง train-test split 80/20 ของ Part machine learning

ในการทดลองแรกใช้ ชุดข้อมูลภาพถ่ายผลกล้วยสามสายพันธุ์ ได้แก่ กล้วยหอม กล้วยน้ำว้า และกล้วยไข่ ซึ่งได้มาจากแหล่งข้อมูลที่กำหนด (ประมาณสายพันธุ์ละ 50 ภาพ) แต่ละภาพแสดงผลกล้วยบนฉากหลังหลากหลาย
การเตรียมข้อมูลเบื้องต้นประกอบด้วยการปรับขนาดรูปภาพให้เหมาะสมและการแปลงภาพเป็นโทนสีเทา (Augmentation) นอกจากนี้อาจมีการปรับแก้ความคมชัดหรือการตัดส่วนเกินของภาพตามความจำเป็น จากนั้นทำการแบ่งชุดข้อมูลออกเป็นสองส่วน ได้แก่ ชุดข้อมูลฝึก (training set) ประมาณ 80% และ ชุดข้อมูลทดสอบ (test set) ประมาณ 20%
โดยเป็นการแบ่งแบบสุ่มโดยรักษาสัดส่วนของแต่ละสายพันธุ์ให้ใกล้เคียงกัน เพื่อใช้ประเมินโมเดลภายหลัง

```
train_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    validation_split=0.20,
    subset="training",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    label_mode='int'
)
val_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    validation_split=0.20,
    subset="validation",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    label_mode='int'
)
```

รูปภาพ 2.1.1 แสดงถึงการ แบ่ง train-test split 80/20 ของ Part CNN

(ผลการรายงานจะเน้นที่ชุดทดสอบ (Validation) เท่านั้นตามโจทย์กำหนด)

สำหรับการทดลองที่สองใช้ ชุดข้อมูลภาพถ่ายใบกะเพราและใบโหระพา ซึ่งรวบรวมมาจากอินเทอร์เน็ตและการถ่ายภาพจริงอย่างละ 100 ภาพ (รวม 200 ภาพ) โดยตัวอย่างภาพทั้งสองประเภทมีลักษณะคล้ายคลึงกัน (เป็นใบโหระพาและใบกะเพราที่มีสีเขียวใกล้เคียงกัน) ข้อมูลถูกแบ่งเป็น ชุดเรียนรู้ 80% และ ชุด validation 20% ในทำนองเดียวกับกรณีแรก อย่างไรก็ตาม ในการใช้โมเดล CNN การแบ่งนี้ทำหน้าที่เสมือนชุดฝึกและชุดตรวจสอบ (validation)

```
def prepare(ds, training=False):
    ds = ds.cache()
    if training:
        ds = ds.shuffle(1000, seed=SEED)
    return ds.prefetch(AUTOTUNE)
```

เนื่องจากไม่มีชุดทดสอบแยกต่างหาก ทั้งนี้ได้ดำเนินการ shuffle ข้อมูลและ ไม่ซ้ำบุคคลต้นทาง (หากภาพหลายภาพมาจากแหล่งเดียวกันหรือถ่ายจากต้นเดียวกัน จะพยายามกระจายในฝึก/ทดสอบให้ต่างกัน) เพื่อป้องกันการเอนเอียงของโมเดล

- ds.cache()
  เก็บผลลัพธ์ของ dataset ไว้ในหน่วยความจำ (หรือดิสก์) หลังจากอ่านครั้งแรก ทำให้รอบถัดไปไม่ต้องอ่านไฟล์จากดิสก์ซ้ำ ช่วยให้การฝึกโมเดลเร็วขึ้นอย่างมีนัยสำคัญ

- ds.shuffle(1000, seed=SEED) (ใช้เฉพาะตอน training)
  ทำการสุ่มลำดับตัวอย่างอย่างต่อเนื่อง โดยใช้ buffer ขนาด 1000 ตัวอย่าง

  ช่วยให้โมเดลไม่เห็นข้อมูลในลำดับเดิม ๆ, ลดการเกิด pattern ที่ทำให้การเรียนรู้ลำเอียง (bias), การใช้ seed ทำให้ผลการสุ่มสามารถทำซ้ำได้

- ds.prefetch(AUTOTUNE)
  สั่งให้ TensorFlow เตรียม batch ถัดไปไว้ล่วงหน้าในขณะที่โมเดลกำลังฝึกกับ batch ปัจจุบัน

  ช่วยลดเวลาว่างของ GPU/CPU, ทำให้การเทรนราบรื่นและรวดเร็วขึ้น

2.2 Feature Extraction

สำหรับการจำแนกสายพันธุ์กล้วย งานนี้ใช้การ Feature Extraction จากภาพแบบ feature engineering ที่ประกอบด้วย Shape, LBP Texture, และ HOG รวมกันเป็นเวกเตอร์คุณลักษณะขนาด 1778 มิติ ดังนี้:

```python
# threshold to get banana shape
_, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU)
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

if len(contours) == 0:
    # no contour
    shape_features = np.zeros(4, dtype=np.float32)
else:
    cnt = max(contours, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(cnt)

    aspect_ratio = w / float(h + 1e-6)
    extent = cv2.contourArea(cnt) / float(w * h + 1e-6)
    hull = cv2.convexHull(cnt)
    solidity = cv2.contourArea(cnt) / (cv2.contourArea(hull) + 1e-6)

    # contour length = banana curvature hint
    perimeter = cv2.arcLength(cnt, True)

    shape_features = np.array([aspect_ratio, extent, solidity, perimeter], dtype=np.float32)
```

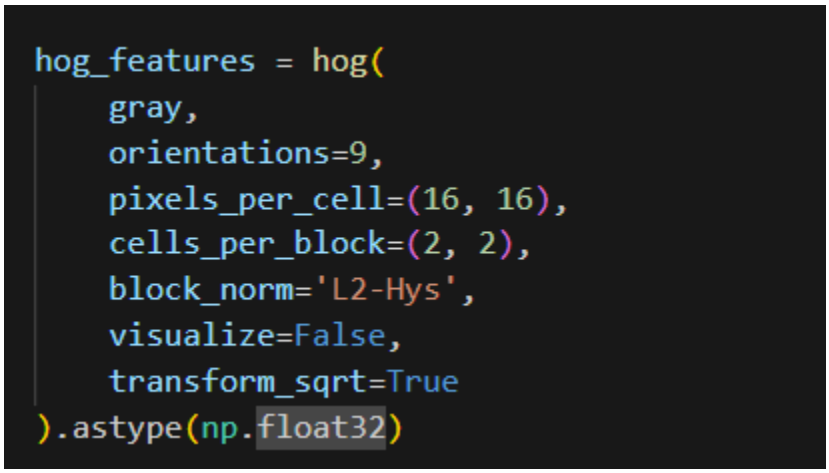
รูป 2.2.1 แสดงโค้ดในส่วนของ Shape features Extraction

Shape Features (4 ค่า): ใช้ Otsu threshold → หา contour → คำนวณ aspect ratio, extent, solidity และ perimeter เพื่อบอกลักษณะรูปทรงและความโค้งของผลกล้วย

```python
# Local Binary Pattern
P = 8      # number of circular neighbors
R = 1      # radius
lbp = local_binary_pattern(gray, P, R, method='uniform')

(lbp_hist, _) = np.histogram(lbp.ravel(),
                             bins=np.arange(0, P + 3),
                             range=(0, P + 2))

lbp_hist = lbp_hist.astype(np.float32)
lbp_hist /= (lbp_hist.sum() + 1e-6)   # normalize

texture_features = lbp_hist
```

รูป 2.2.2 แสดงโค้ดในส่วนของ texture features Extraction

LBP Texture: ใช้ Local Binary Pattern (P=8, R=1, uniform) แล้วสร้างฮิสโตแกรม normalized เพื่ออธิบายลักษณะพื้นผิว



```
hog_features = hog(
    gray,
    orientations=9,
    pixels_per_cell=(16, 16),
    cells_per_block=(2, 2),
    block_norm='L2-Hys',
    visualize=False,
    transform_sqrt=True
).astype(np.float32)
```

รูป 2.2.3 แสดงโค้ดในส่วนของ Hog features Extraction

HOG Features: ใช้ orientations=9, pixels_per_cell=(16,16), cells_per_block=(2,2), block_norm='L2-Hys' เพื่อจับลักษณะขอบและรูปทรงโดยรวมของผลกล้วย

```python
def load_dataset(dataset_dir):
    X = []
    y = []
    class_names = []  # เก็บชื่อคลาสตามลำดับ label

    # วนตามโฟลเดอร์ย่อย (แต่ละโฟลเดอร์ = 1 class)
    for label, class_folder in enumerate(os.listdir(dataset_dir)):
        class_path = os.path.join(dataset_dir, class_folder)
        if not os.path.isdir(class_path):
            continue

        class_names.append(class_folder)

        for filename in os.listdir(class_path):

            img_path = os.path.join(class_path, filename)

            # อ่านรูป

            img = cv2.imread(img_path)
            if img is None:
                continue  # เผื่อไฟล์แปลกๆ

            # สร้างฟีเจอร์
            feat = extract_features(img)

            X.append(feat)
            y.append(label)

    X = np.array(X, dtype=np.float32)
    y = np.array(y, dtype=np.int32)

    return X, y, class_names
```

รูป 2.2.4 แสดง function ที่ใช้เพื่อดึงรูปภาพ พร้อมกับสร้าง feature ไปในตัว

```python
X, y, class_names = load_dataset(DATASET_DIR)

print("รูปทั้งหมด:", X.shape[0])
print("ขนาด feature ต่อรูป:", X.shape[1])
print("คลาสที่พบ:", class_names)  # เช่น ['กล้วยไข่', 'กล้วยน้ำว้า', 'กล้วยหอม']
```

```
รูปทั้งหมด: 156
ขนาด feature ต่อรูป: 1778
คลาสที่พบ: ['egg', 'hom', 'numwa']
```

รูป 2.2.5 แสดงขนาดของ feature และ รูปทั้งหมด

ฟีเจอร์ทั้งหมดถูกรวมเป็นเวกเตอร์เดียวที่มีขนาด 1778 มิติ ใช้เป็นอินพุตของโมเดล Machine Learning สำหรับการจำแนกสายพันธุ์กล้วย

2.3 การทำ Augmentation

Machine Learning

```python
def random_augment(img):
    """Apply a small random augmentation to img (BGR uint8)."""
    out = img.copy()
    h, w = out.shape[:2]

    ops = []

    # flip horizontally
    ops.append(lambda x: cv2.flip(x, 1))
    # rotate by small angle
    def rot(x):
        angle = random.uniform(-20, 20)
        M = cv2.getRotationMatrix2D((w/2, h/2), angle, 1.0)
        return cv2.warpAffine(x, M, (w, h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT)
    ops.append(rot)

    # small translation
    def shift(x):
        tx = int(random.uniform(-0.05, 0.05) * w)
        ty = int(random.uniform(-0.05, 0.05) * h)
        M = np.float32([[1, 0, tx], [0, 1, ty]])
        return cv2.warpAffine(x, M, (w, h), borderMode=cv2.BORDER_REFLECT)
    ops.append(shift)

    # apply 1-2 random ops
    n_ops = random.choice([1, 2])
    chosen = random.sample(ops, n_ops)
    # print(chosen)
    for f in chosen:
        out = f(out)

    return out
```

รูปภาพ 2.3.1 แสดงถึงการทำฟังก์ชัน random_augment

ในงานนี้มีการออกแบบฟังก์ชัน random_augment เพื่อสร้างความหลากหลายให้กับข้อมูลภาพก่อนนำไปใช้ฝึกโมเดล Machine Learning โดยฟังก์ชันนี้จะสุ่มเลือก 1–2 เทคนิคการแปลงภาพจากชุดของการแปลง (augmentation operations) ที่เตรียมไว้ เพื่อเพิ่มความแข็งแกร่งให้โมเดลและลดโอกาสเกิด overfitting รายละเอียดของแต่ละการแปลงมีดังนี้

```python
# flip horizontally
ops.append(lambda x: cv2.flip(x, 1))
# rotate by small angle
```

รูปภาพ 2.3.2 แสดงถึงการทำ horizontal Flip Augmentation

1. **การกลับภาพในแนวนอน (Horizontal Flip)**
   เทคนิคนี้ใช้คำสั่ง cv2.flip(x, 1) เพื่อกลับภาพจากซ้ายไปขวา โดยพิกัดของภาพจะถูกสลับในแกนแนวนอน ทำให้วัตถุในภาพมีลักษณะเหมือนมองจากอีกด้านหนึ่ง

```python
def rot(x):
    angle = random.uniform(-20, 20)
    M = cv2.getRotationMatrix2D((w/2, h/2), angle, 1.0)
    return cv2.warpAffine(x, M, (w, h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT)
ops.append(rot)
```

รูปภาพ 2.3.3 แสดงถึงการทำ Random Small Rotation

2. การหมุนภาพด้วยมุมสุ่ม (Random Small Rotation)

ในฟังก์ชัน rot(x) มีการสุ่มมุมระหว่าง -20 ถึง +20 องศา จากนั้นใช้ cv2.getRotationMatrix2D เพื่อสร้างเมทริกซ์สำหรับหมุนภาพรอบจุดกึ่งกลางภาพ แล้วใช้ cv2.warpAffine เพื่อแปลงภาพตามเมทริกซ์ดังกล่าว โดยใช้ BORDER_REFLECT เพื่อสะท้อนขอบภาพแทนการสร้างขอบดำ

```python
# small translation
def shift(x):
    tx = int(random.uniform(-0.05, 0.05) * w)
    ty = int(random.uniform(-0.05, 0.05) * h)
    M = np.float32([[1, 0, tx], [0, 1, ty]])
    return cv2.warpAffine(x, M, (w, h), borderMode=cv2.BORDER_REFLECT)
ops.append(shift)
```

รูปภาพ 2.3.4 แสดงถึงการทำ Random Translation

3. การเลื่อนภาพแบบสุ่ม (Random Translation)

ฟังก์ชัน shift(x) สุ่มระยะการเลื่อนในแนวนอน (tx) และแนวตั้ง (ty) ประมาณ ±5% ของขนาดภาพ แล้วใช้เมทริกซ์แปลงเชิงเส้น (affine transform)

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix}$$

เพื่อนำมาขยับตำแหน่งของภาพโดยรวม ก่อนนำไปประมวลผลด้วย cv2.warpAffine

```python
# apply 1-2 random ops
n_ops = random.choice([1, 2])
chosen = random.sample(ops, n_ops)
# print(chosen)
for f in chosen:
    out = f(out)

return out
```

รูปภาพ 2.3.5 แสดงถึงการทำงานของการสุ่มเลือก augment

หลังจากได้แล้ว ก็จะมีการสุ่มเลือกและเรียงลำดับการ Augment (Random Selection of Operations) ฟังก์ชันจะสุ่มจำนวน augmentation ที่ต้องใช้ (1 หรือ 2) จากนั้นเลือกชุดของฟังก์ชันแปลงภาพแบบไม่ซ้ำกันด้วย random.sample แล้วนำไปประยุกต์กับภาพตามลำดับที่สุ่มได้

ตัวอย่างผลที่อาจเกิดขึ้น

- ใช้เพียงการหมุนอย่างเดียว
- ใช้การกลับภาพ + การเลื่อนภาพ
- ใช้การเลื่อนภาพ + การหมุนภาพ ฯลฯ

CNN

```
9  data_augment = keras.Sequential([
8      layers.RandomFlip("horizontal"),
7      layers.RandomRotation(0.10),
6      layers.RandomZoom(0.12),
5      layers.RandomContrast(0.10),
4      layers.RandomTranslation(0.08, 0.08),
3      layers.RandomBrightness(0.12),
2      layers.GaussianNoise(0.01),
1  ], name="augment")
```

ในส่วนนี้มีการสร้างเลเยอร์ data_augment แบบ Sequential ซึ่งทำหน้าที่แปลงภาพแบบสุ่มระหว่างการฝึกโมเดล โดยแต่ละเลเยอร์มีบทบาทดังนี้

1. **RandomFlip("horizontal") – การกลับภาพในแนวนอน**

   ช่วยเพิ่มความหลากหลายของทิศทางวัตถุ

   ทำให้โมเดลไม่ยึดติดกับด้านซ้าย/ขวาของภาพ

2. **RandomRotation(0.15) – การหมุนภาพแบบสุ่ม**

   หมุนภาพภายในช่วงประมาณ ±15% ของวงรอบ (ประมาณ ±27 องศา)

   ทำให้โมเดลทนต่อมุมเอียงของวัตถุในภาพ

3. **RandomZoom(0.15) – การซูมเข้า/ออกแบบสุ่ม**

   จำลองสถานการณ์ที่กล้องถ่ายภาพระยะใกล้หรือไกลแตกต่างกัน

   ช่วยให้โมเดลเข้าใจว่าวัตถุเดียวกันอาจมีขนาดต่างกันในภาพ

4. **RandomContrast(0.15) – การปรับคอนทราสต์แบบสุ่ม**

   จำลองสภาพแสงที่แตกต่าง เช่น ภาพมืด ภาพสว่าง

   ช่วยให้โมเดลแยกแยะวัตถุได้ดีขึ้นแม้สภาพแสงไม่สม่ำเสมอ

5. **RandomTranslation(0.1, 0.1) – การเลื่อนตำแหน่งวัตถุแบบสุ่ม**

   เลื่อนภาพในแนวนอนและแนวตั้งประมาณ 10% ของขนาดภาพ

   ลดการยึดติดว่าควรวัตถุต้องอยู่กลางภาพเสมอ

2.4 การพัฒนาโมเดล Machine Learning สำหรับการจำแนกกล้วย การพัฒนาโมเดล Machine Learning สำหรับการจำแนกกล้วย

หลังจากได้เวคเตอร์คุณลักษณะของผลกล้วยแต่ละผล เราได้สร้างโมเดลจำแนก (Classifier) โดยทดลองอย่างน้อย 5 วิธีการ ดังต่อไปนี้:

- **Decision Tree**: โครงสร้างของต้นไม้จะทำการแบ่งข้อมูลออกเป็นกลุ่มตามเงื่อนไขของคุณลักษณะ เช่น shape, texture, หรือ hog feature จนกระทั่งโหนดใบสุดท้ายจำแนกกล้วยได้ครบทุกชนิดการฝึกโมเดลใช้เกณฑ์ เช่น Gini index หรือ entropy เพื่อเลือกคุณลักษณะที่ดีที่สุดในการแยกข้อมูลที่แต่ละโหนด แม้ว่าโมเดล Decision Tree จะสามารถตีความผลการตัดสินใจได้ง่ายและเป็นมิตรกับการอธิบายผลลัพธ์

- **Random Forest**: เป็นการรวมรุ่นของต้นไม้ตัดสินใจจำนวนหลายต้น โมเดลสุ่มป่าจะสร้างต้นไม้หลาย ๆ ต้นจากการสุ่มตัวอย่างข้อมูลและสุ่มเลือกชุดคุณลักษณะบางส่วนในการแบ่งแต่ละโหนด ผลการจำแนกสุดท้ายจะใช้การโหวตส่วนใหญ่จากต้นไม้ทั้งหมด Random Forest มีความสามารถในการลดความลำเอียงของต้นไม้เดี่ยวและลดการ overfitting ทำให้ได้ความแม่นยำสูงกว่าต้นไม้ตัดสินใจเดี่ยว ผลการทดลองใช้ Random Forest คาดว่าจะให้ประสิทธิภาพที่ดีในการจำแนกสายพันธุ์กล้วยเนื่องจากรวมการตัดสินใจจากลักษณะหลายด้าน

- **Naïve Bayes**: เราใช้ตัวจำแนก Naïve Bayes แบบ Gaussian สำหรับคุณลักษณะต่อเนื่อง โดยคำนวณความน่าจะเป็นของคุณลักษณะของภาพที่จะอยู่ในแต่ละประเภทกล้วย โมเดลนี้เรียนรู้ได้รวดเร็วและต้องการข้อมูลในการฝึกน้อย ทั้งยังสามารถจัดการกับคุณลักษณะที่มีมิติสูงได้ดี อย่างไรก็ตาม ความแม่นยำอาจต่ำกว่าวิธีอื่นหาก feature ไม่มีความ independence และ feature มีความสัมพันธ์กัน

- **Logistic Regression**: Logistic Regression จะสร้างสมการเชิงเส้นของ vector, feature และ weight เพื่อประมาณความน่าจะเป็นที่ภาพจะเป็นกลุ่มเป้าหมายแต่ละชนิด จากนั้นเลือกประเภทที่มีความน่าจะเป็นสูงสุดเป็นผลลัพธ์ โมเดลนี้สามารถตีความน้ำหนักของคุณลักษณะว่าอันไหนสำคัญต่อการทำนาย อย่างไรก็ตาม เนื่องจากความสัมพันธ์เป็นเชิงเส้น โมเดลอาจมีความยากลำบากถ้าความสัมพันธ์ที่แท้จริงระหว่างคุณลักษณะกับประเภทกล้วยเป็นแบบไม่เชิงเส้น

- **Neural Network**: เราใช้โครงข่ายประสาทเทียมชนิด Multilayer Perceptron (MLP) ซึ่งประกอบด้วย ชั้นซ่อน 2 ชั้น ที่มีจำนวนโหนดเท่ากับ 256 และ 128 ตามลำดับ ใช้ฟังก์ชันกระตุ้นแบบ tanh และใช้ solver แบบ L-BFGS สำหรับการหา gradient ซึ่งเหมาะกับงานที่ข้อมูลมีขนาดไม่ใหญ่มาก โครงสร้างนี้มีความสามารถในการเรียนรู้ความสัมพันธ์เชิงซับซ้อนระหว่างคุณลักษณะของภาพ (feature vector) กับประเภทของกล้วยทั้ง 3 ชนิดได้ดีนอกจากนี้ยังมีการตั้งค่า max_iter = 1000 เพื่อให้สามารถฝึกโมเดลได้นานพอสำหรับ convergence อย่างไรก็ตาม โมเดลอาจเสี่ยงต่อ overfitting ได้เช่นกัน โดยเฉพาะหากจำนวนโหนดมากเกินกว่าที่ข้อมูลรองรับ จึงมีการตรวจสอบผลการเรียนรู้โดยใช้ชุด validation เพื่อควบคุมการเรียนรู้ให้อยู่ในช่วงที่เหมาะสม

ผลลัพธ์ที่ได้จากโมเดลแต่ละตัวจะนำมาเปรียบเทียบกันในบทที่ 3

2.5 การใช้ CNN สำหรับการจำแนกใบกะเพรา–โหระพา

สำหรับปัญหาการจำแนกใบกะเพรากับใบโหระพาซึ่งมีลักษณะใกล้เคียงกัน งานนี้เลือกใช้วิธี Deep Learning โดยอาศัยโครงข่าย Convolutional Neural Network (CNN) ที่ผ่านการฝึกมาก่อนบนชุดข้อมูลขนาดใหญ่ (pre-trained on ImageNet) แล้วนำมาปรับใช้กับข้อมูลของเรา (transfer learning).

```python
def head_block(x, num_classes, dropout=0.3, l2_weight=1e-4, MIXED=False, units=128):
    # 1) สรุป feature จาก CNN backbone
    x = layers.GlobalAveragePooling2D()(x)

    # 2) ทำให้ distribution ของ feature นิ่งขึ้น (ช่วยทั้ง train/val)
    x = layers.BatchNormalization()(x)

    # 3) Dense ซ่อน 1 ชั้น (พอแล้วสำหรับ data เล็ก) + L2
    x = layers.Dense(
        units,
        use_bias=False,
        kernel_regularizer=regularizers.l2(l2_weight),
        dtype='float32' if MIXED else None,
    )(x)

    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    # 4) Dropout กัน overfit
    x = layers.Dropout(dropout)(x)

    # 5) ชั้น output
    logits = layers.Dense(
        num_classes,
        kernel_regularizer=regularizers.l2(l2_weight),
        dtype='float32' if MIXED else None,
    )(x)

    out = layers.Activation('softmax', dtype='float32' if MIXED else None)(logits)
    return out
```

รูปภาพ 2.5.1 แสดงถึงฟังก์ชั่น ของ head block

สำหรับสถาปัตยกรรมส่วนใหญ่ (ยกเว้น Xception ซึ่งใช้ head block ที่เรียบง่ายกว่าเล็กน้อย) งานนี้ใช้ head block โมเดลร่วมกันผ่านฟังก์ชัน head_block โดยมีโครงสร้างดังนี้

1. ใช้ Global Average Pooling สรุปคุณลักษณะเชิงพื้นที่จาก convolutional backbone
2. ทำ Batch Normalization เพื่อลดการกระจายของค่า feature และช่วยให้การฝึกนิ่งขึ้น
3. ใช้ชั้น Dense ซ่อน 1 ชั้น ขนาด 128 หน่วย พร้อมตัวปรับน้ำหนักแบบ L2 (weight decay) เพื่อเพิ่มความสามารถในการเรียนรู้ความสัมพันธ์ที่ไม่เชิงเส้น
4. ทำ Batch Normalization และใช้ฟังก์ชันกระตุ้นแบบ ReLU
5. ใช้ Dropout 0.3 ช่วยลดโอกาสการเกิด overfitting
6. ปิดท้ายด้วยชั้น Dense ขนาด 2 หน่วย (แทนคลาส holy basil และ Thai basil) ตามด้วยฟังก์ชัน Softmax เพื่อให้ได้ความน่าจะเป็นของแต่ละคลาส

งานนี้เราได้สร้าง function สำหรับการ build model ไว้เพื่อให้ง่ายต่อการนำมาใช้ซ้ำ

```python
def build_model(base, model_name,model_lable):

    base.trainable = False

    inputs = keras.Input(shape=IMG_SIZE + (3,))
    x = data_augment(inputs, training=True)
    x = keras.applications.__getattribute__(model_name).preprocess_input(x)
    x = base(x, training=False)

    outputs = head_block(
        x,
        num_classes=num_classes,
        dropout=0.3,
        l2_weight=1e-4,
        MIXED=MIXED,
        units=128,
    )

    return keras.Model(inputs, outputs, name=model_lable)
```

รูปภาพ 2.5.2 แสดงถึงฟังก์ชั่นการสร้าง model

```python
base.trainable = False
```

รูปภาพ 2.5.3แสดงถึง Freeze base

- กำหนด base.trainable = False เพื่อ freeze พารามิเตอร์ของชั้น convolution ทั้งหมด ไม่ให้เรียนรู้ใหม่ในช่วงแรก

```python
inputs = keras.Input(shape=IMG_SIZE + (3,))
x = data_augment(inputs, training=True)
```

รูปภาพ 2.5.4 แสดงถึงการตั้งค่าของการสร้าง Model

- สร้าง input tensor ที่มีขนาดรูปแบบ (height, width, 3)
- ใช้ data_augment (ซึ่งเป็น Sequential layer) ทำการหมุน กลับด้าน ซูม และเลื่อนตำแหน่งภาพแบบสุ่มในทุก batch

```python
x = keras.applications.__getattribute__(model_name).preprocess_input(x)
```

รูปภาพ 2.5.5 แสดงถึงการปรับภาพของการสร้าง model

- ปรับค่าสีของภาพให้เป็น domain ที่เหมาะกับ Model เช่น
    - การปรับช่วงค่าสีให้อยู่ในช่วงที่โมเดลเรียนรู้ไว้

- ช่วยให้ฟีเจอร์ที่ผ่าน convolution ถูกแปลผลได้ถูกต้อง

```
x = base(x, training=False)
```

รูปภาพ 2.5.6 แสดงถึงการตั้งค่าของการสร้าง mobilenetv2

- ใช้ Model เป็น feature extractor เพื่อดึงคุณลักษณะเชิงลึก (high-level features) ของภาพ
- ระบุ training=False เพื่อปิด dropout และ batch normalization ที่พบในบางโมเดลฐาน

```
outputs = head_block(
    x,
    num_classes=num_classes,
    dropout=0.3,
    l2_weight=1e-4,
    MIXED=MIXED,
    units=128,
)
```

รูปภาพ 2.5.7 แสดงถึงการใช้งาน head_block

งานทดลองใช้สถาปัตยกรรม CNN มาตรฐานจำนวน 5 แบบ ได้แก่

```
def build_mobilenetv2():
    base = keras.applications.MobileNetV2(
        include_top=False,
        weights="imagenet",
        input_shape=IMG_SIZE + (3,)
    )
    return build_model(base,"mobilenet_v2","mobilenetv2")
```

รูปภาพ 2.5.8 แสดงถึงการใช้งาน mobilenetv2

MobileNetV2 – โมเดลขนาดเล็กที่ออกแบบให้ประหยัดพารามิเตอร์และคำนวณเร็ว เหมาะกับงานบนอุปกรณ์พกพา โดยใช้ชั้น depthwise separable convolution เพื่อลดจำนวนตัวแปร

```python
def build_xception(input_shape=IMG_SIZE + (3,), num_classes=num_classes):
    base = keras.applications.Xception(
        include_top=False,
        weights="imagenet",
        input_shape=input_shape
    )

    return build_model(base,"xception","xception")
```

รูปภาพ 2.5.9 แสดงถึงการใช้งาน Xception

Xception – โมเดลที่ขยายแนวคิดของ Inception โดยใช้ depthwise separable convolution ทั้งเครือข่าย ช่วยให้เรียนรู้ลักษณะเชิงพื้นที่ได้ละเอียดขึ้นพร้อมจำนวนพารามิเตอร์ที่คุมได้

```python
def build_efficientnetb0():
    base = keras.applications.EfficientNetB0(
        include_top=False,
        weights="imagenet",
        input_shape=IMG_SIZE + (3,)
    )
    return build_model(base,"efficientnet","efficientnetb0")
```

รูปภาพ 2.5.10 แสดงถึงการใช้งาน EfficientNetB0

EfficientNetB0 – โมเดลที่ใช้แนวคิด compound scaling ในการปรับ “ความลึก–ความกว้าง–ขนาดภาพ” อย่างสมดุล ทำให้รุ่นเล็กอย่าง B0 ยังให้ความแม่นยำสูงเมื่อเทียบกับขนาดโมเดล

```python
def build_resnet50v2():
    base = keras.applications.ResNet50V2(
        include_top=False,
        weights="imagenet",
        input_shape=IMG_SIZE + (3,)
    )
    return build_model(base,"resnet_v2","resnet50v2")
```

รูปภาพ 2.5.11 แสดงถึงการใช้งาน ResNet50V2

ResNet50V2 – เครือข่ายแบบ Residual Network 50 ชั้น ที่มี skip connections ช่วยแก้ปัญหา gradient หายไปเมื่อโมเดลลึกมาก และเรียนรู้ลักษณะที่ซับซ้อนได้ดี

```python
def build_nasnetmobile():
    base = keras.applications.NASNetMobile(
        include_top=False,
        weights="imagenet",
        input_shape=IMG_SIZE + (3,)
    )
    return build_model(base,"nasnet","nasnetmobile")
```

รูปภาพ 2.5.12 แสดงถึงการใช้งาน NASNetMobile

NASNetMobile – โมเดลขนาดเล็กที่ได้จากเทคนิค Neural Architecture Search ออกแบบให้เหมาะกับงานบนอุปกรณ์ที่ทรัพยากรจำกัด แต่ยังคงประสิทธิภาพที่ดี

ขั้นตอนการฝึกแบ่งเป็นสองช่วงหลัก

ช่วงที่ 1 – Frozen backbone

```python
    model = build_fn()
    compile_model(model, lr=3e-4)

    ckpt = keras.callbacks.ModelCheckpoint(
        filepath='best.keras',
        monitor='val_loss',
        save_best_only=True
    )
    history = model.fit(
        train_ds_aug,
        validation_data=val_ds_prep,
        epochs=EPOCHS,
        callbacks=[ckpt],
        verbose=2
    )
```

รูปภาพ 2.5.13 แสดงถึง ช่วงที่ 1 ของ function train_and_eval

ในช่วงแรกของการฝึกโมเดล ได้ใช้เทคนิค Transfer Learning โดยกำหนดให้ ส่วนของ backbone ที่เป็น convolutional layers Freeze เพื่อป้องกันไม่ให้มีการอัปเดต Weight ของชั้นฐานที่ได้มาจาก ImageNet และฝึกเฉพาะส่วนหัว (classification head) ที่สร้างขึ้นใหม่เท่านั้น ซึ่งช่วยลดความเสี่ยงของ overfitting เมื่อจำนวนข้อมูลมีจำกัด

```python
def compile_model(model, lr=1e-3):
    opt = keras.optimizers.Adam(learning_rate=lr)
    model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```python
model = build_fn()
compile_model(model, lr=3e-4)
```

รูปภาพ 2.5.14 แสดงถึงโค้ดที่เริ่มต้นการ train model

- Optimizer: ใช้ตัวปรับพารามิเตอร์แบบ Adam
- Learning rate: เริ่มต้นที่ 3×10^{-4} (3e-4)
- Loss function: ใช้ sparse_categorical_crossentropy ซึ่งเหมาะกับ label แบบ integer
- Metric: ใช้ค่า accuracy เป็นตัวชี้วัดหลัก

```python
ckpt = keras.callbacks.ModelCheckpoint(
    filepath='best.keras',
    monitor='val_loss',
    save_best_only=True
)
```

รูปภาพ 2.5.15 แสดงถึงโค้ดตั้งค่า callback ของการ train

- บันทึกเฉพาะโมเดลที่ให้ค่า **val_loss ต่ำที่สุด**
- ป้องกันการสูญเสียโมเดลที่ดีที่สุดในกรณีที่ epoch ถัดไปให้ผลลัพธ์แย่ลง
- ใช้ไฟล์ best.keras เป็นจุดตรวจ (checkpoint)

```python
history = model.fit(
    train_ds_aug,
    validation_data=val_ds_prep,
    epochs=EPOCHS,
    callbacks=[ckpt],
    verbose=2
)
```

รูปภาพ 2.5.16 แสดงถึงโค้ดที่เริ่มต้น train model

- ฝึกทั้งหมด **100 epochs** (ตามค่าที่ตั้งในตัวแปร EPOCHS)
- ใช้ชุด training augmented และ validation ที่ผ่านการเตรียมข้อมูลแล้ว
- บันทึกค่า loss และ accuracy ของทั้ง training และ validation ลงในตัวแปร history

ช่วงที่ 2 – ปรับจูนแบบละเอียด (Fine-tuning)

```python
if hasattr(model.layers[2], 'trainable') and hasattr(model.layers[2], 'layers'):
    # Unfreeze last ~20% of layers (heuristic)
    base = None
    for lyr in model.layers:
        if isinstance(lyr, keras.Model) and lyr.name not in ['sequential', 'model']:
            base = lyr
    if base is None:
        # try find include_top=False backbone by type
        for lyr in model.layers:
            if isinstance(lyr, keras.layers.Layer) and hasattr(lyr, 'trainable') and len(lyr.weights) > 0:
                base = lyr

    if base is not None:
        total = len(base.layers) if hasattr(base, 'layers') else 0
        unfreeze_from = int(total * 0.8)
        if hasattr(base, 'layers') and total > 0:
            for i, lyr in enumerate(base.layers):
                lyr.trainable = (i >= unfreeze_from)

        compile_model(model, lr=1e-5)
        print(f"\nFine-tuning {model.name} from layer {unfreeze_from}/{total}")
        ft_history = model.fit(
            train_ds_aug,
            validation_data=val_ds_prep,
            epochs=max(8, EPOCHS//2),
            class_weight=class_weight,
            callbacks=[
                keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=4, restore_best_weights=True)
            ],
            verbose=2
        )
```

รูปภาพ 2.5.17 แสดงถึงโค้ดในส่วนของการ fine-tuning

หลังจากฝึกโมเดลด้วย backbone แบบ freeze เรียบร้อยแล้ว ในช่วงที่สองมีการปรับจูนโมเดลเพิ่มเติมโดยเปิดให้บางส่วนของโมเดลฐาน (backbone) สามารถเรียนรู้จากข้อมูลชุดใหม่ได้ เทคนิคนี้เรียกว่า **Fine-tuning** ซึ่งช่วยให้โมเดลสามารถปรับตัวกับลักษณะเฉพาะของข้อมูลปัญหาจริงได้ดียิ่งขึ้น โดยยังคงรักษาคุณลักษณะที่เรียนรู้จาก ImageNet ไว้เป็นฐาน โดยเปิดให้บางส่วนของ backbone สามารถเรียนรู้เพิ่มเติมจากข้อมูลใหม่ได้ โค้ดจะค้นหาโมเดลฐาน (backbone) จากชั้นภายในโมเดลหลัก และทำการ **unfreeze ประมาณ 20% ของชั้นสุดท้าย**

```python
total = len(base.layers) if hasattr(base, 'layers') else 0
unfreeze_from = int(total * 0.8)
if hasattr(base, 'layers') and total > 0:
    for i, lyr in enumerate(base.layers):
        lyr.trainable = (i >= unfreeze_from)
```

รูปภาพ 2.5.18 แสดงถึงโค้ดในส่วนของการ unfreeze

การเลือกเปิดเฉพาะชั้นบนสุดของ backbone มีจุดประสงค์เพื่อให้โมเดลสามารถปรับฟีเจอร์ระดับสูงให้เข้ากับลักษณะเฉพาะของข้อมูลจำแนกใบโหระพา–กะเพรา โดยไม่รบกวนโครงสร้างการเรียนรู้พื้นฐานซึ่งได้มาจาก ImageNet

หลังจาก unfreeze แล้ว โมเดลจะถูก compile ใหม่ด้วยอัตราการเรียนรู้ที่ต่ำมาก คือ

```python
def compile_model(model, lr=1e-3):
    opt = keras.optimizers.Adam(learning_rate=lr)
    model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```python
compile_model(model, lr=1e-5)
```

รูปภาพ 2.5.18 แสดงถึงโค้ดในส่วนของการ complie model

ค่า learning rate ขนาดเล็กระดับ 1×10^{-5} ช่วยป้องกันไม่ให้การอัปเดตน้ำหนักทำลายข้อมูลฟีเจอร์ที่ backbone เคยเรียนรู้ไว้ ทำให้การปรับจูนเกิดขึ้นอย่างค่อยเป็นค่อยไปและมีความเสถียร

```python
ft_history = model.fit(
    train_ds_aug,
    validation_data=val_ds_prep,
    epochs=max(8, EPOCHS//2),
    class_weight=class_weight,
    callbacks=[
        keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=4, restore_best_weights=True)
    ],
    verbose=2
)
```

รูปภาพ 2.5.19 แสดงถึงโค้ดในส่วนของการ Train mode ของช่วง fine-tune

- การฝึกในช่วง Fine-tuning ถูกกำหนดให้ใช้จำนวน epoch อย่างน้อย **8 epochs** หรือ **ครึ่งหนึ่งของจำนวน epoch เดิม** (แล้วแต่จำนวนที่มากกว่า)
- เพื่อควบคุมไม่ให้โมเดลเกิดการ overfitting มากเกินไป มีการใช้ **EarlyStopping** โดยเฝ้าดูค่า *val_accuracy* หากไม่ดีขึ้นภายใน 4 epoch ติดต่อกัน การฝึกจะหยุดลงและโหลดน้ำหนักที่ดีที่สุดกลับมาอัตโนมัติ

บทที่ 3 ผลการทดลอง

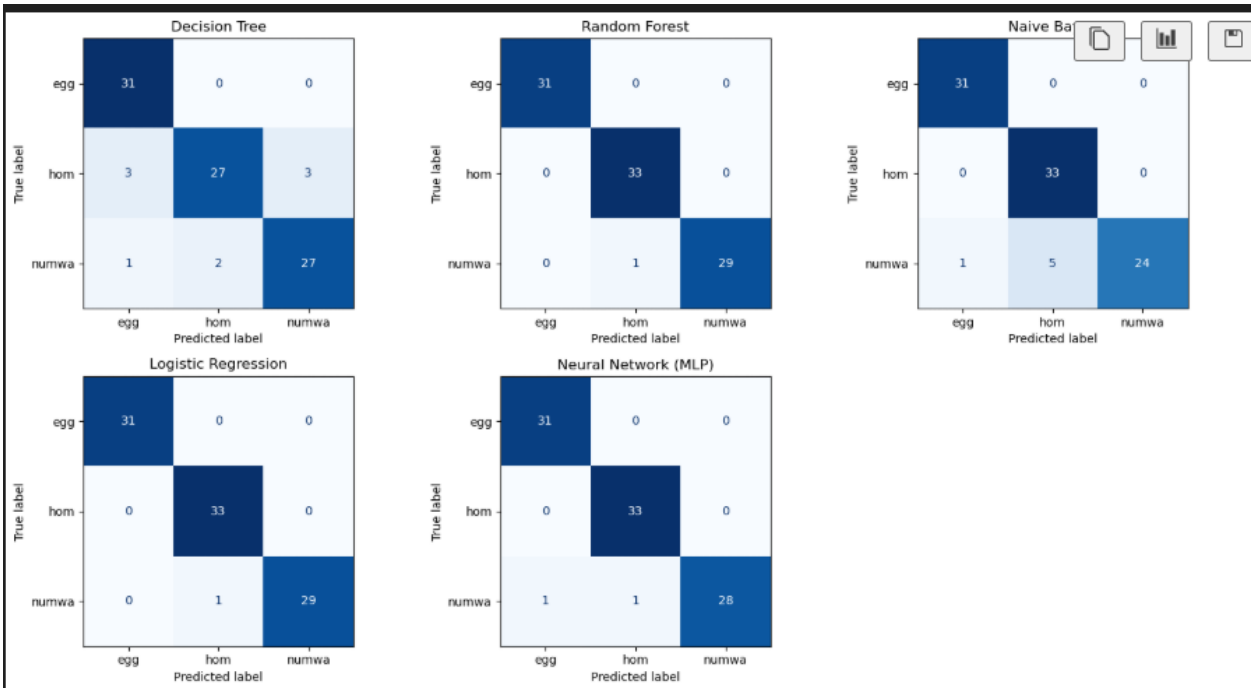
3.1 ผลการจำแนกสายพันธุ์กล้วยด้วย Machine Learning

โมเดลทั้ง 5 รูปแบบที่พัฒนาในงาน (Decision Tree, Random Forest, Naïve Bayes, Logistic Regression และ MLP) ได้รับการประเมินบนชุดข้อมูลทดสอบของภาพกล้วยสามสายพันธุ์ ซึ่งในที่นี้ผลการวัดประสิทธิภาพหลักคือ ความแม่นยำ (Accuracy) ที่คำนวณจากเปอร์เซ็นต์ของจำนวนภาพที่จำแนกสายพันธุ์ถูกต้องจากทั้งหมด

ตารางที่ 3.1 ผลการจำแนกสายพันธุ์กล้วยด้วยฟีเจอร์ 1778 มิติ (ค่าเฉลี่ยจาก Stratified K-Fold, k=5)

วิธีการ	ความแม่นยำบน validation (%)	ความแม่นยำฝึก (%)	train_loss_mean	val_loss_mean
Decision Tree	87.39% ± 2.49	100.00%	2.22e-16	4.5452
Random Forest	97.65% ± 0.81	100.00%	0.0860	0.2116
Naïve Bayes	93.80% ± 2.29	97.33%	0.9037	2.0270
Logistic Regression	99.14% ± 0.80	100.00%	0.1033	0.1246
Neural Network (MLP)	96.59% ± 2.17	99.84%	0.0090	0.3444

จากตารางที่ 3.1 จะเห็นว่าโมเดลที่ให้ความแม่นยำสูงที่สุดคือ Logistic Regression (99.14% ± 0.80) รองลงมาคือ Random Forest (97.65% ± 0.81) และ Neural Network (MLP) (96.59% ± 2.17) ซึ่งสะท้อนว่าโมเดลทั้งสามสามารถจัดการกับฟีเจอร์จำนวนมาก (1778 มิติ) ได้อย่างมีประสิทธิภาพ โดยเฉพาะ Logistic Regression ที่ทำผลงานได้ดีที่สุดแม้เป็นโมเดลเชิงเส้น เนื่องจากฟีเจอร์ที่สกัดมีความจำเพาะและแยกสายพันธุ์ได้ดีอยู่แล้ว ส่วน Naïve Bayes (93.80% ± 2.29) แม้ให้ผลดีในระดับหนึ่งแต่ยังต่ำกว่าเพราะข้อจำกัดด้านสมมติฐานความเป็นอิสระของฟีเจอร์ ขณะที่ Decision Tree มีความแม่นยำต่ำสุด (87.39% ± 2.49) และแสดงอาการ overfitting อย่างชัดเจน โดย train accuracy สูงถึง 100% แต่ validation accuracy ต่ำกว่ามาก ซึ่งสอดคล้องกับลักษณะของโมเดลต้นไม้ที่มักจำข้อมูลฝึกมากเกินไปหากไม่ได้ควบคุมความลึกอย่างเพียงพอ



ผลจาก Confusion Matrix (กล้วย 3 สายพันธุ์ – จากการทดลองจริง)
โมเดลทั้งหมดให้ผลสอดคล้องกับค่าความแม่นยำ โดยมีจุดเด่นดังนี้:

Decision Tree มักสับสน กล้วยหอม มากที่สุด ซึ่งตรงกับการเกิด overfitting

Random Forest ให้การจำแนกที่นิ่งกว่า ลดข้อผิดพลาดในคู่ที่ใกล้เคียงกัน

Naïve Bayes มีแนวโน้มทำนายผิดในคลาสที่มี distribution ทับซ้อนสูง เช่น ลักษณะพื้นผิวใกล้กัน

Logistic Regression จำแนกทั้งสามสายพันธุ์ได้เกือบสมบูรณ์ โดยเฉพาะ กล้วยหอม ที่เด่นชัดที่สุด

MLP ให้ผลดีมากแต่ยังมีจุดที่สับสนเล็กน้อยในกล้วยน้ำว้ากับกล้วยไข่

โดยรวม Confusion Matrix ชี้ว่า คู่ที่สับสนมากที่สุดคือ น้ำว้า–หอม ขณะที่ กล้วยหอม ถูกจำแนกถูกต้องสูงที่สุดในทุกโมเดล

3.2 ผลการจำแนกใบกะเพราและโหระพาด้วยโมเดล CNN

ในการทดลองส่วนที่สอง เราได้ฝึกโมเดลโครงข่าย CNN ที่มีสถาปัตยกรรมแตกต่างกัน 5 แบบโดยใช้วิธี transfer learning ผลลัพธ์ค่าความแม่นยำ (Accuracy) และความสูญเสีย (Loss) บนชุด validation ของแต่ละโมเดลสรุปแสดงในตารางที่ 3.2

ตารางที่ 3.2 ประสิทธิภาพของโมเดล CNN แต่ละสถาปัตยกรรมในการจำแนกใบกะเพรา-โหระพา (ชุดข้อมูล validation)

โมเดล CNN	ความแม่นยำ (%)	Loss (ค่าความสูญเสีย)
MobileNetV2	92.50	0.3125
EfficientNetB0	95.00	0.1499
NASNetMobile	92.50	0.2868
ResNet50V2	92.50	0.3098
Xception	90.00	0.2417

ผล Fine-tuning ของโมเดลแต่ละตัว

โมเดล CNN	val_accuracy	val_loss
MobileNetV2	0.8250	0.5801
EfficientNetB0	1.0000	0.0988
NASNetMobile	0.8750	0.3099
ResNet50V2	0.9250	0.3811
Xception	0.9000	0.2313

สรุปผลจากตาราง Fine-tuning

จากผล Fine-tuning จะเห็นว่ามีเพียง EfficientNetB0 ที่ประสิทธิภาพดีขึ้นอย่างชัดเจน ความแม่นยำเพิ่มจาก 95% เป็น 100% และ loss ลดลงเหลือเพียง ~0.0988 แสดงว่าโมเดลตอบสนองต่อการปรับพารามิเตอร์ระดับลึกได้ดีแม้ข้อมูลจะมีจำนวนจำกัด

โมเดลอื่น ๆ ไม่ได้มีแนวโน้มดีขึ้น โดยเฉพาะ MobileNetV2 และ NASNetMobile ที่ความแม่นยำลดลงและ loss สูงขึ้นหลัง fine-tune ซึ่งเป็นสัญญาณ overfitting จากข้อมูลที่มีจำนวนค่อนข้างน้อย ขณะที่ ResNet50V2 และ Xception แม้ความแม่นยำจะไม่ลดลงมาก แต่ไม่ได้ดีขึ้นเช่นกัน

ตารางเปรียบเทียบ Frozen vs Fine-tuned (ภาพรวม)

โมเดล CNN	Frozen Accuracy	Frozen Loss	Fine-tuned Accuracy	Fine-tuned Loss	แนวโน้ม
MobileNetV2	0.9250	0.3125	0.8250	0.5801	แย่ลง (overfitting)
EfficientNetB0	0.9500	0.1499	1.0000	0.0988	ดีขึ้นชัดเจน
NASNetMobile	0.9250	0.2868	0.8750	0.3099	แย่ลง
ResNet50V2	0.9250	0.3098	0.9250	0.3811	ไม่ดีขึ้น (loss แย่ลง)
Xception	0.9000	0.2417	0.9000	0.2313	เท่าเดิม (loss ดีขึ้นเล็กน้อย)

สรุปเทียบผลรวม

EfficientNetB0 เป็นเพียงโมเดลเดียวที่ fine-tuning แล้วดีขึ้นจริง

MobileNetV2 และ NASNetMobile แย่ลง อย่างเห็นได้ชัด เนื่องจาก overfitting

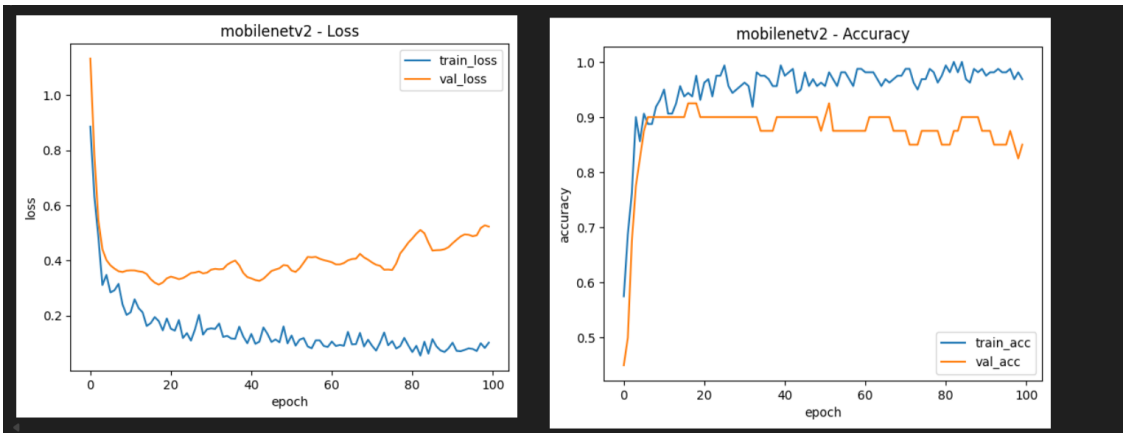
Xception และ ResNet50V2 แทบไม่เปลี่ยนแปลง และไม่สามารถเพิ่ม accuracy ได้จากข้อมูลชุดนี้

โดยภาพรวม แสดงให้เห็นว่าการ fine-tuning จะได้ผลก็ต่อเมื่อโมเดลมีความสมดุลระหว่างขนาดโมเดล–จำนวนพารามิเตอร์–และจำนวนข้อมูลที่มีเพียงพอ เช่น EfficientNetB0 ส่วนโมเดลที่ซับซ้อนเกินไปในบริบทของข้อมูลขนาดเล็กจะมีแนวโน้ม overfit เมื่อทำ fine-tuning | 0.8750 | 0.3099 |

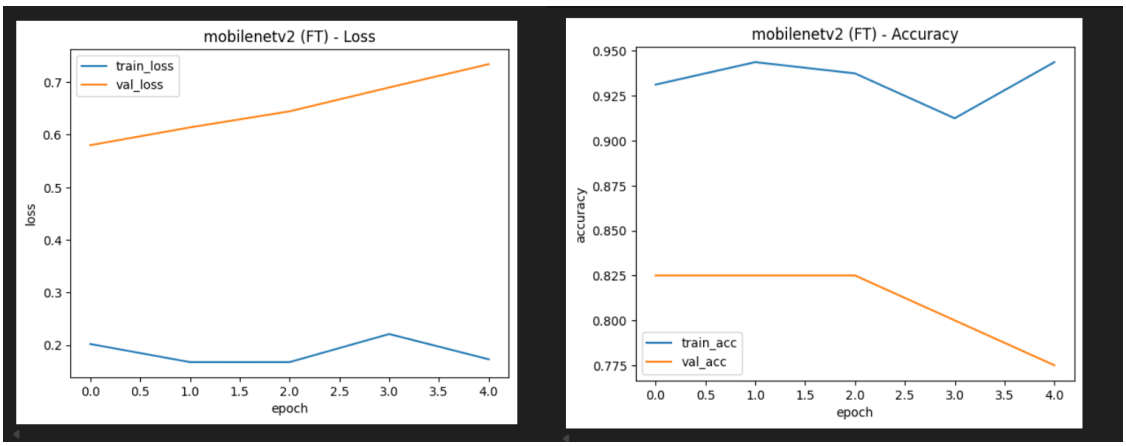
โมเดล MobileNetV2, NASNetMobile และ ResNet50V2 ให้ผลใกล้เคียงกัน โดยมีความแม่นยำสูงถึง 92.50% แต่ค่า Loss ต่างกันเล็กน้อย ซึ่งบ่งชี้ว่าแต่ละสถาปัตยกรรมมีลักษณะการเรียนรู้ที่ต่างกัน โดย NASNetMobile มีค่า val_loss ต่ำที่สุดในกลุ่มรองลงมา แสดงถึงการทำนายที่ค่อนข้างมั่นคง ขณะที่ ResNet50V2 และ MobileNetV2 ให้ความแม่นยำเท่ากันแต่ Loss สูงกว่าเล็กน้อย ส่วนโมเดล Xception แม้มีสถาปัตยกรรมที่ซับซ้อนกว่าบางโมเดล แต่ให้ความแม่นยำต่ำที่สุดในชุดนี้ที่ 90.00% ซึ่งอาจเกิดจากจำนวนข้อมูลที่มีจำกัด ทำให้โมเดลใหญ่ไม่สามารถเรียนรู้ได้เต็มที่และมีแนวโน้มเกิด overfitting ง่ายกว่า

โดยรวมแล้ว ผลลัพธ์ชี้ให้เห็นว่า โมเดลขนาดเล็กถึงกลางที่มีประสิทธิภาพสูงและผ่านการออกแบบอย่างสมดุล เช่น EfficientNetB0 สามารถทำงานได้ดีกว่าโมเดลที่มีความซับซ้อนสูงกว่าในชุดข้อมูลขนาดเล็ก นอกจากนี้ ค่าความสูญเสียที่ลดลงตามความแม่นยำที่สูงขึ้นบ่งบอกว่าโมเดลไม่ได้เพียงทำนายถูกต้องเท่านั้น แต่ยังทำนายด้วยความมั่นใจมากกว่าอีกด้วย

ในการวิเคราะห์เชิงลึกเพิ่มเติม เราได้พิจารณาพฤติกรรมการเรียนรู้ของโมเดล MobileNetV2 ซึ่งเป็นหนึ่งในโมเดลที่ทดลอง โดยเปรียบเทียบระหว่างกรณี ฝึกเฉพาะชั้นบนสุด (FROZEN BASE) กับกรณี ปรับค่าพารามิเตอร์แบบ fine-tuning รูปที่ 3.1 แสดงกราฟการเรียนรู้ของ MobileNetV2 เมื่อฝึกด้วยFROZEN BASE (Frozen)
เปรียบเทียบความสูญเสียและความแม่นยำระหว่างชุดฝึกและชุด validation ตลอด 100 epochs ในขณะที่รูปที่ 3.2 แสดงกราฟในกรณีที่ทำ fine-tuning โมเดล (ซึ่งฝึกต่อเนื่องเพียงไม่กี่ epochs ในที่นี้)



รูปที่ 3.1 กราฟแสดงค่าความสูญเสีย (loss) และความแม่นยำ (accuracy) ของการฝึกโมเดล MobileNetV2 โดยแช่แข็งฐานโมเดล ไม่ปรับน้ำหนักของ convolutional layers จาก ImageNet (แกน x คือ epoch ตั้งแต่ 0 ถึง 100) จะเห็นว่าความสูญเสียชุดฝึก (train_loss – เส้นสีน้ำเงิน) ลดลงอย่างรวดเร็วในช่วงแรกและทรงตัวที่ค่าต่ำ (~0.2 หรือต่ำกว่า) ขณะที่ความสูญเสียชุด validation (val_loss – เส้นสีส้ม) ลดลงช้ากว่าและแกว่งอยู่ราว 0.3–0.5 ในช่วงกลางก่อนจะขยับขึ้นเล็กน้อยในช่วงท้าย ส่วนกราฟความแม่นยำแสดงให้เห็นว่าโมเดลมีความแม่นยำชุดฝึก (train_acc) เพิ่มขึ้นถึงประมาณ 1.0 (100%) อย่างรวดเร็วและแกว่งอยู่ใกล้ค่านั้น ในขณะที่ความแม่นยำชุด validation (val_acc – เส้นสีส้มกราฟขวา) เริ่มต้นที่ประมาณ 0.5 และไต่ขึ้นมาที่ ~0.85–0.90 ภายใน ~10 epochs แต่อาจตกลงบ้างเล็กน้อยและแกว่งอยู่ในช่วง ~0.80–0.90 ตลอดการฝึก

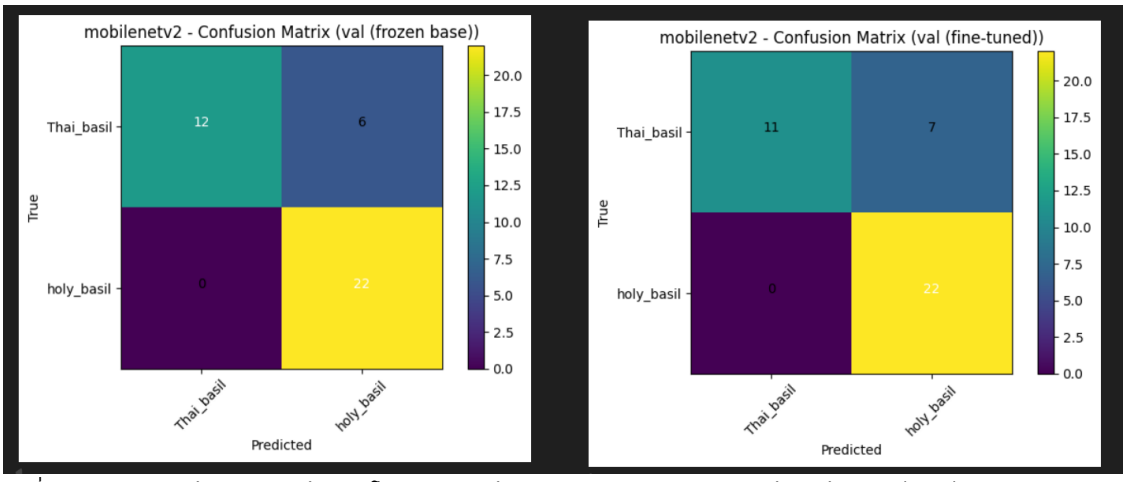


รูปที่ 3.2 กราฟแสดงค่าความสูญเสียและความแม่นยำของโมเดล MobileNetV2 กรณีทำ Fine-tuning (ปรับค่า weight ในบางชั้นของ convolutional base) ที่ดำเนินการฝึกต่ออีก 5 epochs หลังจากฐานแช่แข็ง โมเดลมีความแม่นยำชุดฝึกสูงขึ้นเล็กน้อย (~0.95 หรือ 95%) แต่ความแม่นยำชุด validation (เส้นสีส้ม) กลับลดต่ำลงจาก ~0.82 ในตอนเริ่มเหลือ ~0.78 เมื่อสิ้นสุด 5 epochs และค่าความสูญเสียบน validation เพิ่มขึ้น (จาก ~0.6 เป็น ~0.75) สะท้อนปัญหา overfitting ที่เกิดขึ้นเมื่อทำการ fine-tune โมเดลด้วยข้อมูลจำนวนน้อยเกินไป

จากกราฟรูปที่ 3.1 และ 3.2 จะเห็นแนวโน้มว่าเมื่อใช้ฐานโมเดลที่ฝึกจาก ImageNet โดยไม่ปรับน้ำหนัก (รูปที่ 3.1) โมเดลสามารถเรียนรู้ชั้นบนสุดให้เข้ากับปัญหาใหม่ได้ดีพอสมควร ความแม่นยำบน validation สูงถึง ~85–90% และ loss อยู่ระดับต่ำ โดยโมเดลไม่ได้แสดงอาการ overfit ชัดเจนมากนัก (val_acc และ train_acc ใกล้เคียงกันช่วงท้าย ๆ) ในทางกลับกัน เมื่ออนุญาตให้ปรับน้ำหนักระหว่างการ fine-tuning (รูปที่ 3.2) เราพบว่าความแม่นยำบนชุดฝึกเพิ่มสูงขึ้นอย่างรวดเร็วเกือบแตะ 95–100% ในไม่กี่ epoch แต่ความแม่นยำบน validation กลับลดต่ำลงและค่าความสูญเสียสูงขึ้น

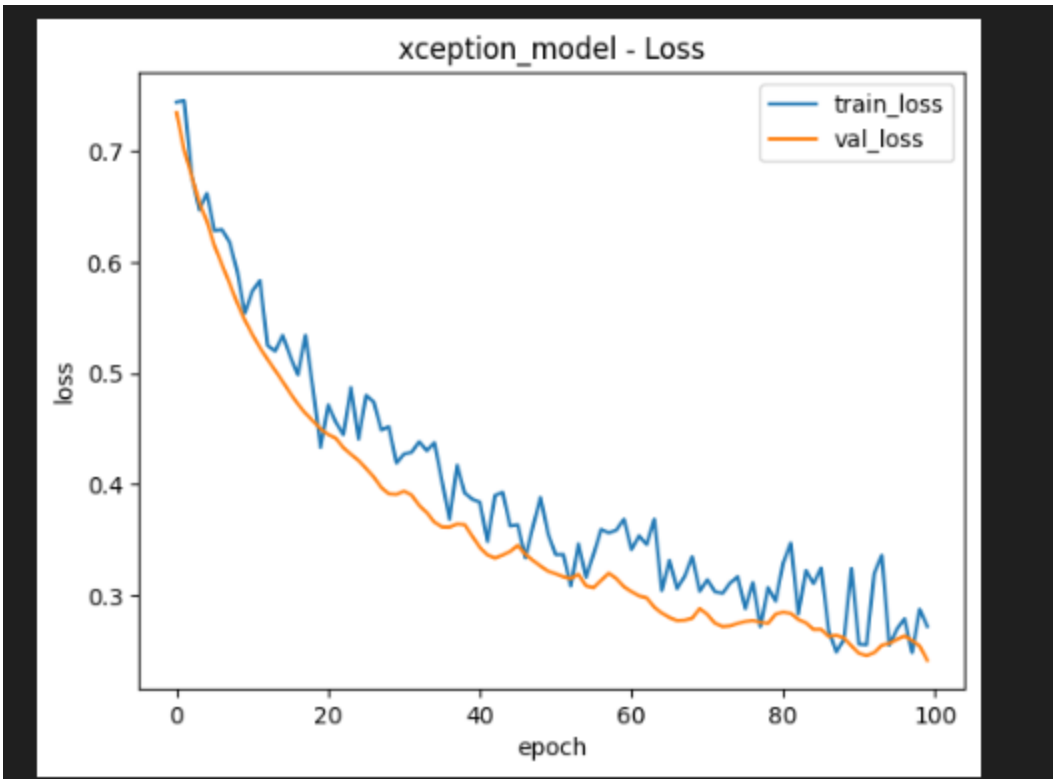
แสดงว่าโมเดลเริ่มจดจำรายละเอียดของชุดฝึกมากเกินไปจนสูญเสียความสามารถในการทั่วไปกับข้อมูลใหม่ – ซึ่งเป็นลักษณะของการฝึกมากเกินไป (Overfitting) นั่นเอง

เพื่อทำความเข้าใจเชิงคุณภาพเกี่ยวกับข้อผิดพลาดของโมเดล เราได้สร้าง Confusion Matrix สำหรับโมเดล MobileNetV2 ทั้งในกรณีฐานแช่แข็งและกรณี fine-tuned ดังแสดงในรูปที่ 3.3 โดยแกนแนวตั้งคือประเภทจริงของภาพ (True label) และแกนแนวนอนคือประเภทที่โมเดลทำนาย (Predicted label) ภายในตารางมีตัวเลขจำนวนภาพในแต่ละกรณี (โมเดลทำนายเป็น X ในขณะที่จริง ๆ เป็น Y)

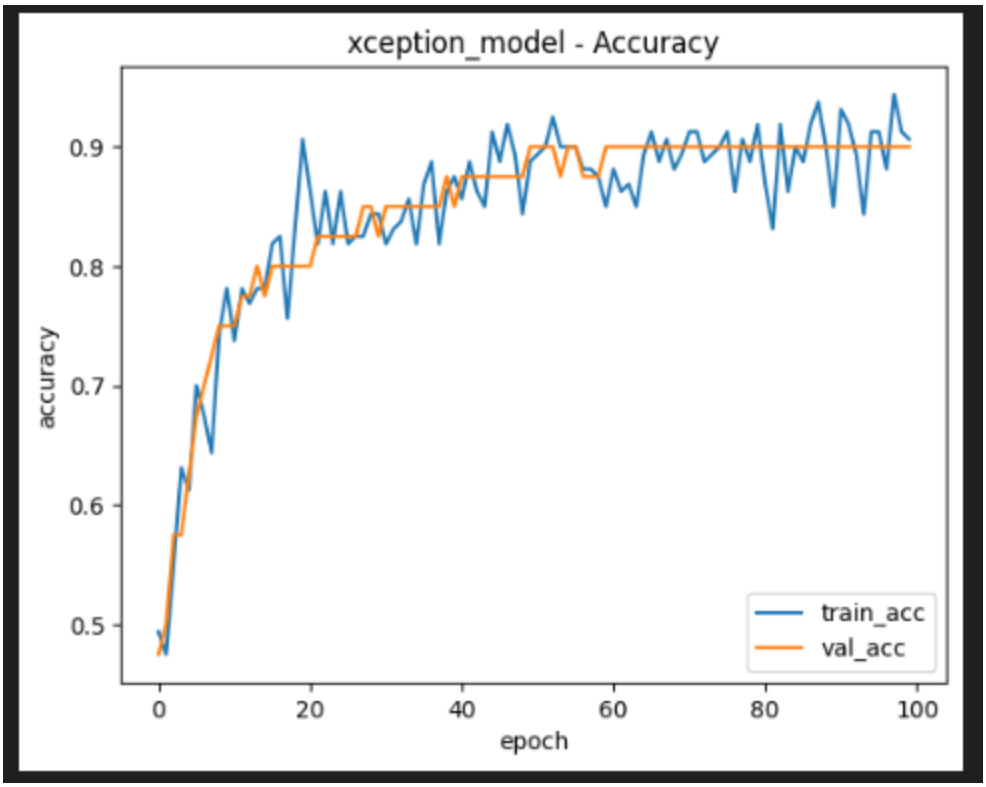


รูปที่ 3.3 confusion Matrix ของโมเดล MobileNetV2 บนชุดข้อมูล validation เปรียบเทียบระหว่าง (ซ้าย) FROZEN BASE และ (ขวา) Fine-tuned จะเห็นว่าในทั้งสองกรณี โมเดลสามารถจำแนกภาพใบholy_basil (กะเพรา) ได้ถูกต้องทั้งหมด (ทำนายเป็น holy_basil ถูกต้อง 22 ภาพ จากทั้งหมด 22 ภาพ ไม่มีข้อผิดพลาด) ในขณะที่การจำแนกใบ Thai_basil (โหระพา) มีความสับสนเกิดขึ้น โดยกรณีฐานแช่แข็ง (รูปซ้าย) จำแนกใบโหระพาถูกต้อง 12 ภาพ และผิดเป็นกะเพรา 6 ภาพ ในขณะที่กรณี fine-tuned (รูปขวา) จำแนกโหระพาถูกต้อง 11 ภาพ และผิดเป็นกะเพรา 7 ภาพ จะเห็นว่าโมเดลที่ fine-tune แล้วไม่ได้ช่วยเพิ่มความแม่นยำสำหรับคลาสโหระพา แต่มิหนำซ้ำยังทำให้จำนวนที่ทำนายผิดเพิ่มขึ้นเล็กน้อย (จาก 6 เป็น 7 ภาพ)

ต่อจากการวิเคราะห์โมเดล MobileNetV2 เราได้พิจารณา โมเดล Xception ซึ่งเป็นสถาปัตยกรรมที่ลึกกว่าและใช้ depthwise separable convolution เช่นเดียวกัน แต่มีโครงสร้างบล็อกที่ซับซ้อนกว่า รูปที่ 3.4 และ 3.5 แสดงกราฟการเรียนรู้ของ Xception ในกรณีFrozen base โดยรูปที่ 3.4 เป็นกราฟ loss (train_loss และ val_loss) และรูปที่ 3.5 เป็นกราฟ accuracy (train_acc และ val_acc) ตลอด 100 epochs

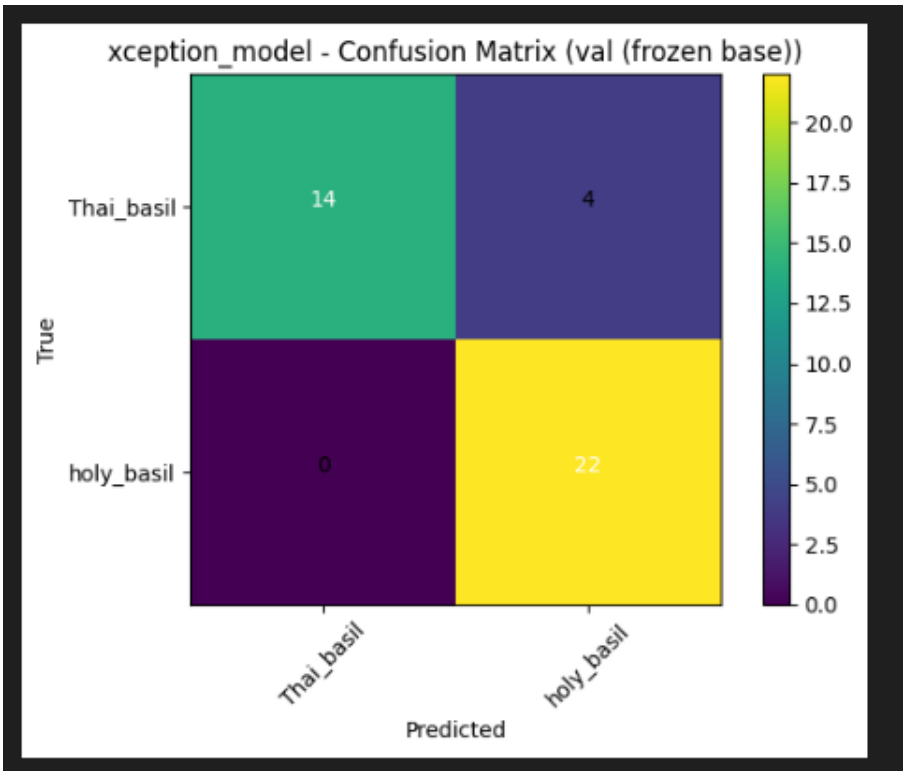


รูปที่ 3.4 กราฟ loss ของโมเดล Xception (Frozen base) – เส้น train_loss (สีน้ำเงิน) และ val_loss (สีส้ม) ลดลงต่อเนื่องตั้งแต่ต้นจนจบการฝึก โดย val_loss จะค่อย ๆ ลดลงอย่างราบเรียบจากราว 0.7 ลงมาใกล้ 0.25 ในช่วงท้าย และมักอยู่ต่ำกว่า train_loss เล็กน้อย แสดงให้เห็นว่าโมเดลไม่ได้จดจำข้อมูลฝึกมากเกินไป แต่มีแนวโน้ม generalize ได้ดีต่อชุด validation ความแตกต่างระหว่าง train_loss และ val_loss ไม่กว้างมากและไม่มีช่วงที่ val_loss ดีดขึ้นอย่างรุนแรง จึงไม่ปรากฏสัญญาณ overfitting ชัดเจนในกรณีนี้



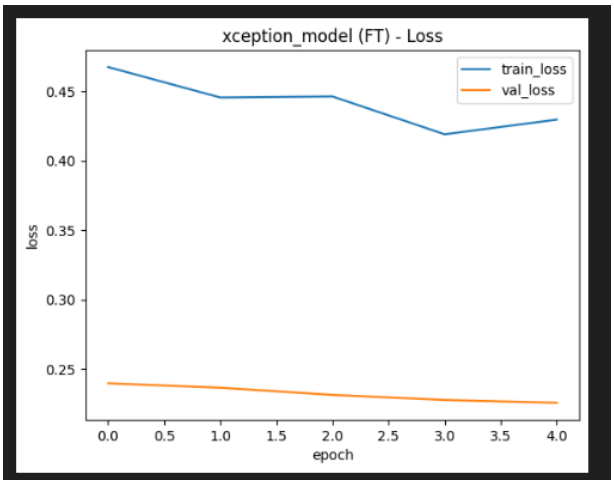
รูปที่ 3.5 กราฟ accuracy ของโมเดล Xception (Frozen base) – train_acc (สีน้ำเงิน) และ val_acc (สีส้ม) เพิ่มขึ้นอย่างรวดเร็วในช่วง 10–15 epochs แรก จากประมาณ 0.48 ขึ้นไปใกล้ 0.8 และค่อย ๆ ไต่ขึ้นจนแตะระดับประมาณ 0.90 ภายในช่วงกลางของการฝึก หลังจากนั้นค่า val_acc ทรงตัวใกล้ 0.90 ตลอดช่วงท้าย ในขณะที่ train_acc แกว่งอยู่รอบ ๆ 0.88–0.93 แสดงให้เห็นว่าโมเดล Xception สามารถเรียนรู้ได้ดีและมีเสถียรภาพสูง train_acc ไม่หนีห่างจาก val_acc มากนัก จึงไม่บ่งชี้ถึงปัญหา overfitting รุนแรง

ในแง่เชิงคุณภาพ เราได้พิจารณา Confusion Matrix ของ Xception บนชุด validation ดังแสดงในรูปที่ 3.6 โดยในกรณีFrozen base โมเดลสามารถจำแนกใบ holy_basil ได้ถูกต้องทุกภาพ (22 จาก 22 ภาพ) เช่นเดียวกับกรณีของ MobileNetV2 ในขณะที่ใบ Thai_basil ถูกจำแนกถูกต้อง 14 ภาพ จากทั้งหมด 18 ภาพ และถูกทำนายผิดว่าเป็น holy_basil จำนวน 4 ภาพ จะเห็นว่าลักษณะความผิดพลาดคล้ายกับ MobileNetV2 คือโมเดลไม่ค่อยสับสนใบกะเพราไปเป็นโหระพา แต่มีแนวโน้มจะมองใบโหระพบางรูปที่ลักษณะคล้ายกะเพรา (เช่น ใบไม่มันหรือมีรอยหยักมาก) แล้วทำนายผิดว่าเป็น holy_basil แทน

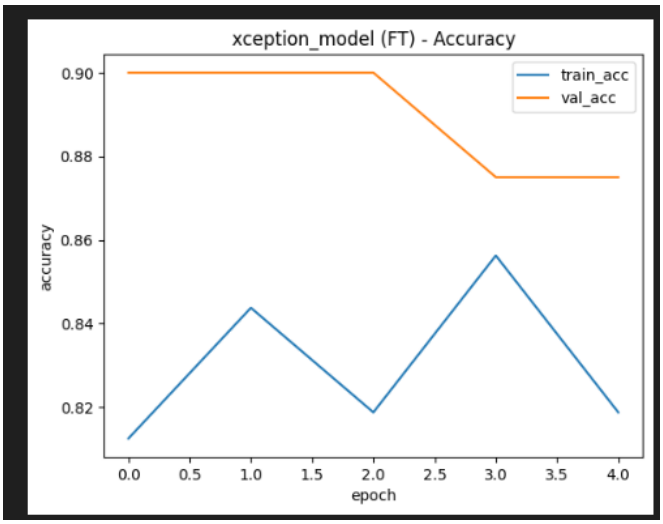


รูปที่ 3.6 Confusion Matrix ของโมเดล Xception บนชุด validation (Frozen base) – แสดงจำนวนภาพ Thai_basil ที่ทำนายถูก 14 และผิด 4 ภาพ ในขณะที่ holy_basil ถูกต้องทั้งหมด 22 ภาพ รวม accuracy โดยรวมประมาณ 90%

สำหรับกรณี fine-tuning โมเดล Xception เราได้ปลดล็อกบางชั้นของ convolutional base และฝึกต่ออีก 4 epochs โดยใช้ learning rate ที่ต่ำลงมาก ผลลัพธ์กราฟ loss และ accuracy แสดงในรูปที่ 3.7 และ 3.8 ตามลำดับ จะเห็นว่า val_loss ลดลงเล็กน้อยจากประมาณ 0.24 ลงมาบริเวณ ~0.23 ในขณะที่ train_loss ยังคงอยู่ในช่วง 0.42–0.46 ส่วน val_acc ทรงตัวใกล้ 0.90 ตลอด 4 epochs และมีการแกว่งเล็กน้อยในระดับ ±0.02 เท่านั้น นั่นคือ การ fine-tune Xception ไม่ได้ช่วยเพิ่มความแม่นยำอย่างมีนัยสำคัญ แต่ช่วยปรับปรุงค่าความสูญเสียให้ลดลงเล็กน้อย แสดงว่าฐานโมเดลเดิมที่ฝึกจาก ImageNet นั้นเหมาะสมกับงานพอสมควรอยู่แล้ว และการปรับจูนเพิ่มเติมด้วยข้อมูลจำนวนน้อยจึงให้ผลต่างเพียงเล็กน้อย

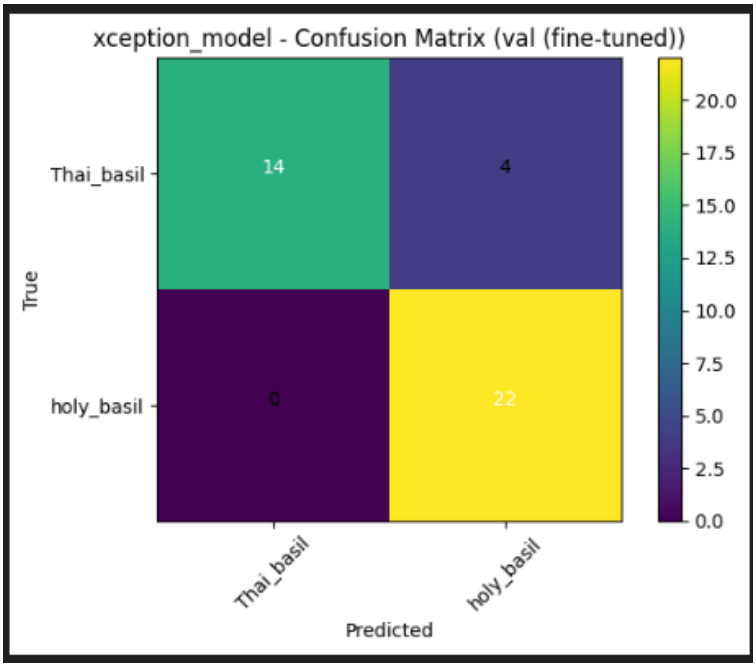


รูปที่ 3.7 กราฟ loss ของโมเดล Xception กรณี fine-tuning (ฝึกต่อ 4 epochs) – train_loss มีค่าอยู่ในช่วง ~0.42–0.46 ขณะที่ val_loss ค่อย ๆ ลดลงจากประมาณ 0.24 ไปใกล้ 0.22 แสดงว่าความมั่นใจของโมเดลบนชุด validation ดีขึ้นเล็กน้อย แม้ accuracy จะแทบไม่เปลี่ยน



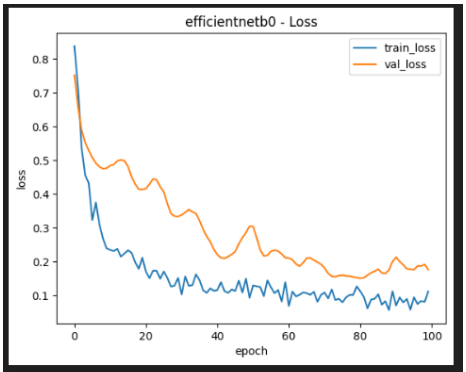
รูปที่ 3.8 กราฟ accuracy ของโมเดล Xception กรณี fine-tuning – train_acc แกว่งอยู่ราว 0.81–0.86 ในขณะที่ val_acc เริ่มต้นที่ 0.90 และลดลงเล็กน้อยมาอยู่บริเวณ 0.88–0.89 ทำให้โดยรวมแล้วการ fine-tune ไม่ได้ช่วยเพิ่ม accuracy แต่ยังคงรักษาระดับเดิมไว้ได้

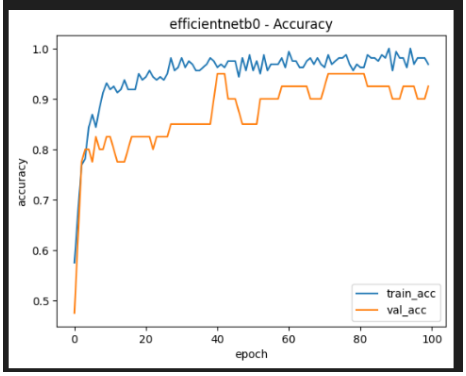
Confusion Matrix ของ Xception หลังการ fine-tune (รูปที่ 3.9) แสดงให้เห็นว่าจำนวนภาพที่จำแนกถูก/ผิดในแต่ละคลาสเหมือนเดิมทุกประการกับกรณี frozen model คือ Thai_basil ถูกต้อง 14 ผิด 4 และ holy_basil ถูกต้องทั้งหมด 22 ภาพ แสดงว่าแม้ค่า loss จะดีขึ้นเล็กน้อย แต่การ fine-tune ไม่ได้เปลี่ยนรูปแบบข้อผิดพลาดของโมเดลเลย กล่าวอีกนัยหนึ่ง โมเดล Xception ในงานนี้อยู่ในจุดที่ “เรียนรู้มากพอ” ตั้งแต่เฟสแรกแล้ว และการปรับจูนต่อด้วยข้อมูลชุดเดิมเพียง 4 epochs ไม่ได้สร้าง decision boundary ใหม่ที่แตกต่างอย่างมีนัยสำคัญ



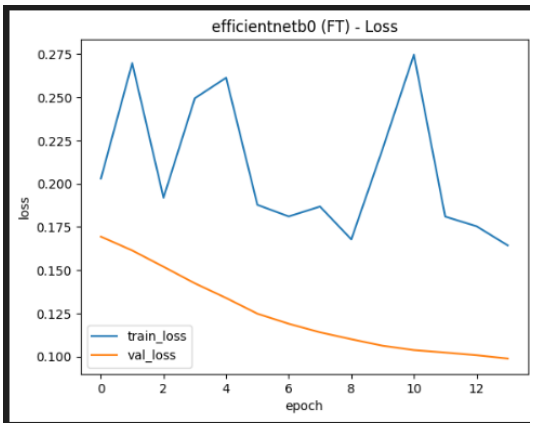
รูปที่ 3.9 Confusion matrix ของโมเดล Xception บนชุด validation หลังการ fine-tune (ปรากฏรูปแบบตัวเลขเหมือนกับ frozen ทุกประการ))

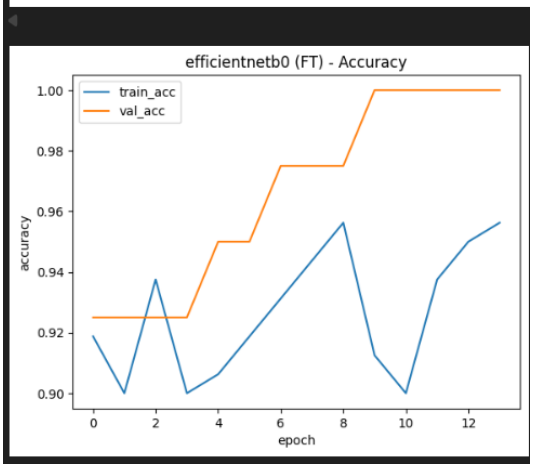
การวิเคราะห์เชิงลึกของโมเดล EfficientNetB0 โมเดล EfficientNetB0 แสดงประสิทธิภาพสูงสุดในงานนี้ ทั้งในเฟส frozen base และ fine-tuning



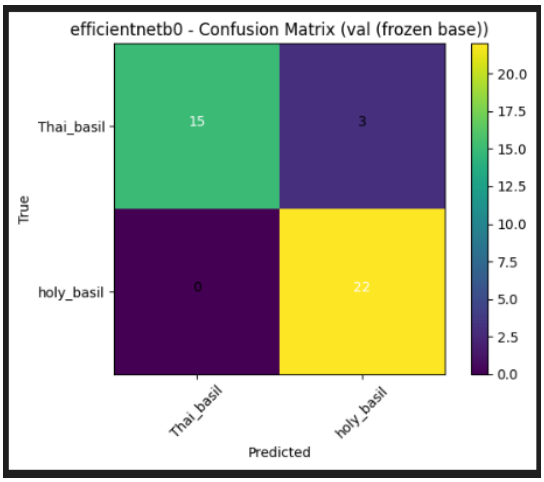


จากกราฟ loss และ accuracy ของเฟส frozen base จะเห็นว่าค่า train_loss ลดลงเร็วและอยู่ในระดับต่ำมาก (~0.07–0.12) ในขณะที่ val_loss ค่อย ๆ ลดลงอย่างสม่ำเสมอจากราว 0.55 ลงมาอยู่บริเวณ ~0.18 ช่วงท้าย โดยไม่มีอาการดีดขึ้นซึ่งเป็นสัญญาณ overfitting แสดงว่าโมเดลสามารถ generalize ได้ดี ความแม่นยำ val_acc เพิ่มขึ้นอย่างรวดเร็วแตะระดับ ~0.90 ภายใน ~10 epochs และทรงตัวอยู่ใกล้ระดับนั้นตลอดการฝึก

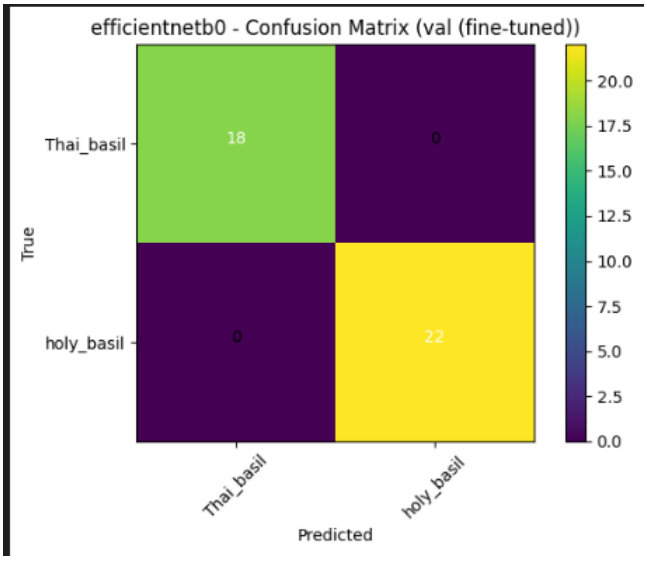




ในส่วน fine-tuning ค่า val_loss ยังคงลดต่อเนื่องจาก ~0.18 ลงไปใกล้ ~0.10 และ val_acc กระโดดจาก ~0.95 ในเฟสก่อนหน้าไปแตะ 1.00 ในช่วงท้ายของการ fine-tuning ขณะที่ train_loss แกว่งอยู่ในช่วง ~0.20–0.27 ซึ่งยังถือว่าต่ำและไม่แสดงสัญญาณ overfitting รุนแรง ผลนี้ชี้ให้เห็นว่า EfficientNetB0 ตอบสนองต่อการ fine-tuning ได้ดีกว่าโมเดลอื่นในงานนี้ และได้ประโยชน์จากการ unfreeze ชั้นบน ๆ ของ backbone อย่างสมดุล



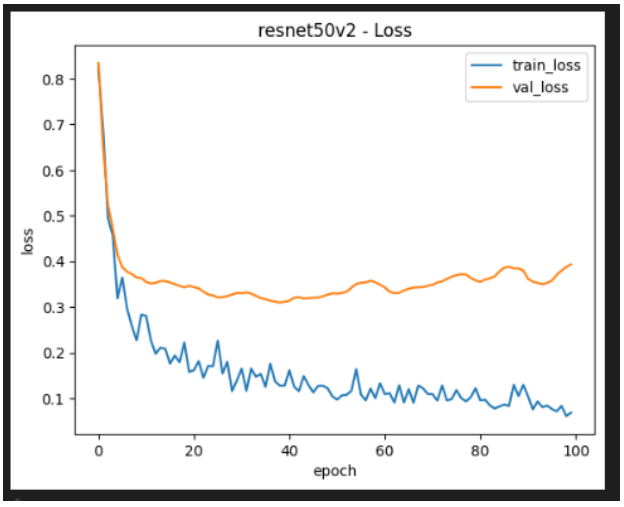
สำหรับ confusion matrix ของ EfficientNetB0 ก่อนการ fine-tune โมเดลจำแนก Thai_basil ถูกต้อง 15 จาก 18 ภาพ และจำแนก holy_basil ถูกต้องทั้งหมด 22 ภาพ



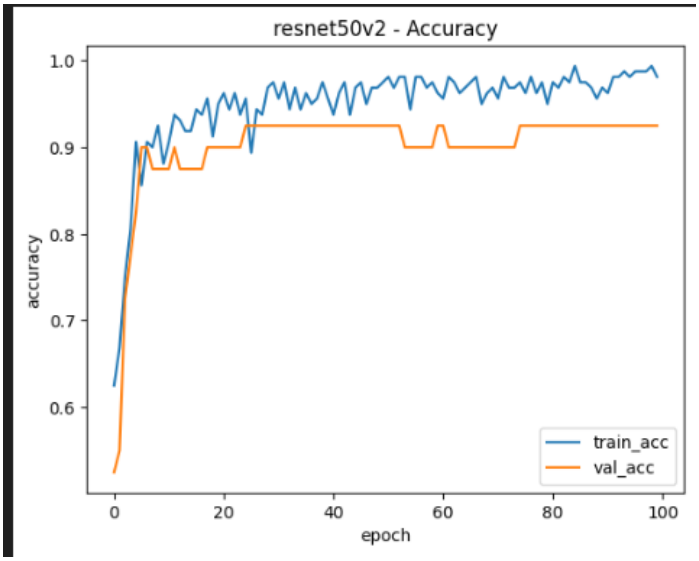
หลังการ fine-tune โมเดลสามารถจำแนกภาพได้ถูกต้อง ทุกภาพ (18/18 และ 22/22) ซึ่งสอดคล้องกับ val_acc = 1.00 ในเฟส fine-tuning แสดงให้เห็นว่า decision boundary ของโมเดลถูกปรับให้เหมาะกับข้อมูลมากขึ้น และไม่พบการทำนายผิดของคลาสใดเลย

โดยสรุป EfficientNetB0 ไม่เพียงให้ความแม่นยำสูงที่สุด แต่ยังมีเสถียรภาพทั้งระหว่าง train/validation และยังได้ประโยชน์ชัดเจนจากการ fine-tune ซึ่งต่างจาก Xception หรือ MobileNetV2 ที่ไม่ได้ดีขึ้นหลังปรับจูน ทำให้ EfficientNetB0 เป็นโมเดลที่เหมาะสมที่สุดสำหรับงานจำแนกใบกะเพรา–โหระพาในชุดข้อมูลนี้

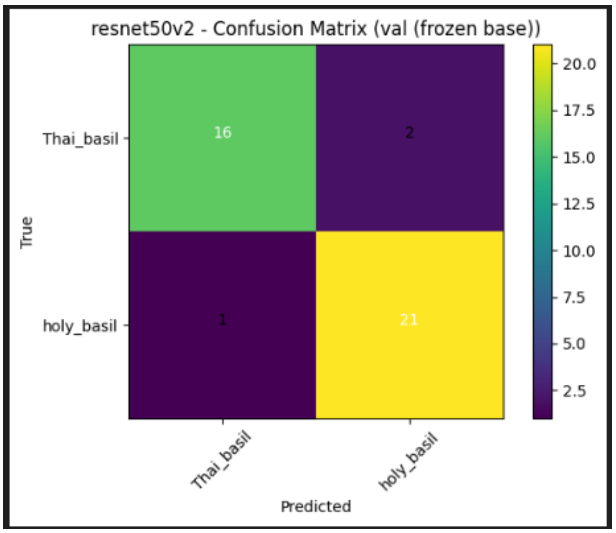
การวิเคราะห์เชิงลึกของโมเดล ResNet50V2 ResNet50V2 เป็นโมเดลที่ใช้ skip connections ช่วยทำให้การไหลของ gradient มีเสถียรภาพมากขึ้นและป้องกันปัญหา vanishing gradient ในโมเดลลึก



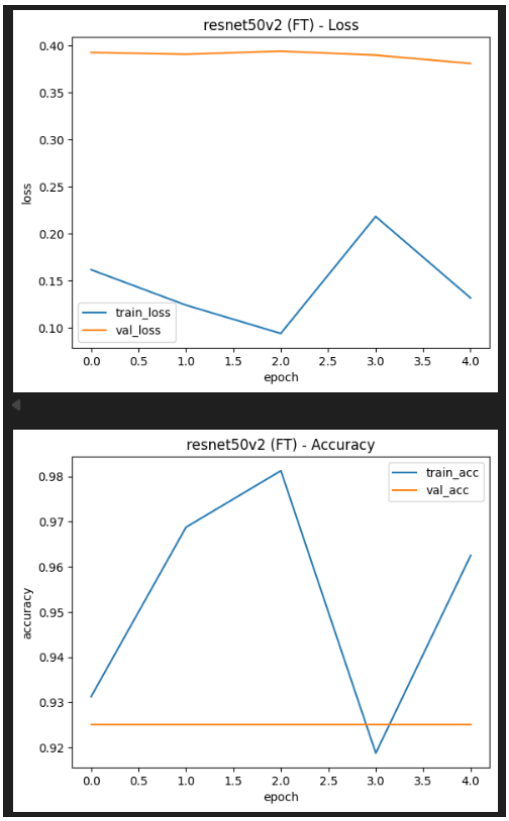
จากกราฟ loss ของเฟส frozen base จะเห็นว่า train_loss ลดลงต่อเนื่องจนอยู่ระดับต่ำมาก (~0.07–0.12) และมีความนิ่งในช่วงท้าย ในขณะที่ val_loss ไม่ลดลงมากเท่าโมเดล EfficientNetB0 แต่ยังคงอยู่ในช่วง ~0.35–0.45 ซึ่งสูงกว่า train_loss ค่อนข้างชัดเจนและมีการแกว่งตัวพอสมควร สะท้อนว่า ResNet50V2 อาจเริ่มเรียนรู้รายละเอียดยิบย่อยของข้อมูลฝึกมากขึ้น และมีแนวโน้ม overfitting เล็กน้อยเมื่อเทียบกับ EfficientNetB0 หรือ Xception ในเฟสเดียวกัน



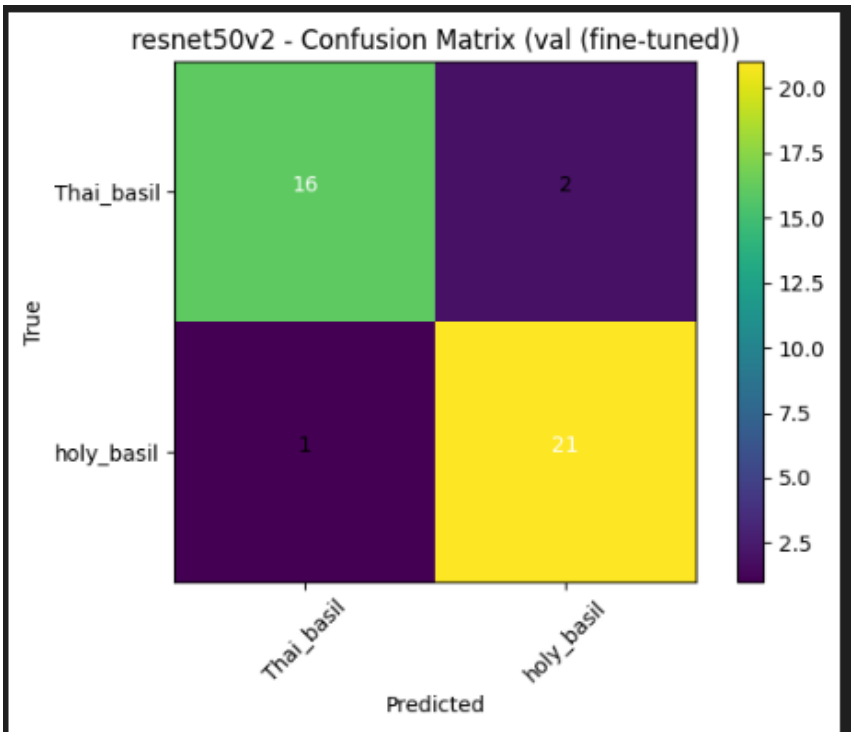
กราฟ accuracy แสดงให้เห็นว่า train_acc เพิ่มขึ้นเรื่อย ๆ จนเกือบแตะ 1.0 ในขณะที่ val_acc อยู่ประมาณ 0.92 แบบคงที่ตลอด 100 epochs โดยแทบไม่ขยับขึ้นหรือลง แสดงว่าโมเดลมีประสิทธิภาพดีและทรงตัว แต่ไม่ได้ปรับปรุงประสิทธิภาพเพิ่มขึ้นแม้ฝึกนานขึ้น ซึ่งบ่งบอกว่าการแช่แข็ง backbone ของ ResNet50V2 อาจจำกัดความสามารถในการเรียนรู้ feature ที่จำเพาะกับใบกะเพรา–โหระพา แม้ว่าโมเดลจะ generalize ได้ดีระดับหนึ่งก็ตาม



สำหรับ confusion matrix ของ ResNet50V2 โมเดลจำแนก Thai_basil ถูกต้อง 16 จาก 18 ภาพ ผิด 2 ภาพ และจำแนก holy_basil ถูกต้อง 21 จาก 22 ภาพ ผิด 1 ภาพ ด้วย accuracy รวม ~92.5% ซึ่งอยู่ระดับเดียวกับ MobileNetV2 และ NASNetMobile และใกล้เคียง Xception แต่ ResNet50V2 มีความสม่ำเสมอในคลาสทั้งสองมากกว่า ไม่ได้มีคลาสใดที่ผิดมากเป็นพิเศษ



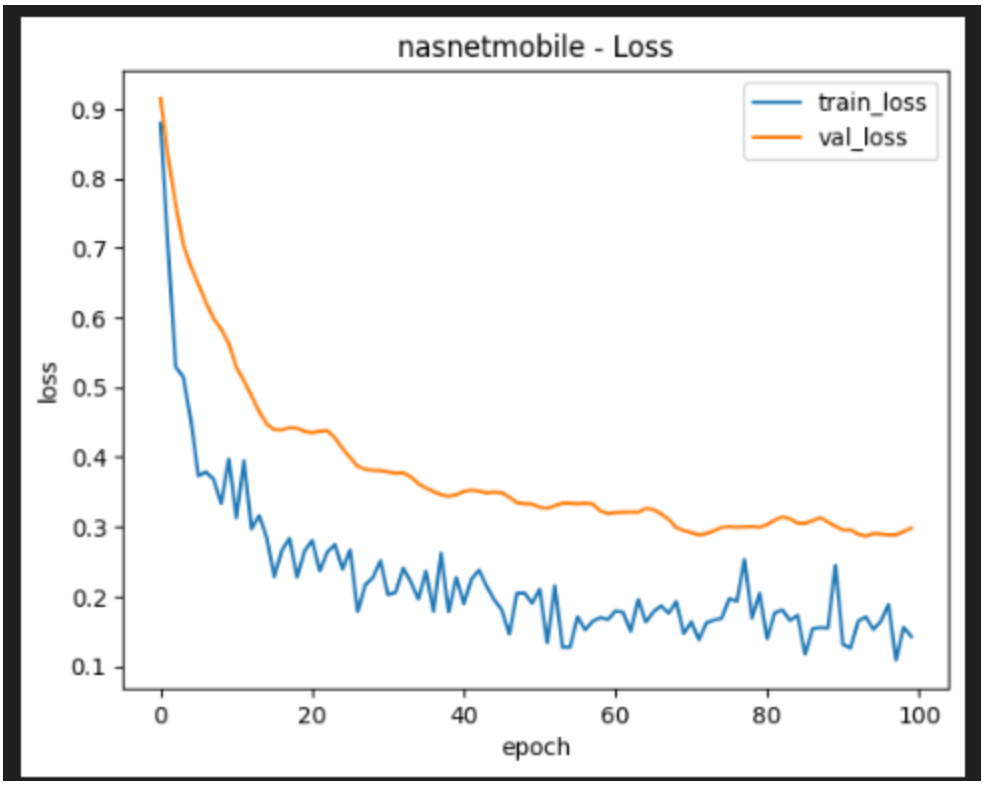
ในส่วน fine-tuning การ unfreeze ประมาณ 20% สุดท้ายของชั้น ResNet50V2 และฝึกต่ออีก 4 epochs ทำให้ train_loss ลดลงมากจากเดิม แต่ val_loss กลับนิ่งอยู่ระดับ ~0.39–0.40 โดยไม่ปรับปรุง ขณะที่ val_acc ยังคงอยู่ระดับ ~0.925 เหมือนเดิมแสดงว่า การ fine-tuning ไม่ได้ช่วยเพิ่มประสิทธิภาพของ ResNet50V2 เลย ตรงกันข้าม train_acc ขยับสูงมากถึง ~0.98 ก่อนจะตกกลับมา ~0.93 ทำให้เห็นสัญญาณ overfitting ชัดเจนเมื่อ weight ถูกปรับมากเกินไปจากข้อมูลน้อย



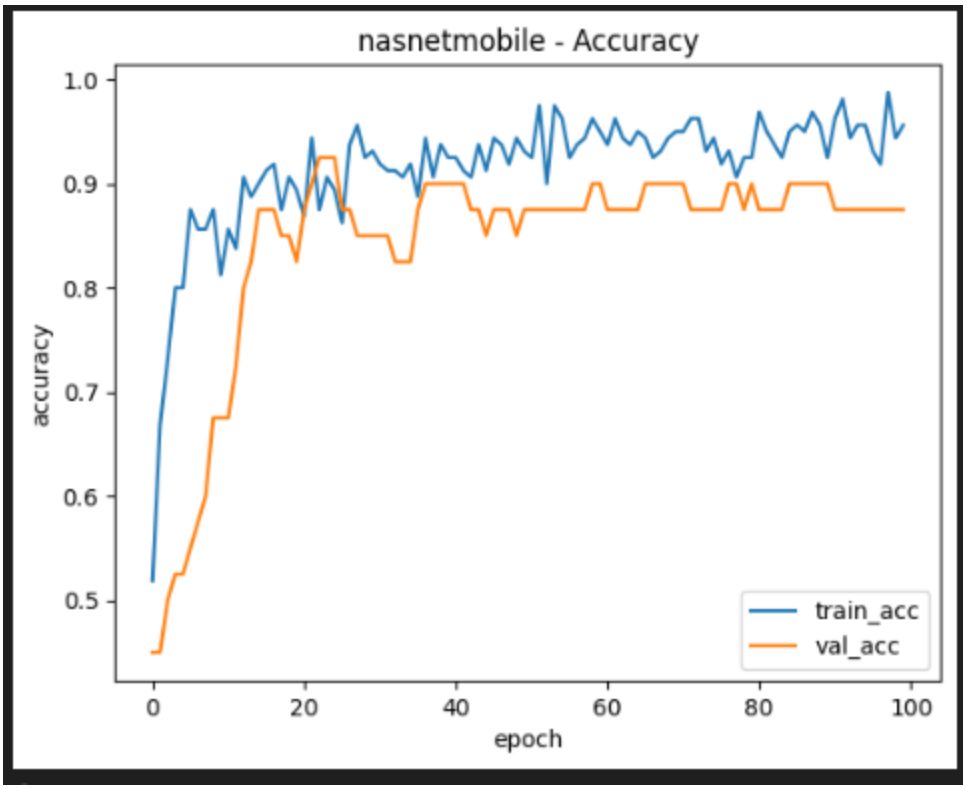
Confusion Matrix หลัง fine-tuning ยืนยันแนวโน้มเดียวกัน โดยจำนวนภาพที่จำแนกถูก/ผิดของทั้งสองคลาสเหมือนเดิมทุกประการกับเฟส frozen base คือ Thai_basil 16/18 และ holy_basil 21/22 แสดงว่า fine-tuning ไม่ได้สร้าง decision boundary ใหม่ที่แตกต่างจากเดิมเลย และ ResNet50V2 มีแนวโน้มเรียนรู้เพียงพอตั้งแต่เฟสแรกแล้ว

โดยสรุป ResNet50V2 เป็นโมเดลที่ให้ผลลัพธ์ดีและเสถียร แต่ไม่เด่นเท่า EfficientNetB0 เนื่องจากไม่สามารถลด val_loss ลงได้มาก และไม่ตอบสนองต่อ fine-tuning เท่าที่ควร แม้จะมีศักยภาพของโมเดลลึก แต่ dataset ที่มีขนาดจำกัดทำให้ความสามารถระดับสูงของ ResNet50V2 ไม่ได้ถูกใช้เต็มประสิทธิภาพ

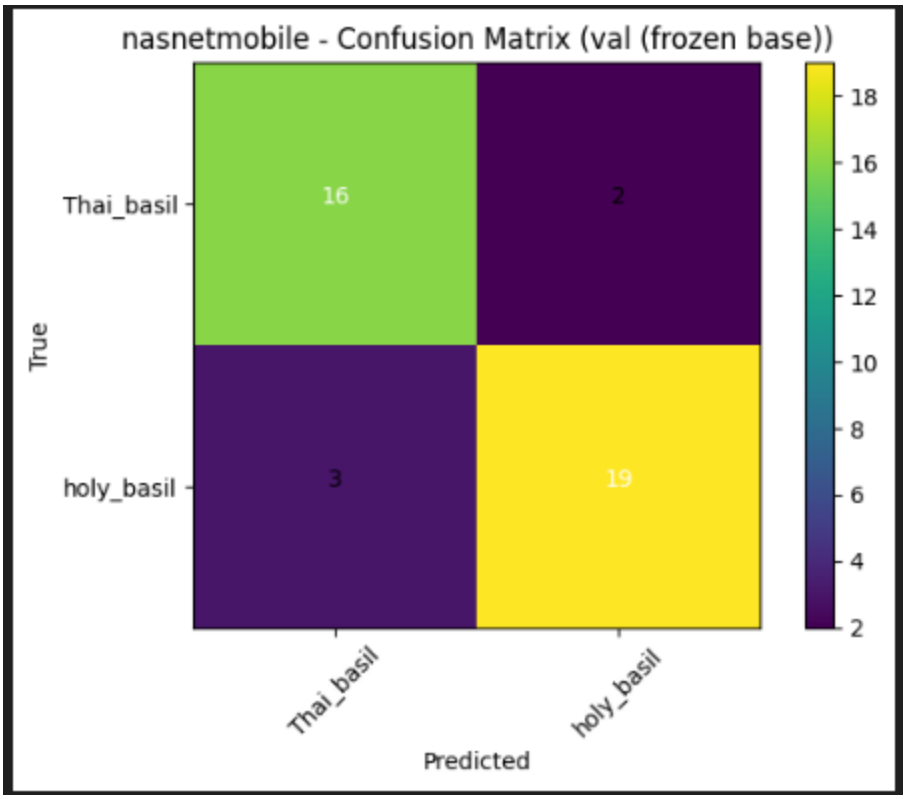
การวิเคราะห์เชิงลึกของโมเดล NASNetMobile NASNetMobile เป็นสถาปัตยกรรมที่ถูกออกแบบด้วย neural architecture search ทำให้มีโครงสร้างที่ค่อนข้างซับซ้อนเมื่อเทียบกับโมเดลระดับ mobile ทั่วไป อย่างไรก็ตาม จากผลการทดลองในงานนี้ พบว่า NASNetMobile ทำงานได้ ดีระดับกลางค่อนไปทางดี แต่ไม่โดดเด่นเท่า EfficientNetB0 หรือ ResNet50V2



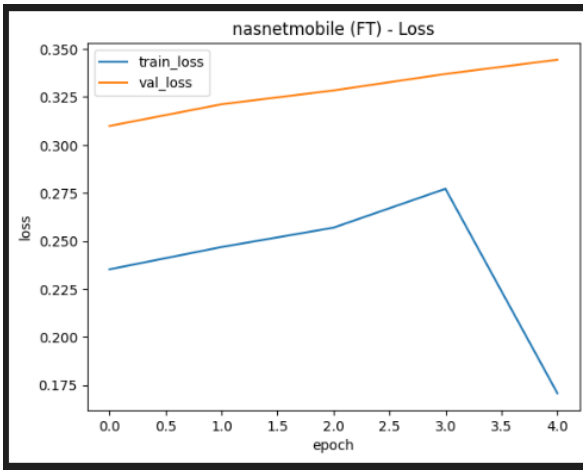
จากกราฟ loss ในเฟส frozen base train_loss ลดลงสม่ำเสมอและค่อนข้างต่ำ (~0.12–0.20) แต่ val_loss คงตัวอยู่ช่วง ~0.28–0.35 โดยแทบไม่ลดลงในช่วงหลังของการฝึก ซึ่งสะท้อนว่าโมเดลเรียนรู้ pattern ของ training set ได้ดี แต่ไม่สามารถ generalize เพิ่มขึ้นเมื่อ epoch สูงขึ้น สัญญาณนี้เข้าข่าย mild overfitting ตั้งแต่ช่วงกลางการฝึกโดยเฉพาะราว epoch 40 เป็นต้นไป

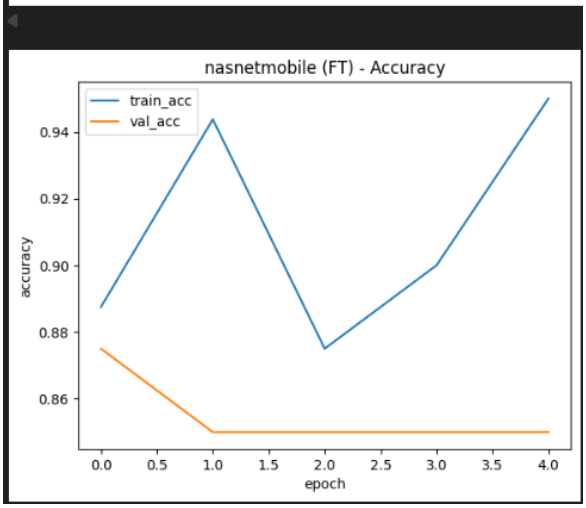


กราฟ accuracy train_acc แตะระดับ ~0.95–0.97 ในช่วงท้าย ในขณะที่ val_acc อยู่ประมาณ ~0.88–0.90 แบบทรงตัว ไม่เพิ่มขึ้นต่อหลังช่วง ~epoch 25–30 แสดงว่าโมเดลเรียนรู้ feature สำคัญได้ระดับหนึ่งแต่ dataset มีความซับซ้อนไม่พอให้โมเดลซับซ้อนอย่าง NASNetMobile ได้แสดงประสิทธิภาพเต็มที่

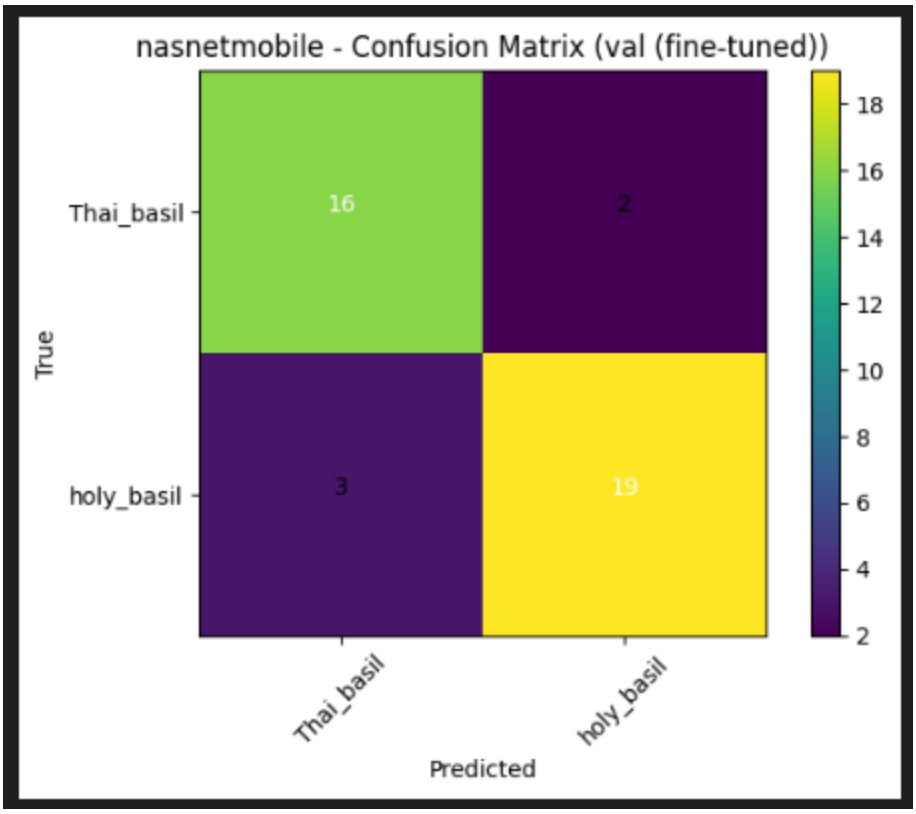


จาก confusion matrix แบบ frozen โมเดลจำแนก Thai_basil ได้ถูกต้อง 16/18 และ holy_basil ถูก 19/22 ซึ่งเป็นอัตราที่รองจาก EfficientNetB0 และ ResNet50V2 แต่ยังคงอยู่ในระดับที่ใช้งานได้ดี





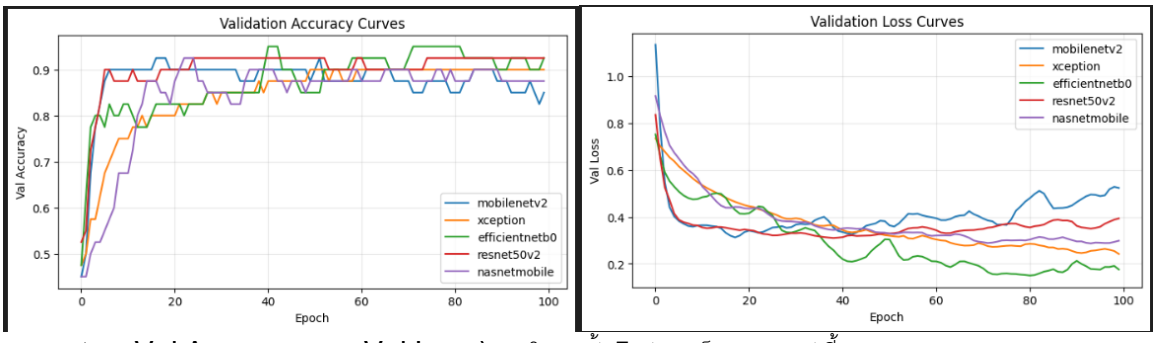
ในเฟส fine-tuning (รูปที่ — กราฟ loss FT NASNetMobile และรูปที่ — กราฟ accuracy FT NASNetMobile) ผลลัพธ์ชัดเจนว่า fine-tuning ไม่ได้ช่วยเพิ่มประสิทธิภาพของโมเดลเลย ค่า val_loss สูงขึ้นเล็กน้อย (~0.31 → ~0.34) และ val_acc ลดลงจาก ~0.90 เหลือ ~0.85–0.87 ตลอด 4 epochs ซึ่งเป็นสัญญาณตรงว่าการ unfreeze ชั้นบนของ NASNetMobile ทำให้โมเดลปรับตัวเข้าหา noise มากกว่าข้อมูลจริง



Confusion Matrix หลัง fine-tune (รูปที่ — CM FT NASNetMobile) ให้ผลเหมือนเดิมทุกประการกับเฟส frozen base (Thai_basil 16/18 และ holy_basil 19/22) ซึ่งสะท้อนว่า decision boundary ของ NASNetMobile ไม่ได้ถูกปรับในทางที่ดีขึ้นเลย

โดยสรุป NASNetMobile ให้ผลลัพธ์ดีระดับหนึ่ง แต่มีข้อจำกัดชัดเจนเมื่อเทียบกับ EfficientNetB0 และ Xception โดยเฉพาะด้าน generalization และการตอบสนองต่อ fine-tuning ทำให้เหมาะกับงานที่ความซับซ้อนต่ำกว่า หรือ dataset มีขนาดใหญ่กว่านี้เพื่อให้โมเดลซับซ้อนระดับ NASNet ดึงศักยภาพออกมาได้เต็มที่

สรุปภาพรวมเพิ่มเติมจากกราฟเปรียบเทียบ (Validation Accuracy & Loss Curves)



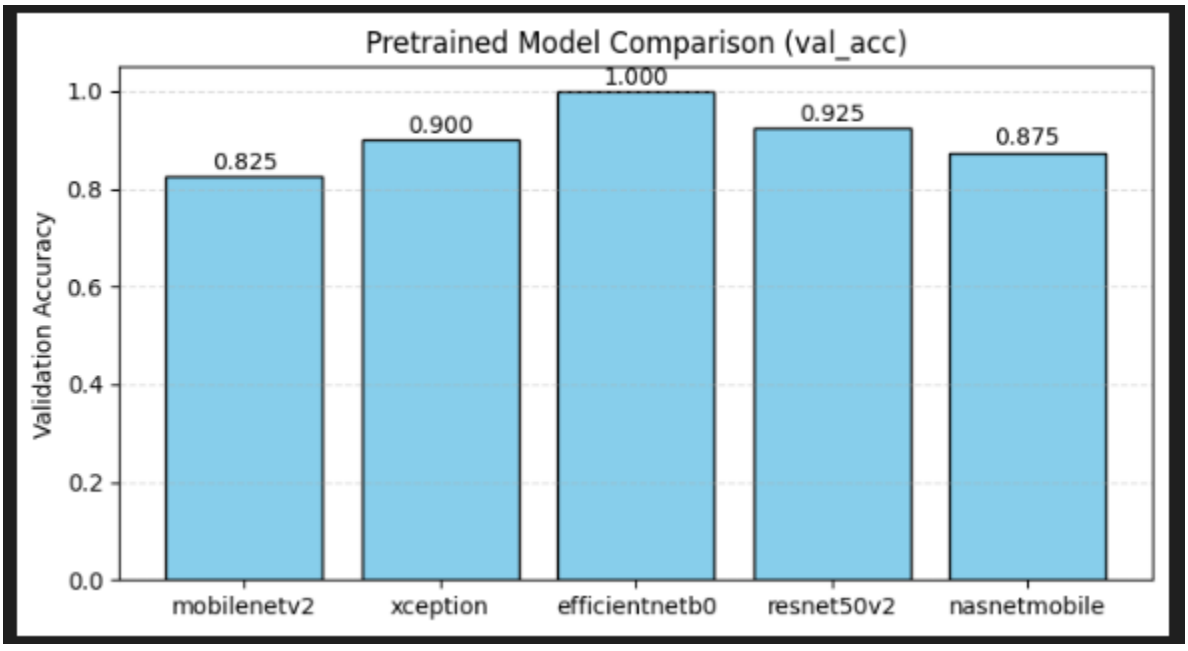
จากกราฟรวม (Val Accuracy และ Val Loss) ของโมเดลทั้ง 5 ตัว จะเห็นภาพรวมดังนี้:

EfficientNetB0 เด่นที่สุด ทั้งด้านความแม่นยำและความเสถียรของ loss โดยกราฟ val_loss ลดลงสม่ำเสมอที่สุด และไม่เกิดการดีดขึ้นท้าย epochs

ResNet50V2 และ NASNetMobile ให้ผลระดับกลางค่อนไปทางสูง แต่กราฟ loss แกว่งตัวมาก แสดงสัญญาณ overfitting เล็กน้อย

MobileNetV2 มี val_acc ดีในช่วงแรก แต่ val_loss แกว่งตัวสูงที่สุดในกลุ่ม สะท้อนความไวต่อ noise และข้อจำกัดของโมเดลขนาดเล็ก

Xception แม้ accuracy ต่ำสุดในกลุ่ม frozen แต่ loss ลดลงนิ่งและเสถียร ถือเป็นโมเดลที่ generalize ได้ดีในระดับข้อมูลจำกัด



กราฟแท่งสรุป (Pretrained Model Comparison) แสดงให้เห็นชัดว่า EfficientNetB0 มี val_accuracy สูงที่สุด (1.00 หลัง fine-tuning) ซึ่งสอดคล้องกับพฤติกรรมจากกราฟทั้งหมด

3.3 ตัวอย่างภาพของข้อมูลที่ใช้ในการทดลอง

เพื่อความชัดเจน ในรูปที่ 3.4–3.6 แสดงตัวอย่างภาพในแต่ละประเภทของข้อมูลที่ใช้ในโครงงานนี้ ได้แก่ ตัวอย่างภาพผลกล้วยและภาพใบกะเพรา/โหระพา

รูปที่ 3.4 ตัวอย่างภาพกล้วยหอม (หนึ่งในสามสายพันธุ์กล้วยที่ใช้ในการทดลอง) – กล้วยหอมมีลักษณะผลยาวโค้ง ผิวเปลือกสีเหลืองมีจุดสีน้ำตาลประปราย เมื่อเปรียบเทียบกับกล้วยน้ำว้าหรือกล้วยไข่จะเห็นว่ากล้วยหอมมีขนาดใหญ่และเรียวยาวกว่า ทำให้เป็นจุดแตกต่างสำคัญที่โมเดลสามารถใช้แยกแยะสายพันธุ์ได้

รูปที่ 3.5 ตัวอย่างภาพใบกะเพรา (Holy basil) – ใบกะเพรามีสีเขียวออกด้าน ผิวใบสากเล็กน้อย ขอบใบหยักแบบฟันเลื่อยไม่เป็นระเบียบ และมักไม่มีดอกให้เห็นชัดในภาพ ใบค่อนข้างบางและเส้นใบชัดเจน ก้านใบและก้านช่อดอกมีสีเขียวอ่อน การแยกใบกะเพราออกจากใบโหระพาด้วยตาเปล่าทำได้โดยสังเกตขอบใบและผิวใบที่สากกว่า

รูปที่ 3.6 ตัวอย่างภาพใบโหระพา (Thai sweet basil) – ใบโหระพามีสีเขียวเข้มกว่าเล็กน้อย รูปร่างใบรียาว ปลายใบแหลม ขอบใบค่อนข้างเรียบหรือหยักน้อยกว่ากะเพรา ผิวใบเรียบมันมากกว่า ใบหนากว่าเล็กน้อย
นอกจากนี้ลักษณะเด่นคือมักมีส่วนของลำต้นหรือช่อดอกสีม่วงให้เห็น (ดังในภาพจะเห็นก้านและยอดดอกสีม่วงแดง) กลิ่นของใบโหระพาหอมกว่าแต่ในภาพถ่ายต้องอาศัยลักษณะทางสายตาดังที่กล่าวมาในการจำแนก

จากตัวอย่างภาพข้างต้น โมเดล Machine Learning และ CNN ในงานนี้จะได้รับข้อมูลที่มีลักษณะดังที่เห็น ซึ่งโมเดลจำเป็นต้องเรียนรู้ความแตกต่างเพียงเล็กน้อยของรูปร่างและรายละเอียดพื้นผิว/เส้นใบเพื่อนำไปสู่การจำแนกที่ถูกต้อง

บทที่ 4 อภิปรายผล

จากผลการทดลองในบทที่ 3 สามารถสรุปและอภิปรายประเด็นที่น่าสนใจเกี่ยวกับประสิทธิภาพของโมเดลต่าง ๆ และลักษณะของข้อมูลได้ดังนี้:

4.1 เปรียบเทียบโมเดลที่ดีที่สุดในแต่ละกรณี

การจำแนกสายพันธุ์กล้วยด้วยวิธี Machine Learning พบว่าโมเดลที่ให้ผลดีที่สุดคือ Logistic Regression แม้เป็นโมเดลเชิงเส้น แต่สามารถใช้ประโยชน์จากชุดฟีเจอร์ที่สกัดมาได้ดี (shape, LBP, HOG) จึงแยกสายพันธุ์ได้แม่นยำ ขณะที่ Random Forest และ MLP ก็ให้ผลดีรองลงมา ส่วน Decision Tree มีแนวโน้มจำข้อมูลมากเกินไป (overfit) และ Naïve Bayes ไม่เหมาะกับฟีเจอร์ที่มีความสัมพันธ์กันสูง

สำหรับงานจำแนกใบกะเพรา–โหระพาด้วย CNN โมเดลที่ดีที่สุดคือ EfficientNetB0 ซึ่งใช้จำนวนพารามิเตอร์น้อยแต่ให้ความแม่นยำสูง เหมาะกับงานที่ข้อมูลมีจำนวนจำกัด โมเดลที่ซับซ้อนกว่านี้ เช่น ResNet50V2 หรือ NASNetMobile แม้มีศักยภาพสูง แต่ไม่ได้ให้ผลเพิ่มขึ้นภายใต้ขนาดข้อมูลชุดนี้

4.2 ปัญหา Overfitting ที่พบ

โมเดลบางตัว โดยเฉพาะ CNN ในช่วง fine-tuning แสดงอาการ overfitting ชัดเจน เช่น ความแม่นยำบนชุดฝึกเพิ่มขึ้น แต่ความแม่นยำบนชุด validation ลดลง สาเหตุหลักมาจากจำนวนข้อมูลที่น้อยและความซับซ้อนของโมเดลที่สูง การใช้ data augmentation ช่วยลดผลกระทบได้บ้าง แต่ยังไม่เพียงพอในบางกรณี

ในฝั่ง Machine Learning เช่น Decision Tree ก็พบอาการลักษณะเดียวกัน โดย train accuracy สูงมากแต่ validation accuracy ต่ำกว่าอย่างเห็นได้ชัด ทำให้ Random Forest ซึ่งรวมต้นไม้หลายต้นกลายเป็นทางเลือกที่เสถียรและทนต่อ overfitting มากกว่า

4.3 การวิเคราะห์จาก Confusion Matrix

งานใบกะเพรา–โหระพา (CNN)
โมเดลเกือบทุกตัวแทบไม่สับสน *holy_basil → Thai_basil* แต่มีโอกาสสับสน *Thai_basil → holy_basil* มากกว่า โดยเฉพาะภาพที่มีลักษณะคล้ายกะเพรา เช่น ใบหยักมากหรือสีกลาง ๆ ทำให้โมเดลถูกดึงไปทำนายเป็น holy_basil

งานกล้วย 3 สายพันธุ์ (Machine Learning) – จาก Confusion Matrix ที่ได้

- คู่ที่สับสนมากที่สุดคือ กล้วยน้ำว้า ↔ กล้วยไข่ เพราะมีรูปทรงและพื้นผิวใกล้เคียงที่สุด
- กล้วยหอม ถูกจำแนกได้ดีที่สุด เนื่องจากมีลักษณะเด่นชัด เช่น ความโค้งและขนาดเฉพาะตัว
- Logistic Regression และ Random Forest มีจุดผิดพลาดน้อยที่สุด ขณะที่ Decision Tree ผิดเยอะที่สุด เพราะ overfit

4.4 ข้อดีข้อเสียของวิธีการแต่ละแบบ

Machine Learning แบบดั้งเดิม: ข้อดีคือ ตีความง่ายและปรับแต่งได้ตรงจุด เช่น วิเคราะห์ได้ว่าฟีเจอร์ใดสำคัญ แต่ข้อเสียคือขึ้นอยู่กับคุณภาพของฟีเจอร์ที่สกัด หากเลือกฟีเจอร์ไม่ดี โมเดลจะไม่สามารถดึงความแตกต่างที่แท้จริงออกมาได้

Deep Learning (CNN): ข้อดีคือ ไม่ต้องออกแบบฟีเจอร์เอง โมเดลเรียนรู้คุณลักษณะที่สำคัญจากภาพได้โดยตรง เหมาะกับงานที่ความต่างละเอียดมาก เช่น กะเพรา–โหระพา ข้อเสียคือ ต้องใช้ทรัพยากรสูง ตีความได้ยากกว่า และเสี่ยง overfitting หากข้อมูลไม่พอ

4.5ข้อสังเกตเพิ่มเติม

คุณภาพและความสมดุลของข้อมูลส่งผลต่อผลลัพธ์อย่างมาก โดยเฉพาะงานใบสมุนไพรที่มีตัวอย่างค่อนข้างน้อย แต่ด้วยการใช้ data augmentation และโครงสร้างโมเดลที่เหมาะสม ทำให้โมเดลยังคงให้ผลแม่นยำสูง จุดนี้สะท้อนว่าในบริบทข้อมูลจำกัด การเลือกโมเดลและการเตรียมข้อมูลมีผลต่อความสำเร็จมากกว่าแค่เลือกโมเดลที่ใหญ่ที่สุดหรือซับซ้อนที่สุด