

# Quantitative Analysis of Tree Populations in Forested Landscapes via UAV

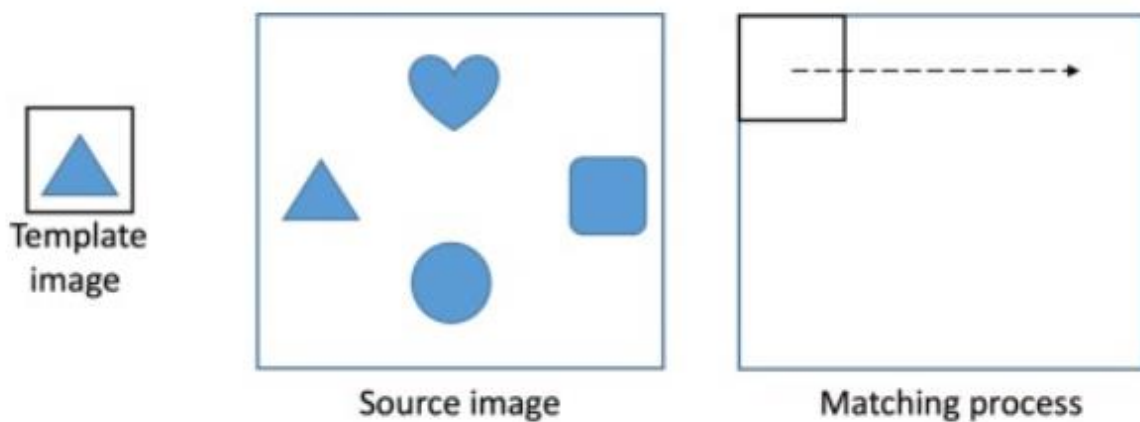
NATSINEE SASANASOPA

INTERN RESEARCH AND DEVELOPMENT PROJECT FROM MAPEDIA

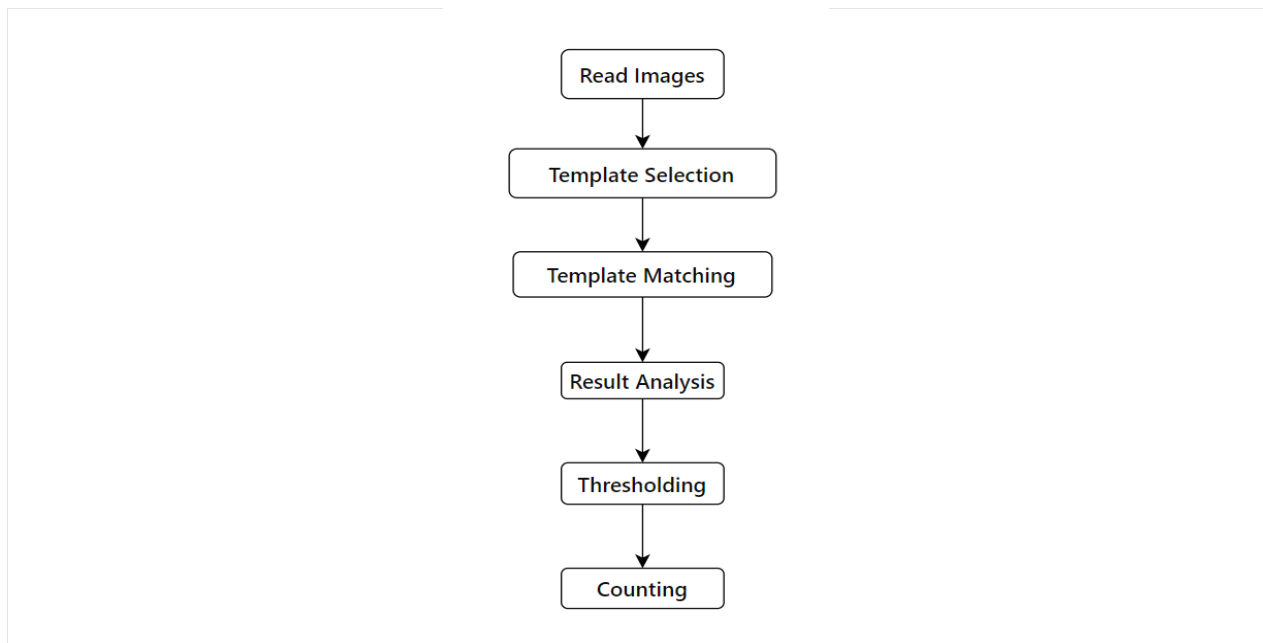
## Template Matching Method

TM หรือ Template Matching เป็นเทคนิคที่ใช้กันอย่างแพร่หลายในการภาพการแพทย์เพื่อตรวจจับวัตถุต่าง ๆ ในภาพ โดยการใช้เทคนิคนี้ เราจะเปรียบเทียบส่วนหนึ่งของภาพต้นฉบับกับภาพแม่แบบที่เป็นภาพของวัตถุหรือรูปแบบเล็ก ๆ และนำมาใช้เป็นแม่แบบเพื่อตรวจจับวัตถุหรือรูปแบบที่คล้ายกันในภาพต้นฉบับ ในกรณีที่ความแตกต่างระหว่างภาพแม่แบบและภาพต้นฉบับเล็กน้อย Template Matching ก็ยังมีประโยชน์

กระบวนการการจับคู่นี้จะเลื่อนภาพแม่แบบไปยังทุกตำแหน่งที่เป็นไปได้ในภาพต้นฉบับ โดยใช้วิธีการ เป็นการคำนวณ pixel ต่อ pixel และคำนวณดัชนีตัวเลข เช่น ค่าสัมพันธ์ ซึ่งบ่งชี้ถึงความเข้ากันได้ระหว่างภาพแม่แบบและภาพในตำแหน่งนั้น



Template Matching Process :



## วิธีการทดลอง

1. **Read Images:** ใช้ `Image.open()` จาก PIL (Python Imaging Library) เพื่อเปิดไฟล์รูปภาพ จากนั้นจะแปลงรูปภาพเป็น NumPy arrays เพื่อประมวลผล ซึ่งเป็นวิธีที่พบบ่อยสำหรับงานวิเคราะห์ภาพ

```
• %pylab inline
  from PIL import Image
  ImagenTotal = numpy.asarray(Image.open('Rst/test/test.jpg'))
  ImagenTemplate = numpy.asarray(Image.open('Rst/test/test_1.jpg'))
```

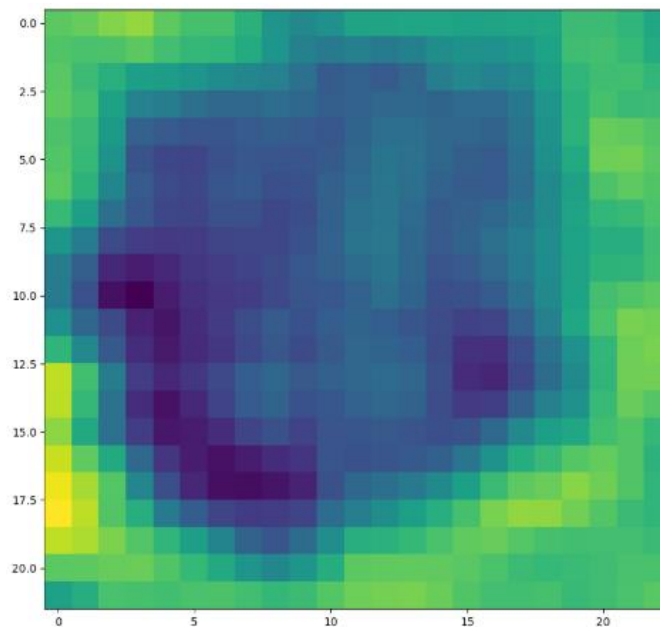
Python

%pylab is deprecated, use %matplotlib inline and import the required libraries.  
Populating the interactive namespace from numpy and matplotlib

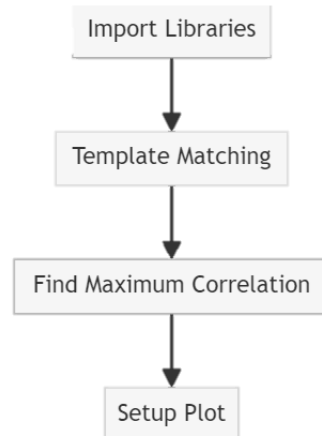
2. **Template Selection:** เลือกรูปภาพของต้นไม้แม่แบบ (ImagenTemplate) ซึ่งจะใช้เป็นแม่แบบในการค้นหาการจับคู่ในภาพที่ใหญ่ขึ้น (ImagenTotal) แม่แบบนี้ต้องมีลักษณะคล้ายคลึงกับต้นไม้ที่ต้องการนับ

```
imagen = ImagenTotal[:, :, 1]
arbol = ImagenTemplate[:, :, 1]
imshow(arbol)
```

Python



3. **Template Matching:** ใช้ฟังก์ชัน `match_template` จาก `skimage.feature` เพื่อทำการจับคู่แม่แบบในภาพ `imagen` โดยใช้แม่แบบ `arbol` ที่ได้รับเข้ามาเป็นอาร์กิวเมนต์ จากนั้นนำผลลัพธ์ที่ได้จากการจับคู่แม่แบบ ใช้ `np.argmax` เพื่อหาตำแหน่งที่มีค่าสูงสุด และ `np.unravel_index` เพื่อแปลงตำแหน่งนั้นเป็นคู่ pixel (x, y)



3.1) Import Libraries:

- `numpy as np`: ใช้สำหรับการดำเนินการทางตัวเลขกับอาร์เรย์
- `matplotlib.pyplot as plt`: ใช้ในการพล็อตที่คล้ายกับ MATLAB
- `data from skimage`: โมดูลนี้ให้การเข้าถึงรูปภาพทดสอบและข้อมูลตัวอย่าง
- `match_template from skimage.feature`: ฟังก์ชันนี้ใช้สำหรับการจับคู่ Template ซึ่งเปรียบเทียบรูปภาพ Template กับรูปภาพเป้าหมายเพื่อหาการตรงกัน

3.2) Template Matching:

- `result = match_template(imagen, arbol)`: บรรทัดนี้ทำการจับคู่ Template โดย `imagen` คือรูปภาพเป้าหมายที่ต้องค้นหา และ `arbol` คือรูปภาพ Template ที่ต้อง Matching ฟังก์ชันนี้ส่งคืนอาร์เรย์ 2 มิติ ที่แต่ละองค์ประกอบบ่งชี้ถึงความใกล้เคียงของ Template กับเป้าหมายในตำแหน่งนั้น

3.3) Find the Maximum Correlation:

- `ij = np.unravel_index(np.argmax(result), result.shape)`: คำสั่งนี้หา index ของค่าสูงสุดใน `result` ซึ่งสอดคล้องกับตำแหน่งที่ตรงกับ Template ที่ดีที่สุดใน `imagen`
- `x, y = ij[::-1]`: คำสั่งนี้แปลง index เป็นพิกัด (x, y) สำหรับการพล็อต โดย `[::-1]` กลับ index เพื่อตรงกับระบบพิกัดที่ใช้โดย `matplotlib`

3.4) Plotting

```

result = match_template(imagen, arbol)
ij = np.unravel_index(np.argmax(result), result.shape)
x, y = ij[::-1]

fig = plt.figure(figsize=(8, 3))
ax1 = plt.subplot(1, 3, 1)
ax2 = plt.subplot(1, 3, 2, adjustable='box')
ax3 = plt.subplot(1, 3, 3, sharex=ax2, sharey=ax2, adjustable='box')

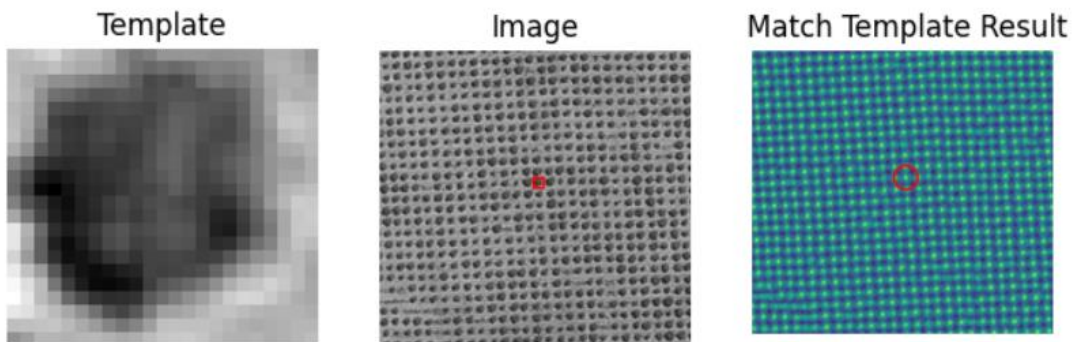
ax1.imshow(arbol, cmap=plt.cm.gray)
ax1.set_axis_off()
ax1.set_title('template')

ax2.imshow(imagen, cmap=plt.cm.gray)
ax2.set_axis_off()
ax2.set_title('imagen')
harbol, warbol = arbol.shape
rect = plt.Rectangle((x, y), warbol, harbol, edgecolor='r', facecolor='none')
ax2.add_patch(rect)

ax3.imshow(result)
ax3.set_axis_off()
ax3.set_title('match_template\nresult')
ax3.autoscale(False)
ax3.plot(x, y, 'o', markeredgecolor='r', markerfacecolor='none', markersize=10)

plt.show()

```



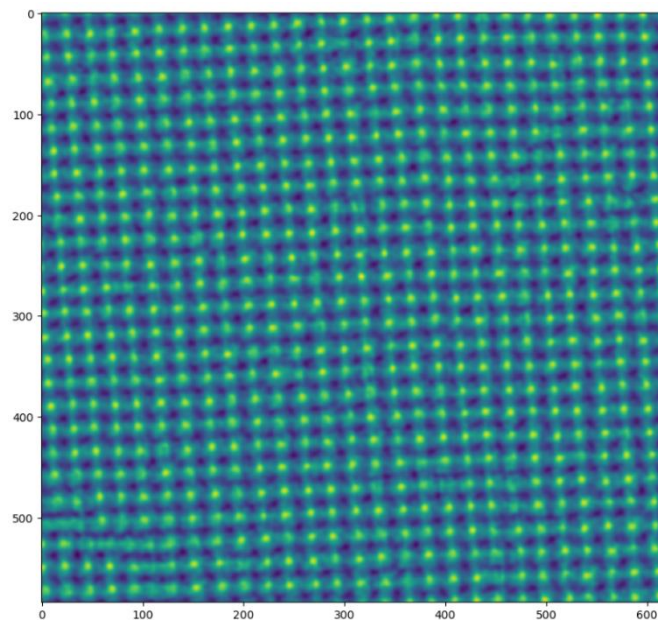
4. **Result Analysis:** ผลลัพธ์จาก Template Matching ที่ต่อไปจะเป็นอาร์เรย์ที่มีค่าแต่ละตำแหน่งแทนคุณภาพของการจับคู่ที่จุดนั้น ๆ แล้วทำการวิเคราะห์ผลลัพธ์นี้เพื่อหาตำแหน่งของการ Matching ที่ดี

```

#closer look of the match template
imshow(result);
figure(figsize = (10,10));

```

Python

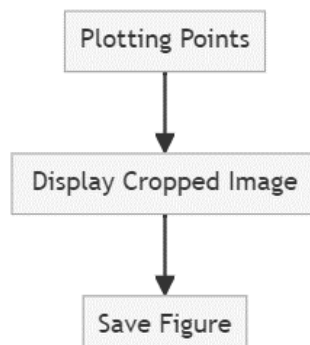


5. **Thresholding:** ใช้เกณฑ์ในการตัดสินใจว่าผลลัพธ์ใดบ้างที่ดีพอที่จะนับว่าเป็นต้นไม้ นี่เป็นขั้นตอนสำคัญในการลดจำนวนการนับที่ผิดพลาด และขึ้นอยู่กับคุณภาพของ Template ที่ใช้ และความแตกต่างของลักษณะต้นไม้ในภาพ

```
resultadosfiltrados = np.where(result>0.3)
resultadomaximo = np.where(result>0.99)
```

Python

6. **Counting:** ขั้นตอนสุดท้ายคือการนับจำนวนต้นไม้ แต่ครั้งที่ตรงกันนั้นสอดคล้องกับ Template โดย ความความแม่นยำในการนับขึ้นอยู่กับความแม่นยำที่จุดสีแดงสามารถถูกแยกออกจากส่วนอื่นๆ ของภาพ ซึ่งขึ้นอยู่กับทางเลือก HSV ที่เลือกใช้



#### 6.1) Plotting Points:

- for loop ทุกจุดในอาร์เรย์ ที่ผ่านการ filter แล้ว (resultadosfiltrados) โดยพล็อตจุดสีแดงบริเวณที่ผ่านการ Matching ว่าเป็นต้นไม้

#### 6.2) Saving the Figure:

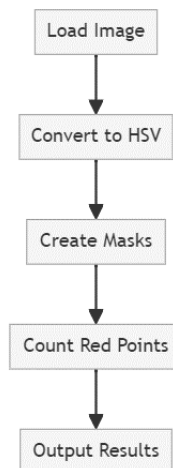
- plt.savefig(...): บันทึกภาพปัจจุบันไปยัง path ที่ระบุไว้ ด้วยความละเอียดสูง (300 dpi)

```
for punto in range(shape(resultadosfiltrados)[1]):
    plot(resultadosfiltrados[1][punto], resultadosfiltrados[0][punto], 'o',
          markeredgecolor='r', markerfacecolor='none', markersize=0.2)

imshow(ImagenTotal[10:-10,10:-10,:])
figsize(10,10)

plt.savefig("C:/Users/beckt/Downloads/11111/Rst/output/00.png", dpi=300, bbox_inches='tight')
```

✓ 32.4s Python



#### 6.3) Image Processing with PIL and OpenCV:

- โหลดภาพที่บันทึกและแปลงเป็นอาร์เรย์ NumPy
- แปลงภาพจากสี RGB เป็นสเปซสี HSV โดยใช้ OpenCV
- กำหนดช่วงสี HSV สำหรับสีแดงและสร้าง Mask เพื่อแยกพื้นที่สีแดง
- นำ Mask ที่ใช้ มาใช้เพื่อนับจุดสีแดงและใช้ส่วนประกอบที่เชื่อมต่อกันเพื่อนับพื้นที่สีแดงที่แตกต่างกัน

#### 6.4) Output:

- แสดงภาพหน้าฉากที่แสดงจุดสีแดง
- แสดงผลการนับจุดสีแดงและจำนวนพื้นที่สีแดงที่แตกต่างกัน (แสดงถึงจำนวนต้นไม้)

```
from PIL import Image
import numpy as np
import cv2

# Load the image
image_path = 'RST/output/00.png'
image = Image.open(image_path)
image_np = np.array(image)

# Convert to HSV color space
hsv_image = cv2.cvtColor(image_np, cv2.COLOR_RGB2HSV)

# Define range for red color and create a mask
lower_red1 = np.array([0, 120, 70])
upper_red1 = np.array([10, 255, 255])
lower_red2 = np.array([170, 120, 70])
upper_red2 = np.array([180, 255, 255])

mask1 = cv2.inRange(hsv_image, lower_red1, upper_red1)
mask2 = cv2.inRange(hsv_image, lower_red2, upper_red2)
full_mask = mask1 + mask2

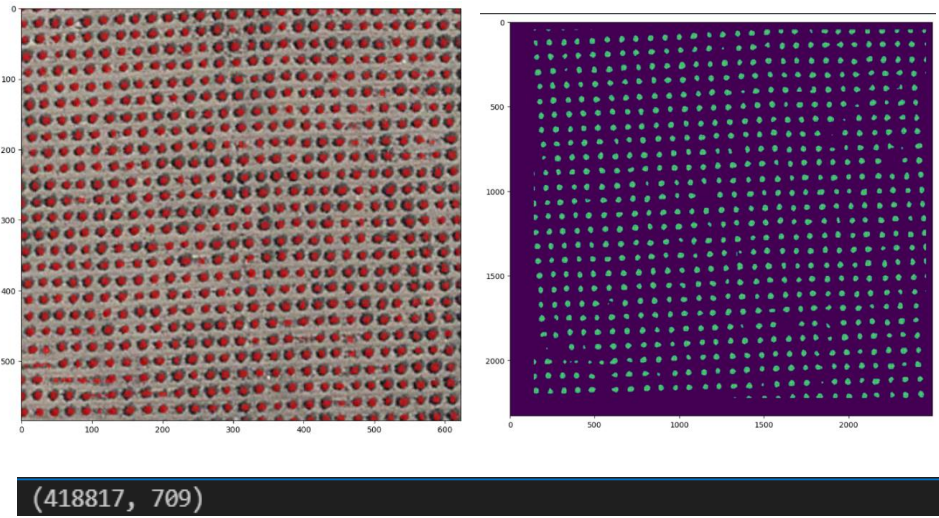
# Count the red points
red_points_count = np.sum(full_mask > 0)

# We can also use connected components to count distinct red points
num_labels, labels_im = cv2.connectedComponents(full_mask)

red_points_count, num_labels - 1 # Subtract 1 for the background label
imshow(full_mask)
```

✓ 0.9s Python

ผลการทดลอง

Output	<div></div>
จำนวน pixel	418817
จำนวนต้นไม้ที่นับได้	709



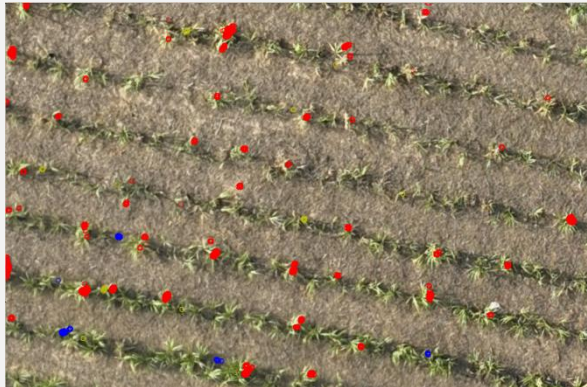
## สรุปผลการทดลอง

### ข้อดี

- ประสิทธิภาพ: นับต้นไม้ได้เร็ว
- ความง่าย: ตัวโปรแกรมเขียนง่ายไม่มีความซับซ้อน
- การใช้ซ้ำ: ต้นแบบสามารถนำมาใช้ซ้ำได้สำหรับภาพต่างๆ ของพื้นที่เดียวกันหรือภาพที่มีประเภทต้นไม้ที่คล้ายกัน

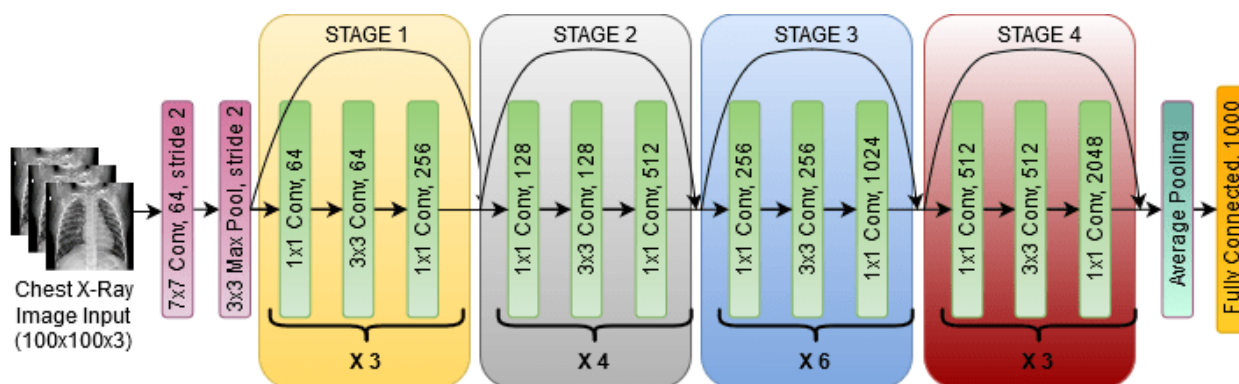
### ข้อเสีย

- ความแม่นยำ : มีปัญหากับความแม่นยำในการนับ เนื่องจากลักษณะของต้นไม้เนื่องจากแสง, ฤดูกาล , การเปลี่ยนมุมมองอาจ หรือ พลาดนับต้นไม้ที่ถูกบังหรือทับซ้อนกัน ทำให้นับไม่ครบ
- ความไว: อาจไวเกินไปต่อต้นแบบ วัตถุที่ไม่ใช่ต้นไม้แต่มีลักษณะคล้ายกับต้นแบบอาจถูกนับเข้าไปโดยผิดพลาด ทั้งนี้ เกิดขึ้นได้จาก Template และ การกำหนด Threshold

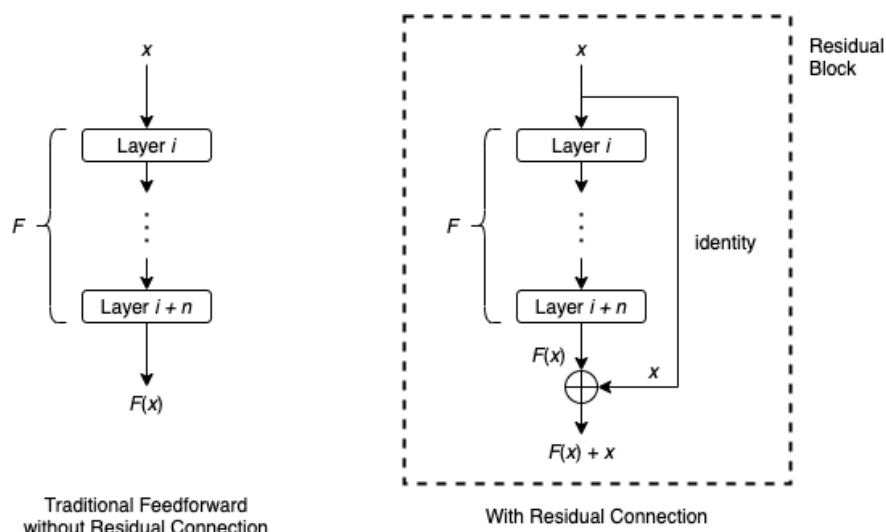


## Resnet - 50 Model

ResNet-50 เป็นโมเดล Deep learning ยอดนิยม อยู่ในกลุ่มของ Residual Networks ซึ่งได้ปฏิวัติวิธีการสร้าง Neural networks โดยการนำเสนอแนวคิดของ "การเรียนรู้ที่เหลือ (Residual Learning)" โมเดลนี้ถูกนำเสนอครั้งแรกโดย Kaiming He และคณะในเอกสารวิจัยปี 2015 ที่มีชื่อว่า "Deep Residual Learning for Image Recognition" และได้กลายเป็นโมเดลพื้นฐานในสาขา Computer Vision เนื่องจากมีประสิทธิภาพและประสิทธิผลสูง



คำว่า "50" ใน ResNet-50 หมายถึงจำนวน Layer ซึ่งรวมถึงชั้น Convolutional 48 Layer พร้อมกับชั้น MaxPool และ AveragePool อย่างละ 1 Layer ทำให้เป็น Deep network และสามารถจัดการกับงานรู้จำภาพที่ซับซ้อนได้ สิ่งที่ทำให้ ResNet-50 แตกต่างคือการใช้ Residual Blocks ซึ่งรวมถึง "การเชื่อมต่อทางลัด" หรือ "Skip Connections" ที่ช่วยให้สามารถ input ข้อมูลนำเข้าของ Layer นั้นๆ ไปยังผลลัพธ์ได้ ช่วยลดปัญหาของการหายไปของการเรียนรู้ (Vanishing Gradients) ซึ่งเป็นปัญหาทั่วไปใน deeper networks ทำให้ ResNet-50 สามารถเรียนรู้จากข้อมูลจำนวนมากได้โดยมีความแม่นยำที่ดีขึ้น แม้ว่าความลึกของเครือข่ายจะเพิ่มขึ้น การออกแบบนี้ไม่เพียงแต่ช่วยปรับปรุงกระบวนการ Training เท่านั้น แต่ยังช่วยเพิ่มประสิทธิภาพของเครือข่ายในงานทำนายหลากหลาย ตั้งแต่การจดจำวัตถุพื้นฐานในภาพไปจนถึงการใช้งานที่ซับซ้อนกว่า เช่น การจับข้อดีโนมิติและการวิเคราะห์ภาพทางการแพทย์



## วิธีการทดลอง

### 1. Install Python libraries:

- 1.1) numpy (NumPy): สำหรับการคำนวณทางวิทยาศาสตร์ใน Python ช่วยสำหรับการคำนวณในอาร์เรย์และเมทริกซ์ขนาดใหญ่และหลายมิติ พร้อมฟังก์ชันทางคณิตศาสตร์ระดับสูงจำนวนมากเพื่อดำเนินการกับอาร์เรย์
- 1.2) pandas: Library ที่ให้โครงสร้างข้อมูลที่ใช้ทำงานง่ายและมีประสิทธิภาพสูงและเหมาะสำหรับการประมวลผลและการจัดการข้อมูลแบบตารางและชุดข้อมูลเวลา
- 1.3) matplotlib: Library สำหรับการพล็อต
- 1.4) seaborn: Library การสร้างแผนผังข้อมูลใน Python เป็น Library สำหรับการวาดกราฟิกทางสถิติที่ดี
- 1.5) torch (PyTorch): สามารถช่วยในการประมวลผลโดยใช้ GPU ในการคำนวณ
- 1.6) torchvision: package ที่ประกอบด้วย datasets, model architectures, และ transformations ของภาพ
- 1.7) Pillow (PIL Fork): Library การประมวลผลภาพ Python เพิ่มความสามารถในการประมวลผลภาพ และการวิเคราะห์รูปแบบไฟล์หลายประเภท
- 1.8) scikit-learn: เครื่องมือที่ง่ายและมีประสิทธิภาพสำหรับการวิเคราะห์ข้อมูลเพื่อการทำนาย

```
%pip install numpy pandas matplotlib seaborn torch torchvision Pillow scikit-learn
```

Python

### 2. Set a Random Seed: กำหนดค่า Seed เพื่อทำการ Random

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import torchvision
from torchvision import transforms, models

from sklearn.metrics import accuracy_score, precision_recall_fscore_support

# Set a random seed for reproducibility.
torch.manual_seed(0)
np.random.seed(0)
```

Python

### 3. Read file : ทำการ read ไฟล์ dataset ที่มีอยู่

```
# Load the CSV files
train_df = pd.read_csv('Train.csv')
test_df = pd.read_csv('Test.csv')
```

Python

4. **batching and loading the data:** นี่คือการโหลดข้อมูลที่กำหนดเองชื่อ 'TreesDataset' ซึ่งสืบทอดมาจากคลาส Dataset ของ PyTorch คลาสนี้ถูกออกแบบมาเพื่อใช้ร่วมกับ DataLoader ของ PyTorch เพื่อการโหลด และ batching ข้อมูล

```
class TreesDataset(Dataset):
    def __init__(self, dataframe, image_dir, transform=None):
        """
        Args:
            dataframe (pd.DataFrame): Dataframe containing image paths and targets.
            image_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied on a sample.
        """
        self.dataframe = dataframe
        self.image_dir = image_dir
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        img_name = os.path.join(self.image_dir, self.dataframe.iloc[idx, 0])
        image = Image.open(img_name)
        target = self.dataframe.iloc[idx, 1] if len(self.dataframe.columns) > 1 else None

        if self.transform:
            image = self.transform(image)

        return image, target
```

Python

5. **Data Transformations:** ขั้นตอนนี้มีจุดมุ่งหมายเพื่อปรับขนาดข้อมูลทำให้ Model มีประสิทธิภาพมากขึ้น โดยจะปรับขนาดรูปภาพให้เป็นขนาดที่สอดคล้องกันแปลงเป็นเทนเซอร์

```
# Define your transformations
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
])
```

Python

6. **Dataset Creation:** สร้างชุดข้อมูลที่พร้อมสำหรับการ train/test

```
# Create the dataset
train_dataset = TreesDataset(dataframe=train_df, image_dir='images', transform=transform)
test_dataset = TreesDataset(dataframe=test_df, image_dir='images', transform=transform)

# Create the DataLoader
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)
```

Python

7. **Model Importing:** การนำเข้าโมเดล ResNet50 จากไลบรารี torchvision ซึ่งมีรายละเอียดของวิธีการนี้

- Transfer Learning: การใช้ประโยชน์จากโมเดลที่ได้รับการ train มาแล้วมาปรับให้เข้ากับชุดข้อมูลที่เฉพาะเจาะจงซึ่งอาจมีขนาดเล็กกว่า ทำให้โมเดลได้เรียนรู้คุณลักษณะที่จำเพาะของต้นไม้
- Fine-tuning: การปรับเปลี่ยนขั้นสุดท้ายของโมเดล ที่ได้รับการ train มาแล้วเพื่อให้สามารถทำนายได้ และเป็นการปรับให้โมเดลสามารถจดจำและคาดการณ์ภาพต้นไม้ที่ไม่เคยเห็นมาก่อนได้

```
from torchvision.models import resnet50

# Load the pre-trained model
model = resnet50(pretrained=True)
num_fts = model.fc.in_features
model.fc = nn.Linear(num_fts, 1)  ## Output one continuous value

# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)  # Lower lr for fine-tuning
```

Python

8. **Training Parameters:** การตั้งค่า `num_epochs = 10` หมายถึงการกำหนดจำนวนครั้งที่ชุดข้อมูลทั้งหมดจะถูกนำผ่าน neural network ระหว่างการtrain โดยหนึ่ง epoch หมายถึงหนึ่งรอบการทำงานครบทั้งชุดข้อมูลการ train พารามิเตอร์นี้มีความสำคัญในการกำหนดระยะเวลาการฝึกฝนของโมเดล หากจำนวน epochs น้อยเกินไป อาจทำให้โมเดลไม่สามารถทำงานได้ดีเพียงพอ (underfit) ในขณะที่จำนวน epochs มากเกินไปอาจทำให้โมเดลทำงานไม่ดีบนข้อมูลที่ไม่เคยพบมาก่อน (overfitting).

```
num_epochs = 10

for epoch in range(num_epochs):
    model.train()  # Set model to training mode
    running_loss = 0.0
    for images, counts in train_loader:  # Assuming counts is a tensor with the number of trees
        counts = counts.float().view(-1, 1)  # Ensure counts is the correct shape and type

        optimizer.zero_grad()

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, counts)

        # Backward and optimize
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

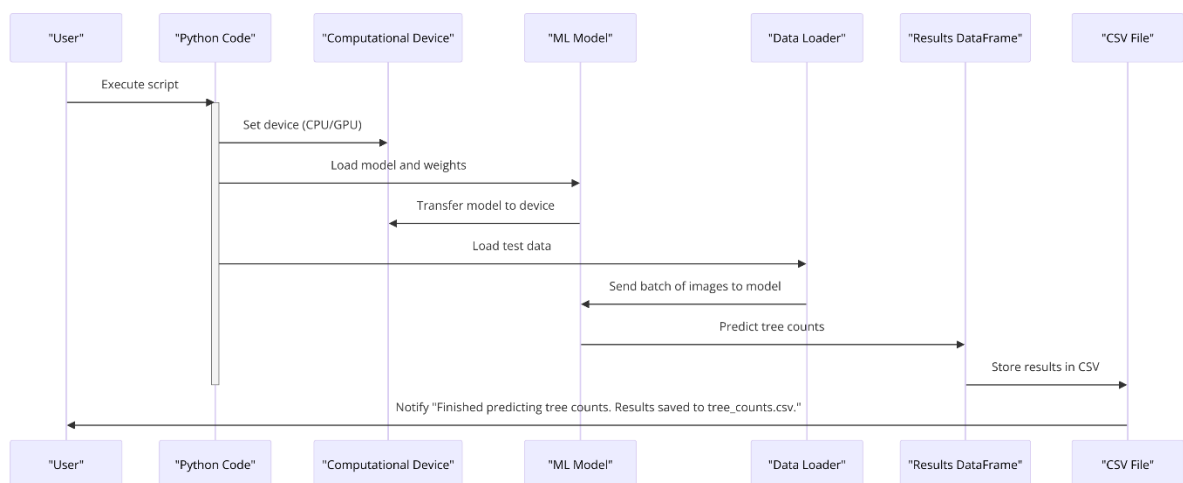
    print(f'Epoch {epoch+1}/{num_epochs}, Loss: {running_loss/len(train_loader)}')

    # Implement a validation phase if you have a validation dataset

torch.save(model.state_dict(), 'model_weights.pth')
```

9. **Prediction:** นำ weight model ที่ได้มาทำการ predict จำนวนต้นไม้

- กำหนด Transformation สำหรับข้อมูล กำหนด transformations ที่จะใช้ในการประมวลผลรูปภาพ ดังนี้
  - การปรับขนาดรูปภาพเป็น 256x256 พิกเซล
  - แปลงรูปภาพเป็น tensor (อาร์เรย์ของตัวเลขที่โมเดลสามารถประมวลผลได้)
- กำหนด Dataset Class
  - คลาส TreesDataset ใช้สำหรับโหลดรูปภาพจากไดเรกทอรีและตารางข้อมูลที่มีข้อมูลของรูปภาพ
  - สำหรับแต่ละรูปภาพ, จะถูกโหลดและแปลงตามที่ระบุด้วย transformation
- ตั้งค่า Device สำหรับการคำนวณของโมเดล
  - ตรวจสอบว่า GPU พร้อมใช้งานหรือไม่ ถ้าไม่จะใช้ CPU
- โหลดโมเดล
  - โหลดโมเดล ResNet50
  - เปลี่ยนชั้นท้ายสุดของโมเดลเพื่อคาดการณ์ค่าเอาต์พุตเป็นตัวเลขเดียว
  - โหลด weight model จากไฟล์และเตรียมโมเดลให้พร้อมใช้งาน
- โหลดข้อมูลทดสอบ
  - โหลดข้อมูลจากไฟล์ CSV และสร้าง dataset และ dataloader สำหรับการทดสอบ
- การทำนายและการบันทึกผลลัพธ์
  - วนลูปผ่าน test loader และทำนายจำนวนต้นไม้สำหรับแต่ละรูปภาพ
  - ผลลัพธ์จะถูกบันทึกใน DataFrame และสุดท้ายบันทึกเป็นไฟล์ CSV
- การแจ้งเสร็จสิ้นการทำงาน



```

import pandas as pd
import torch
from torch.utils.data import DataLoader
from torchvision import transforms, models
from PIL import Image
import os

# Define the transformation used during training
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
])

# Define the dataset class
class TreesDataset(torch.utils.data.Dataset):
    def __init__(self, dataframe, image_dir, transform=None):
        self.dataframe = dataframe
        self.image_dir = image_dir
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        img_name = os.path.join(self.image_dir, self.dataframe.iloc[idx, 0])
        image = Image.open(img_name).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image

# Set device for model computations
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load the model
model = models.resnet50(pretrained=False)
num_ftrs = model.fc.in_features
model.fc = torch.nn.Linear(num_ftrs, 1)
model.load_state_dict(torch.load('model_weights.pth'))
model.to(device)
model.eval()

# Load test data
test_df = pd.read_csv('Test.csv')

```

```
test_dataset = TreesDataset(dataframe=test_df, image_dir='images', transform=transform)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)

# Prepare a DataFrame to store the results
results_df = pd.DataFrame(columns=['ImageId', 'TreeCount'])

# Predict the number of trees for each image in the test dataset
with torch.no_grad():
    for images in test_loader:
        images = images.to(device)
        output = model(images)
        tree_count = output.squeeze().item() # Convert to a scalar
        image_id = test_df.iloc[len(results_df)]['ImageId'] # Get the corresponding ImageId
        new_row = pd.DataFrame({'ImageId': [image_id], 'TreeCount': [tree_count]})
        results_df = pd.concat([results_df, new_row], ignore_index=True)

# Save the results to a CSV file
results_df.to_csv('tree_counts.csv', index=False)

print("Finished predicting tree counts. Results saved to tree_counts.csv.")
```



## ผลการทดลอง

<p>Output</p>	
<p>จำนวนข้อมูลในการทำนาย</p>	<p>859</p>
<p>ตัวอย่างจำนวนต้นไม้ที่นับได้</p>	<div data-bbox="691 1020 1229 1579"> <p>Predicted Tree Count: 11.036566734313965</p>  </div>

## สรุปผลการทดลอง

ข้อดี
<ul style="list-style-type: none"><li>- ประสิทธิภาพสูง: ResNet เป็นหนึ่งในโมเดลที่มีประสิทธิภาพสูงสำหรับการจดจำภาพ เนื่องจากการใช้งานลักษณะ "residual learning" ที่ช่วยให้โมเดลเรียนรู้ได้ดียิ่งขึ้นแม้ว่าจะมีชั้นที่ลึกขึ้น</li><li>- การเรียนรู้คุณสมบัติระดับสูง: โมเดล ResNet สามารถระบุและเรียนรู้คุณสมบัติที่ซับซ้อนในรูปภาพได้ ซึ่งช่วยให้สามารถทำนายจำนวนต้นไม้ในรูปภาพได้อย่างแม่นยำ</li><li>- ความยืดหยุ่นในการประยุกต์ใช้: โมเดลนี้สามารถปรับใช้กับข้อมูลรูปภาพหลากหลายประเภท</li></ul>
ข้อเสีย
<ul style="list-style-type: none"><li>- ข้อจำกัดในการทำนาย: หากโมเดลถูกฝึกฝนกับชนิดของต้นไม้เฉพาะเช่นต้นปาล์ม มันอาจไม่สามารถทำนายจำนวนของชนิดต้นไม้อื่นๆได้ดีเท่าที่ควร ซึ่งเป็นผลมาจากการขาดความหลากหลายในข้อมูลที่ใช้ฝึก</li><li>- Overfitting: ถ้าโมเดลฝึกกับข้อมูลจำกัด อาจจะเรียนรู้รายละเอียดและเฉพาะเจาะจงของชุดข้อมูลนั้นมากเกินไปจนไม่สามารถทำนายข้อมูลใหม่หรือที่แตกต่างออกไปได้ดี</li><li>- ความต้องการทรัพยากรสำหรับการ train: โมเดลที่มีความซับซ้อนเช่น ResNet ต้องการทรัพยากรการคำนวณจำนวนมาก ซึ่งอาจไม่เหมาะสมหากมีทรัพยากรไม่พร้อม</li></ul>

การใช้โมเดล ResNet สำหรับการนับต้นไม้ให้ผลลัพธ์ที่ดีในแง่ของความแม่นยำและประสิทธิภาพ แต่จำเป็นต้องมีการปรับปรุงโมเดลอย่างต่อเนื่องเพื่อให้สามารถทำนายได้อย่างถูกต้องในสถานการณ์ที่แตกต่างกันและหลีกเลี่ยงปัญหาเช่น Overfitting และข้อจำกัดในการทำนาย