โมเดลดึงรูปภาพที่มีหน้าบุคคล ที่เราต้องการจากคลังรูปภาพ

จัดทำโดย
ธนภัทร จำเริญ 6510503417
ภัทรพล ณ เนตร 6510503654

อาจารย์ผู้สอน
ผศ.ดร.ภารุจ รัตนวรพันธุ์

204466 การเรียนรู้เชิงลึก (Deep Learning)

ที่มาและความสำคัญ

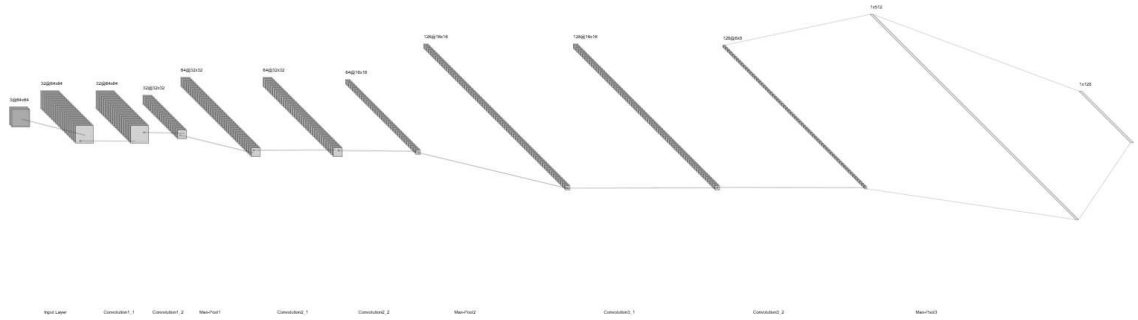
การดำเนินการค้นหาและคัดแยกภาพถ่ายที่ต้องการจากคลังภาพที่มีขนาดใหญ่ ซึ่งประกอบไปด้วยภาพถ่ายจำนวนเป็นพันเป็นหมื่นภาพนั้น ถือเป็นภารกิจที่ก่อให้เกิดความยากลำบากอย่างยิ่งและใช้ระยะเวลาดำเนินการสูง แม้ว่าภาพประเภทอื่น เช่น ภาพถ่ายทิวทัศน์หรือภาพวาดการ์ตูนอาจสามารถแยกแยะได้โดยง่ายเนื่องจากความแตกต่างของลักษณะทางภาพที่เด่นชัด แต่สำหรับภาพถ่ายที่ประกอบไปด้วยบุคคลจริงซึ่งเป็นภาพที่มักมีความคล้ายคลึงกันสูง กลับเป็นเรื่องท้าทายอย่างมาก ผู้ใช้งานจำเป็นต้องใช้วิธีการตรวจสอบและคัดเลือกด้วยสายตาแบบทีละภาพ เพื่อค้นหาและดึงภาพถ่ายที่มีบุคคลคนหนึ่งที่เราต้องการ เช่น ภาพถ่ายของตัวเราเองหรือคนในครอบครัว ซึ่งกระบวนการนี้อาจหมายถึงการต้องไล่ดูภาพถ่ายจำนวนนับหมื่นภาพอย่างต่อเนื่องเพื่อระบุตัวตนที่ถูกต้อง

ด้วยเหตุนี้ ทางกลุ่มของข้าพเจ้าจึงมีความสนใจในการพัฒนา โมเดลปัญญาประดิษฐ์ที่มีความสามารถตรวจรูปภาพที่มีใบหน้าของบุคคลที่เราต้องการจากคลังรูปภาพขนาดใหญ่โดยอัตโนมัติ เพื่อใช้เป็นเครื่องมือหลักในการอำนวยความสะดวกและเพิ่มประสิทธิภาพในการค้นหาภาพถ่ายได้อย่างรวดเร็วและแม่นยำ

แนวคิดการแก้ปัญหา

ใช้สถาปัตยกรรม Convolutional Neural Networks หรือ CNNs ใน Deep Learning มาแก้ปัญหา เนื่องจากมันถูกออกแบบมาเพื่อจัดการกับความซับซ้อนเหล่านี้โดยเฉพาะ มีโครงสร้างเครือข่ายประสาทหลายชั้นที่สามารถทำ Feature Extraction ที่สำคัญของใบหน้าได้อัตโนมัติด้วยตัวของมันเอง ซึ่งดีกว่า Machine Learning แบบดั้งเดิมเช่น SVM,PCA หรือ Eigenfaces ที่นักพัฒนาต้องกำหนดและสกัดลักษณะเด่นของใบหน้า เช่น ระยะห่างระหว่างตาหรือตำแหน่งของปาก ก่อนนำเข้าโมเดล อีกทั้งมีประสิทธิภาพลดลงเมื่อมีปัจจัยรบกวน เช่น แสงที่ต่างไปจากภาพที่ใช้ฝึกสอน ซึ่งแตกต่างจาก CNNs

ออกแบบโครงสร้างสถาปัตยกรรม CNNs



เริ่มจาก Input Layer ตามด้วย Convolution + pool 3 บล็อก แต่ละบล็อกมีการทำ Convolution 2 ครั้ง โดยใช้ฟิลเตอร์เล็กขนาด 3 x 3 สองครั้งติดกัน เพราะมีประสิทธิภาพเทียบเท่ากับการใช้ฟิลเตอร์ขนาดใหญ่ 5 x 5 เพียงครั้งเดียว แต่มีข้อดีกว่าคือ ช่วยลดจำนวนพารามิเตอร์ที่ต้องฝึกสอน ทำให้โมเดลเบาลงและลดโอกาสเกิด Overfitting อีกทั้งเพิ่ม Non-linearity เพราะใน CNNs นี้มีการใส่ Activation Function (ReLU) คั่นกลางระหว่าง Conv ทั้งสองทำให้เครือข่ายมีขีดความสามารถในการจำลองความสัมพันธ์ที่ซับซ้อนมากขึ้น ในแต่ละบล็อก จะมีการเพิ่ม Depth เป็นสองเท่าและลดมิติเชิงพื้นที่ด้วย Max Pooling 2 x 2 ก่อนนำไป Flatten ข้อมูลจาก 2D เป็น 1D เพื่อเข้า Fully Connected Layer และ Output Layer

อธิบายตามลำดับชั้น

1. Input Layer รับภาพขนาด 64 x 64
2. Block 1 ทำ Convolution 2 ครั้ง ใช้ 32 ฟิลเตอร์ วัตถุประสงค์คือการสกัดคุณสมบัติพื้นฐานเช่น ขอบ หลังจากนั้น Max Pooling ลดขนาดภาพลงครึ่งหนึ่งเหลือ 32 x 32
3. Block 2 ทำ Convolution 2 ครั้ง ใช้ 64 ฟิลเตอร์ สกัดคุณสมบัติระดับกลาง และเพิ่มมิติเชิงลึกเป็นสองเท่า หลังจากนั้น Max Pooling ลดขนาดภาพลงครึ่งหนึ่งเหลือ 16 x 16
4. Block 3 ทำ Convolution 2 ครั้ง ใช้ 128 ฟิลเตอร์ สกัดคุณสมบัติระดับสูง หลังจากนั้น Max Pooling ลดขนาดภาพลงครึ่งหนึ่งเหลือ 8 x 8
5. Flatten: แปลง Feature Map 2 มิติ ให้เป็นเวกเตอร์ 1 มิติความยาว 128 x 8 x 8
6. Fully Connected Layer ย่อขนาดเวกเตอร์คุณสมบัติลงเหลือ 512 มิติ
7. Output Layer สร้างเวกเตอร์สุดท้ายที่มีขนาดเท่า embedding dimension โดยกำหนดเป็น 128

หลักการของ Triplet Loss ในการพัฒนาโมเดล

ในการพัฒนาโมเดลนี้มีการใช้ Triplet Loss เพื่อฝึกโมเดลให้เรียนรู้ที่จะสร้างเวกเตอร์ฝังตัว (Embedding Vector) ที่มีคุณสมบัติเฉพาะ เวกเตอร์ของบุคคลเดียวกัน ควรอยู่ใกล้กันในพื้นที่ Embedding (Euclidean Space) เวกเตอร์ของบุคคลต่างกัน ควรอยู่ห่างกันในพื้นที่ Embedding ในการฝึกด้วย Triplet Loss แต่ละตัวอย่างจะต้องประกอบด้วยรูปภาพ 3 รูป หรือที่เรียกว่า Triplet ประกอบด้วย

1. Anchor (A) : รูปภาพต้นแบบ
2. Positive (P): รูปภาพของ บุคคลเดียวกันกับ Anchor
3. Negative (N) : รูปภาพของ บุคคลอื่น ที่ต่างจาก Anchor

เป้าหมายของ Triplet Loss คือการบังคับให้ ระยะห่างระหว่าง A กับ P มีค่าน้อยกว่า ระยะห่างระหว่าง A กับ N อยู่ในจำนวนที่กำหนด เรียกว่า Margin ($\propto$) โดยสูตร Triplet Loss คือ

$$L(A,P,N) = \max(0,d(A,P)-d(A,N) + \propto)$$

d(A,P)คือระยะห่างระหว่าง Anchor และ Positive
d(A,N)คือระยะห่างระหว่าง Anchor และ Positive
Margin ($\propto$) คือระยะห่างขั้นต่ำที่ต้องการให้ d(A,P) ห่างจาก d(A,N)

ดังนั้นแล้วในโมเดลจึงจะต้องมีอย่างต่ำ 2 คลาสในการพัฒนาเพื่อใช้งาน Triplet Loss โดยทางผู้จัดทำจะใช้ Dataset ของหน้าดาราที่ยืมมาจาก Kaggle ทั้งหมด 4 คลาสอย่างละ 80 ภาพเพื่อฝึกและทดสอบโมเดล

อธิบายโค้ดในการพัฒนาโมเดล

อธิบายเรียงลำดับตาม Code Cell ใน Colab เข้าดูโค้ดด้วย ลิงค์ Github Repository : https://github.com/ProgrammerTon/Facial-Image-Search-Retrieval

1. Setup & Imports

```python
from google.colab import drive
import os
import cv2
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, Dataset, random_split
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from tqdm import tqdm
import random
import math
import glob
import shutil

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

ติดตั้งไลบรารีที่จำเป็น, ตั้งค่าสภาพแวดล้อม, และนำเข้าโมดูลที่ใช้ในโปรเจกต์ทั้งหมด และเนื่องจากทำผ่าน Colab และดึงข้อมูลเทรนจากใน Google Drive จึงมีการ Mount Drive

2. Paths & Hyperparameters

```python
BASE_PROJECT_DIR = "/content/drive/MyDrive/DeepLearnProject"

DATA_ROOT = os.path.join(BASE_PROJECT_DIR, "content")
PROCESSED_DIR = os.path.join(BASE_PROJECT_DIR, "processed_faces")
TEST_IMAGES_DIR = os.path.join(BASE_PROJECT_DIR, "test_images")

# Hyperparameters
INPUT_SIZE = (64, 64)
BATCH_SIZE = 16
LEARNING_RATE = 1e-4
NUM_EPOCHS = 20
EMBEDDING_DIM = 128
TRIPLET_MARGIN = 0.2
RECOGNITION_THRESHOLD = 0.5
```

ส่วนแรก Path เชื่อมไปยัง Directory ของโฟลเดอร์ต่างๆ ในไดรฟ์ประกอบด้วยโฟลเดอร์ content ซึ่งเป็น Dataset รวมหน้าคนที่ใช้สำหรับการเทรนประกอบ

ด้วยหน้า RobertDowneyJr, Billie Ellish, Alexandra Daddario และ Bradd Pitt อย่างละ 80 ภาพ ต่อมาโฟลเดอร์ processed_faces รวมหน้าคนที่ผ่านการตรวจสอบใบหน้าจาก MTCNN แล้วว่าสามารถนำมาเทรนได้ และสุดท้าย test_images รวมภาพไว้ทำการทดสอบประสิทธิภาพโมเดล

ส่วนสองคือการกำหนด Hyperparameters สำหรับการฝึก

- INPUT_SIZE = (64,64) กำหนดขนาดภาพในการฝึกเป็น 64x64 Pixels
- BATCH_SIZE = 16 กำหนดขนาด Batch ในการฝึกเป็นรอบละ 16
- LEARNING_RATE = 1e-4 กำหนดอัตราการเรียนรู้โมเดล 1e-4
- NUM_EPOCHS = 20 กำหนดรอบการฝึกเป็น 20
- EMBEDDING_DIM = 128 กำหนด ขนาดของเวกเตอร์ฝังตัว (Embedding Vector) ที่โมเดลจะสร้างออกมาตอนจบ ตามที่แสดงไว้ในโครงสร้าง CNN เป็น 128
- TRIPLET_MARGIN กำหนด ค่า Margin สำหรับ Triplet Loss เพื่อกำหนดระยะห่างขั้นต่ำระหว่าง (A, P) กับ (A, N) ซึ่งกำหนดเป็น 0.2
- RECOGNITION_THRESHOLD กำหนดระยะทางที่ต่ำที่สุดเพื่อจำแนกว่าภาพบุคคลนี้อยู่ในคลาสที่ฝึกหรือเป็น Unknown ซึ่งตั้งไว้ที่ 0.5

3. MTCNN

```
!pip install mtcnn
from mtcnn.mtcnn import MTCNN

detector = MTCNN()
print("✅ MTCNN detector initialized.")
```

```
Requirement already satisfied: mtcnn in /usr/local/lib/python3.12/dist-packages (1.0.0)
Requirement already satisfied: joblib>=1.4.2 in /usr/local/lib/python3.12/dist-packages (from mtcnn) (1.5.2)
Requirement already satisfied: lz4>=4.3.3 in /usr/local/lib/python3.12/dist-packages (from mtcnn) (4.4.4)
✅ MTCNN detector initialized.
```

นำโมเดล MTCNN เข้ามาใช้งานซึ่งถือว่าเป็นโมเดลสำเร็จรูปที่ไม่ได้สร้างเองแต่ใช้เพื่อช่วยตรวจจับใบหน้าบุคคลในภาพ ส่วนโมเดลในการจำแนกภาพบุคคลจะถูกพัฒนาเองโดยทางผู้จัดทำ

4. Data Preprocessing

```
Starting MTCNN face detection (Min Confidence: 0.9)...
Processing folder: BraddPitt
  -> BraddPitt: 100%|██████████| 80/80 [00:12<00:00,  6.58it/s]
  -> Processed 80 images, found 80 faces.
Processing folder: AlexandraDaddario
  -> AlexandraDaddario: 100%|██████████| 80/80 [00:12<00:00,  6.44it/s]
  -> Processed 80 images, found 80 faces.
Processing folder: RobertDowneyJR
  -> RobertDowneyJR: 100%|██████████| 80/80 [00:12<00:00,  6.44it/s]
  -> Processed 80 images, found 79 faces.
Processing folder: BillieEllish
  -> BillieEllish: 100%|██████████| 80/80 [01:13<00:00,  1.09it/s]  -> Processed 80 images, found 78 faces.

✅ Face preparation complete (MTCNN).
Total images scanned: 320
Total faces extracted: 317
```

```python
def prepare_face_data_mtcnn():
    os.makedirs(PROCESSED_DIR, exist_ok=True)

    total_images_processed = 0
    total_faces_found = 0
    min_confidence = 0.9 #

    print(f"Starting MTCNN face detection (Min Confidence: {min_confidence})...")

    for person in os.listdir(DATA_ROOT):
        person_path = os.path.join(DATA_ROOT, person)
        if not os.path.isdir(person_path):
            continue

        save_dir = os.path.join(PROCESSED_DIR, person)
        os.makedirs(save_dir, exist_ok=True)

        print(f"Processing folder: {person}")
        person_image_count = 0
        person_face_count = 0

        for img_name in tqdm(os.listdir(person_path), desc=f"  -> {person}"):
            img_path = os.path.join(person_path, img_name)
            try:
                img_bgr = cv2.imread(img_path)
                if img_bgr is None: continue
                img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
            except Exception as e:
                print(f"Error reading {img_name}: {e}")
                continue

            total_images_processed += 1
            person_image_count += 1
            faces = detector.detect_faces(img_rgb)

            if len(faces) > 0:
                high_conf_faces = [f['box'] for f in faces if f['confidence'] >= min_confidence]

                if high_conf_faces:
                    (x, y, w, h) = max(high_conf_faces, key=lambda rect: rect[2] * rect[3])
                    x, y = abs(x), abs(y)
                    face_crop = img_bgr[y:y+h, x:x+w]

                    if face_crop.size == 0: continue

                    face_resized = cv2.resize(face_crop, IMPUT_SIZE)
                    save_path = os.path.join(save_dir, img_name)
                    cv2.imwrite(save_path, face_resized)

                    total_faces_found += 1
                    person_face_count += 1

        print(f"  -> Processed {person_image_count} images, found {person_face_count} faces.")

    print("\n Face preparation complete (MTCNN).")
    print(f"Total images scanned: {total_images_processed}")
    print(f"Total faces extracted: {total_faces_found}")

prepare_face_data_mtcnn()
```

ใช้ MTCNN เพื่อตรวจจับและตัดส่วนใบหน้าออกจากรูปภาพในโฟลเดอร์ Content และแปลงรูปให้อยู่ในขนาด 64x64 แล้วใส่ในโฟลเดอร์ processed_faces ก่อนที่จะนำไปใช้เทรนในขั้นตอนต่อไป

5. Model Definition

```python
class FaceRecogCNN(nn.Module):
    def __init__(self, embedding_dim):
        super(FaceRecogCNN, self).__init__()
        # Block 1: 64x64 -> 32x32
        self.conv1_1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv1_2 = nn.Conv2d(32, 32, 3, padding=1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.bn1 = nn.BatchNorm2d(32)

        # Block 2: 32x32 -> 16x16
        self.conv2_1 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv2_2 = nn.Conv2d(64, 64, 3, padding=1)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.bn2 = nn.BatchNorm2d(64)

        # Block 3: 16x16 -> 8x8
        self.conv3_1 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv3_2 = nn.Conv2d(128, 128, 3, padding=1)
        self.pool3 = nn.MaxPool2d(2, 2)
        self.bn3 = nn.BatchNorm2d(128)

        # FC layers
        # ขนาด Input = 128 (channels) * 8 * 8 (ขนาดภาพหลัง Pool 3 ครั้ง)
        self.fc1 = nn.Linear(128 * 8 * 8, 512)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, embedding_dim)

    def forward(self, x):
        # Block 1
        x = F.relu(self.conv1_1(x))
        x = self.bn1(self.pool1(F.relu(self.conv1_2(x))))

        # Block 2
        x = F.relu(self.conv2_1(x))
        x = self.bn2(self.pool2(F.relu(self.conv2_2(x))))

        # Block 3
        x = F.relu(self.conv3_1(x))
        x = self.bn3(self.pool3(F.relu(self.conv3_2(x))))

        x = x.view(x.size(0), -1) # Flatten
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)

        x = F.normalize(x, p=2, dim=1) # Normalize embedding
        return x

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = FaceRecogCNN(EMBEDDING_DIM).to(device)
print(f"Model V2 (Deeper) loaded to {device}")
```

เขียนโค้ดสร้างโมเดลตามที่ได้ออกแบบไว้ โดยเพิ่มเติมจากที่อธิบายในโครงสร้าง มีการใช้ Dropout ที่ 0.5 เพื่อป้องกันการ Overfitting

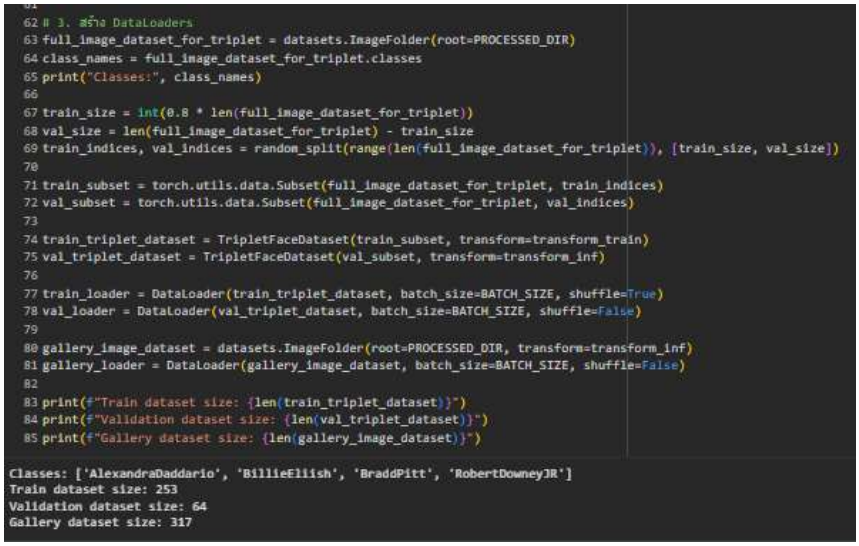
6. Transforms & Triplet Dataset

```python
transform_train = transforms.Compose([
    transforms.Resize(INPUT_SIZE),
    transforms.RandomHorizontalFlip(p=0.5),      # พลิกซ้ายขวา (ดี)
    transforms.RandomRotation(10),               # หมุนแค่ 10 องศา (พอประมาณ)
    transforms.ColorJitter(brightness=0.2, contrast=0.2), # ปรับแสงสีเล็กน้อย
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

transform_inf = transforms.Compose([
    transforms.Resize(INPUT_SIZE),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

```

เตรียมข้อมูลภาพให้พร้อมก่อนเข้าสู่ Triplet Dataset แบ่งเป็น 2 ส่วน
- transform_train ใช้เพื่อเพิ่มความหลากหลายของข้อมูลและทำให้โมเดลไม่เกิดอาการ Overfitting กับข้อมูลชุดเดิม ทำให้โมเดลทนทานต่อการเปลี่ยนแปลงสภาพแวดล้อม เช่น แสง, มุม, การหันหน้า
- transform_inf ใช้เพื่อเตรียมรูปภาพให้อยู่ในขนาดที่ตั้งไว้ ก่อนเข้าสู่โมเดลในขั้นตอนการทดสอบหรือใช้งานจริง

```python
# 2. สร้าง TripletFaceDataset (รองรับ Subset)
class TripletFaceDataset(Dataset):
    def __init__(self, subset, transform=None):
        self.subset = subset
        self.transform = transform
        self.full_dataset = self.subset.dataset
        self.all_labels = np.array(self.full_dataset.targets)
        self.indices = self.subset.indices
        self.labels = self.all_labels[self.indices]
        self.labels_set = set(self.labels)
        self.label_to_indices_map = {label: np.where(self.labels == label)[0]
                                     for label in self.labels_set}

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

    def __getitem__(self, index):
        anchor_label = self.labels[index]
        anchor_true_index = self.indices[index]
        anchor_img, _ = self.full_dataset[anchor_true_index]

        positive_subset_index = index
        while positive_subset_index == index:
            positive_subset_index = np.random.choice(self.label_to_indices_map[anchor_label])
        positive_true_index = self.indices[positive_subset_index]
        positive_img, _ = self.full_dataset[positive_true_index]

        possible_negative_labels = list(self.labels_set - {anchor_label})

        if not possible_negative_labels:
            negative_img = positive_img
        else:
            negative_label = np.random.choice(possible_negative_labels)
            negative_subset_index = np.random.choice(self.label_to_indices_map[negative_label])
            negative_true_index = self.indices[negative_subset_index]
            negative_img, _ = self.full_dataset[negative_true_index]

        if self.transform:
            anchor_img = self.transform(anchor_img)
            positive_img = self.transform(positive_img)
            negative_img = self.transform(negative_img)

        return anchor_img, positive_img, negative_img

```

คลาส TripletFaceDataset ทำหน้าที่สร้าง Triplet (A,P,N) สำหรับทุกครั้งที่มีการเรียกข้อมูลจาก DataLoader โดยจะสุ่มรูปภาพ 3 รูปตามหลักการ Triplet Loss ประกอบด้วย ภาพต้นแบบ ภาพเหมือนและภาพไม่เหมือนจากคลาสต่างๆ เพื่อเรียนรู้หลักการคำนวณระยะห่างของแต่ละคลาส



```python
# 3. สร้าง DataLoaders
full_image_dataset_for_triplet = datasets.ImageFolder(root=PROCESSED_DIR)
class_names = full_image_dataset_for_triplet.classes
print("Classes:", class_names)

train_size = int(0.8 * len(full_image_dataset_for_triplet))
val_size = len(full_image_dataset_for_triplet) - train_size
train_indices, val_indices = random_split(range(len(full_image_dataset_for_triplet)), [train_size, val_size])

train_subset = torch.utils.data.Subset(full_image_dataset_for_triplet, train_indices)
val_subset = torch.utils.data.Subset(full_image_dataset_for_triplet, val_indices)

train_triplet_dataset = TripletFaceDataset(train_subset, transform=transform_train)
val_triplet_dataset = TripletFaceDataset(val_subset, transform=transform_inf)

train_loader = DataLoader(train_triplet_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_triplet_dataset, batch_size=BATCH_SIZE, shuffle=False)

gallery_image_dataset = datasets.ImageFolder(root=PROCESSED_DIR, transform=transform_inf)
gallery_loader = DataLoader(gallery_image_dataset, batch_size=BATCH_SIZE, shuffle=False)

print(f"Train dataset size: {len(train_triplet_dataset)}")
print(f"Validation dataset size: {len(val_triplet_dataset)}")
print(f"Gallery dataset size: {len(gallery_image_dataset)}")
```

```
Classes: ['AlexandraDaddario', 'BillieEllish', 'BraddPitt', 'RobertDowneyJR']
Train dataset size: 253
Validation dataset size: 64
Gallery dataset size: 317
```

สร้าง DataLoaders แบ่งชุดข้อมูลเป็น training_set กับ validation_set สำหรับการฝึกโมเดลในอัตราส่วน 80/20 จากข้อมูลทั้งหมด

7. Training Function (Triplet Loss)

```python
def train_model(model, train_loader, val_loader, num_epochs, lr, margin):
    criterion = nn.TripletMarginLoss(margin=margin)
    optimizer = optim.Adam(model.parameters(), lr=lr)

    train_losses, val_losses = [], []
    best_val_loss = float('inf')

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        train_loader_tqdm = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs} [Train]")
        for anchor_img, positive_img, negative_img in train_loader_tqdm:
            anchor_img = anchor_img.to(device)
            positive_img = positive_img.to(device)
            negative_img = negative_img.to(device)

            optimizer.zero_grad()

            emb_a = model(anchor_img)
            emb_p = model(positive_img)
            emb_n = model(negative_img)

            loss = criterion(emb_a, emb_p, emb_n)

            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            train_loader_tqdm.set_postfix(Loss=running_loss/len(train_loader_tqdm))

        train_loss = running_loss / len(train_loader)
        train_losses.append(train_loss)

        model.eval()
        val_loss = 0.0
        val_loader_tqdm = tqdm(val_loader, desc=f"Epoch {epoch+1}/{num_epochs} [Val]")
        with torch.no_grad():
            for anchor_img, positive_img, negative_img in val_loader_tqdm:
                anchor_img = anchor_img.to(device)
                positive_img = positive_img.to(device)
                negative_img = negative_img.to(device)

                emb_a = model(anchor_img)
                emb_p = model(positive_img)
                emb_n = model(negative_img)

                loss = criterion(emb_a, emb_p, emb_n)
                val_loss += loss.item()
                val_loader_tqdm.set_postfix(Loss=val_loss/len(val_loader_tqdm))

        val_loss /= len(val_loader)
        val_losses.append(val_loss)

        print(f"Epoch [{epoch+1}/{num_epochs}] Train Loss: {train_loss:.4f} | Val Loss: {val_loss:.4f}")

        # บันทึกโมเดลที่ดีที่สุด (วัดจาก Val Loss)
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            torch.save(model.state_dict(), MODEL_SAVE_PATH)
            print(f"  -> New best model saved to {MODEL_SAVE_PATH} (Val Loss: {val_loss:.4f})")


    print("✅ Training complete.")
```

โค้ดส่วนนี้ เป็นส่วนที่สำคัญ จึงมีการอธิบายการทำงานอย่างละเอียดตามลำดับบรรทัดดังนี้

บรรทัดโค้ด	การทำงาน
def train_model(model, train_loader, val_loader, num_epochs, lr, margin)	กำหนดฟังก์ชัน: รับพารามิเตอร์ที่จำเป็นในการฝึก ได้แก่ model (โครงสร้าง CNN), train_loader, val_loader, จำนวน num_epochs, lr (Learning Rate), และ margin สำหรับ Triplet Loss

criterion = nn.TripletMarginLoss(margin=margin)	กำหนด Loss Function โดยใช้ค่า margin ที่ส่งเข้ามา ฟังก์ชันนี้จะคำนวณความสูญเสียตามสูตรที่อธิบายในหลักการ Triplet Loss
optimizer = optim.Adam(model.parameters(), lr=lr)	ใช้ Adam ซึ่งเป็นอัลกอริทึมที่นิยมในการปรับน้ำหนักของโมเดลตาม learning rate ที่กำหนด
train_losses, val_losses = [], [] best_val_loss = float('inf')	สร้าง List เปล่าเพื่อเก็บค่า Loss และกำหนด best_val_loss ให้เป็นค่า Infinity เพื่อใช้ในการติดตามและบันทึกโมเดลที่มีประสิทธิภาพที่สุด
for epoch in range(num_epochs):	เริ่มวนซ้ำตามจำนวน Epochs ที่กำหนด
model.train()	ตั้งค่าโมเดลให้อยู่ในโหมด Training ซึ่งจะเปิดใช้งาน Layer ที่ทำงานแตกต่างกันระหว่างการฝึกและการทดสอบ เช่น Dropout
running_loss = 0.0	รีเซ็ต Loss ตอนขึ้น epoch ใหม่
train_loader_tqdm = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs} [Train]")	ใช้แสดง Progress Bar ระหว่างฝึก
for anchor_img, positive_img, negative_img in train_loader_tqdm: anchor_img = anchor_img.to(device) positive_img = positive_img.to(device) negative_img = negative_img.to(device)	วนซ้ำ Batch เริ่ม Loop เพื่อดึงข้อมูล Triplet (A, P, N) ทีละ Batch จาก DataLoader ย้าย Tensor ของรูปภาพทั้งสามไปทำงานบนอุปกรณ์ที่ใช้งานอยู่
optimizer.zero_grad()	ลบค่า Gradient เก่าทั้งหมดที่คำนวณไว้ก่อนหน้า เพื่อไม่ให้เกิดการสะสมข้าม Batch

emb_a = model(anchor_img) emb_p = model(positive_img) emb_n = model(negative_img)	Forward Pass: ส่งรูปภาพทั้งสามผ่านโมเดล เพื่อสร้าง Embedding Vector ทั้งสามตัว
loss = criterion(emb_a, emb_p, emb_n)	ส่ง Embeddings ทั้งสามเข้าสู่ Triplet Loss Function เพื่อคำนวณค่าความสูญเสียตามเป้าหมาย
loss.backward()	Backpropagation คำนวณ Gradient ย้อนกลับของ Loss เทียบกับน้ำหนักทั้งหมดของโมเดล
optimizer.step()	ปรับปรุงน้ำหนักของโมเดลโดยใช้ Optimizer และ Gradient ที่คำนวณได้
running_loss += loss.item() train_loader_tqdm.set_postfix(Loss=running_loss/len(train_loader_tqdm))	สะสมค่า Loss และอัปเดตแถบความคืบหน้าด้วยค่าเฉลี่ย Loss ปัจจุบันของ Batch
train_loss = running_loss / len(train_loader) train_losses.append(train_loss)	คำนวณค่าเฉลี่ย Loss สุดท้ายของ Epoch สำหรับชุด Training
model.eval()	ตั้งค่าโมเดลให้อยู่ในโหมด Evaluation ปิด Dropout
val_loss = 0.0	รีเซ็ต Loss
val_loader_tqdm = tqdm(val_loader, desc=f"Epoch {epoch+1}/{num_epochs} [Val]")	แสดง Progress
with torch.no_grad():	ปิด Gradient Calculation เป็นคำสั่งสำคัญที่บอก PyTorch ว่าไม่ต้องคำนวณและจัดเก็บ Gradient ในระหว่าง Forward Pass สิ่งนี้ช่วย ลดการใช้หน่วยความจำ และ เพิ่มความเร็ว ในการทดสอบ

บรรทัด 38-49	ทำซ้ำขั้นตอนเดียวกับการฝึก (โหลด Triplet, Forward Pass, คำนวณ Loss) แต่ ไม่มี การเรียกใช้ optimizer.zero_grad(), loss.backward(), และ optimizer.step()
val_loss /= len(val_loader) val_losses.append(val_loss)	คำนวณค่าเฉลี่ย Loss สุดท้ายของ Epoch สำหรับชุด Validation
print(f"Epoch [{epoch+1}/{num_epochs}] Train Loss: {train_loss:.4f} | Val Loss: {val_loss:.4f}")	แสดงผลสรุป Loss ของ Training และ Validation ใน Epoch นั้น
if val_loss < best_val_loss: best_val_loss = val_loss torch.save(model.state_dict(), MODEL_SAVE_PATH) print(f" -> New best model saved to {MODEL_SAVE_PATH} (Val Loss: {val_loss:.4f})")	เทียบกับ Epoch ที่ได้ Loss น้อยที่สุด ถ้าได้ Loss น้อยกว่า Epoch นั้นให้ บันทึกโมเดลนั้นเป็นโมเดลที่มี ประสิทธิภาพที่สุด

หลังจากนั้นเขียนโค้ดเพื่อแสดง Loss Curve และจากการพัฒนาโมเดลนี้ ได้กราฟตามภาพ

8. Create Embedding Gallery

```python
def create_embedding_gallery(model, data_loader, class_names):
    model.eval()
    embeddings = {}

    with torch.no_grad():
        for images, labels in data_loader:
            images = images.to(device)
            embeds = model(images)

            for i in range(len(labels)):
                label_name = class_names[labels[i]]
                if label_name not in embeddings:
                    embeddings[label_name] = []
                embeddings[label_name].append(embeds[i])

    gallery = {}
    for label_name, embs_list in embeddings.items():
        gallery[label_name] = torch.mean(torch.stack(embs_list), dim=0)

    print(f"✅ Embedding gallery created for {list(gallery.keys())}")
    return gallery

# สร้าง Gallery
gallery = create_embedding_gallery(model, gallery_loader, class_names)
```

ใช้โมเดลที่ได้จากการเทรนมาสร้าง Embedding Gallery สำหรับใช้งานจริง เพื่อทดสอบกับ test_images

9. Inference Function Recognization

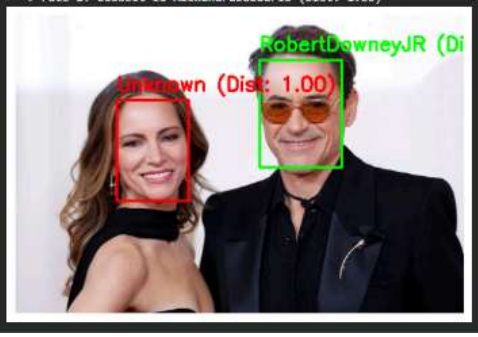
```python
def run_inference(image_path, model, gallery, transform_inf):
    model.eval()
    img_bgr = cv2.imread(image_path)
    if img_bgr is None:
        print(f"❌ Image not found at {image_path}")
        return None

    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    faces = detector.detect_faces(img_rgb)

    if len(faces) == 0:
        print(f"⚠️ No face detected in {os.path.basename(image_path)}.")
        return img_bgr

    print(f"Found {len(faces)} faces in {os.path.basename(image_path)}.")
    img_with_boxes = img_bgr.copy()

    for i, face in enumerate(faces):
        if face['confidence'] < 0.9:
            continue

        (x, y, w, h) = face['box']
        x, y = abs(x), abs(y)

        face_crop = img_bgr[y:y+h, x:x+w]
        if face_crop.size == 0: continue

        face_tensor = transform_inf(Image.fromarray(cv2.cvtColor(face_crop, cv2.COLOR_BGR2RGB))).unsqueeze(0).to(device)

        with torch.no_grad():
            test_emb = model(face_tensor)
            distances = []
            for person_name, person_emb in gallery.items():
                dist = F.pairwise_distance(test_emb, person_emb.unsqueeze(0))
                distances.append((person_name, dist.item()))

        distances.sort(key=lambda x: x[1])
        predicted_name = distances[0][0]
        min_dist = distances[0][1]

        if min_dist > RECOGNITION_THRESHOLD:
            display_name = "Unknown"
            color = (0, 0, 255)
        else:
            display_name = predicted_name
            color = (0, 255, 0)

        label = f"{display_name} (Dist: {min_dist:.2f})"

        cv2.rectangle(img_with_boxes, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img_with_boxes, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)

        print(f"  -> Face {i+1}: Closest is {predicted_name} (Dist: {min_dist:.2f})")

    return img_with_boxes
```

ฟังก์ชันสุดท้ายซึ่งใช้ในการตรวจจับใบหน้าใน test_images และนำรูปภาพส่วนใบหน้าไปเทียบกับข้อมูลใน Embedding Gallery ว่าใกล้เคียงกับคลาสไหนมากที่สุด แล้วส่งผลลัพธ์ที่ได้ออกมา แสดงสี่เหลี่ยมครอบใบหน้าบุคคลที่พบในภาพและจำแนกว่าอยู่ในคลาสที่เทรนไว้หรือไม่ พร้อมบอกระยะห่าง โดยตั้ง Threshold ไว้ที่ 0.5

10. Testing with 1 Image

```
os.makedirs(TEST_IMAGES_DIR, exist_ok=True)

TEST_IMAGE_PATH_SINGLE = os.path.join(TEST_IMAGES_DIR, "Test03.png") # ใส่ชื่อรูปที่ต้องการทดสอบ
print(f"--- Testing single image: {TEST_IMAGE_PATH_SINGLE} ---")

output_image = run_inference(TEST_IMAGE_PATH_SINGLE, model, gallery, transform_inf)

if output_image is not None:
    plt.imshow(cv2.cvtColor(output_image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
else:
    print("Cannot display image (file not found or no face detected).")
```

```
--- Testing single image: /content/drive/MyDrive/DeepLearnProject/test_images/Test03.png ---
Found 2 faces in Test03.png.
  -> Face 1: Closest is RobertDowneyJR (Dist: 0.49)
  -> Face 2: Closest is AlexandraDaddario (Dist: 1.00)
```

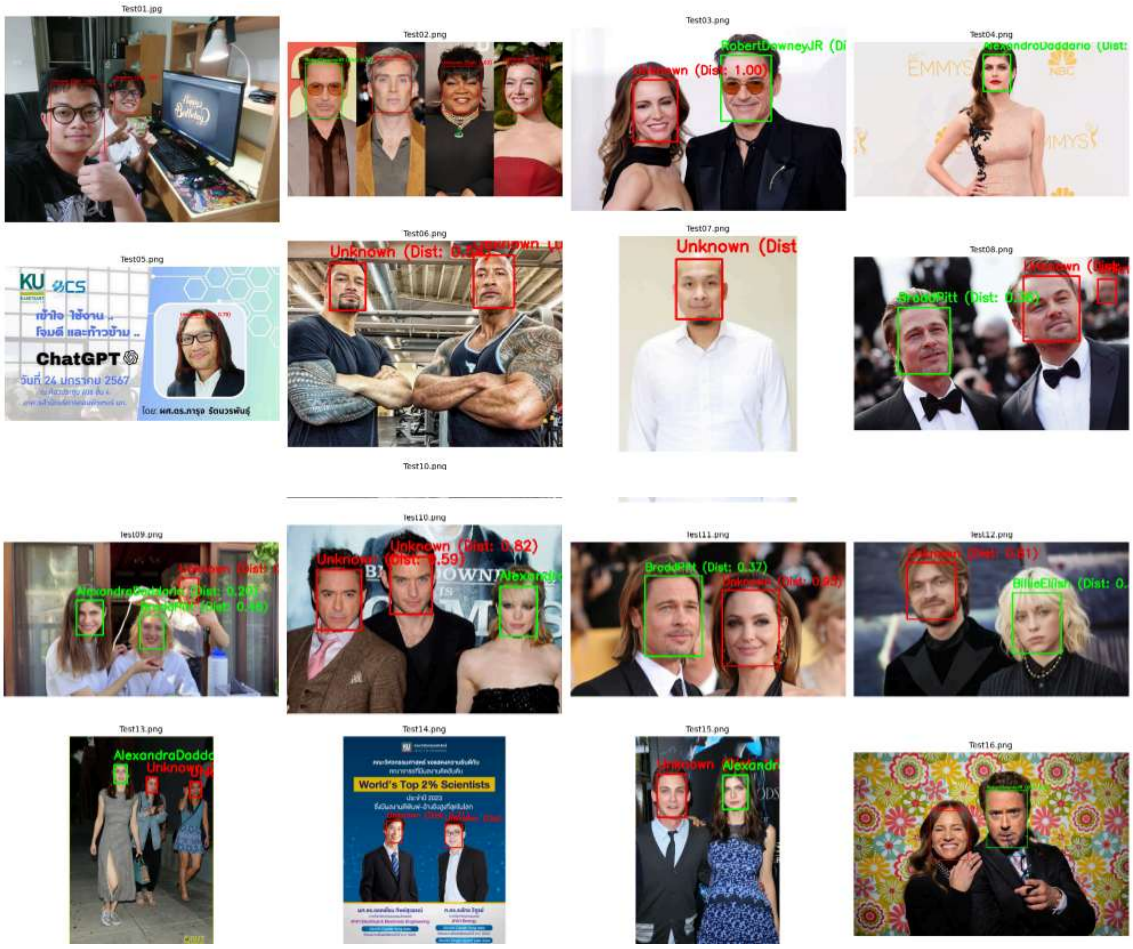


ทดสอบโมเดลด้วยภาพ 1 ภาพ โดยการส่งภาพที่ต้องการไปในฟังก์ชัน run_inference

11. Testing with Multiple Images

```
def run_batch_inference(test_images_folder, model, gallery, transform_inf):
    image_paths = glob.glob(os.path.join(test_images_folder, '*.*'))
    image_paths = [p for p in image_paths if p.lower().endswith(('.png', '.jpg', '.jpeg'))]
    image_paths.sort()

    if not image_paths:
        print(f"None")
        return

    num_images = len(image_paths)
    print(f"พบ {num_images} ภาพ, กำลังเริ่มทดสอบ...")

    cols = 4
    rows = math.ceil(num_images / cols)
    fig, axes = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))
    axes = axes.flatten()

    for i, image_path in enumerate(image_paths):
        output_image_bgr = run_inference(image_path, model, gallery, transform_inf)

        if output_image_bgr is not None:
            img_rgb = cv2.cvtColor(output_image_bgr, cv2.COLOR_BGR2RGB)
            axes[i].imshow(img_rgb)
            axes[i].set_title(os.path.basename(image_path), fontsize=10)

        axes[i].axis('off')

    for j in range(i + 1, len(axes)):
        axes[j].axis('off')

    plt.tight_layout()
    plt.show()

print(f"\n--- Testing all images (Threshold: {RECOGNITION_THRESHOLD}) ---")
os.makedirs(TEST_IMAGES_DIR, exist_ok=True)
run_batch_inference(TEST_IMAGES_DIR, model, gallery, transform_inf)
```

```
--- Testing all images (Threshold: 0.5) ---
พบ 16 ภาพ, กำลังเริ่มทดสอบ...
Found 2 faces in Test01.jpg.
  -> Face 1: Closest is AlexandraDaddario (Dist: 1.05)
  -> Face 2: Closest is BraddPitt (Dist: 0.68)
Found 4 faces in Test02.png.
  -> Face 1: Closest is BillieEilish (Dist: 0.63)
  -> Face 2: Closest is BillieEilish (Dist: 0.81)
  -> Face 3: Closest is AlexandraDaddario (Dist: 0.51)
  -> Face 4: Closest is RobertDowneyJR (Dist: 0.37)
Found 2 faces in Test03.png.
  -> Face 1: Closest is RobertDowneyJR (Dist: 0.49)
  -> Face 2: Closest is AlexandraDaddario (Dist: 1.00)
Found 1 faces in Test04.png.
  -> Face 1: Closest is AlexandraDaddario (Dist: 0.21)
Found 1 faces in Test05.png.
  -> Face 1: Closest is BillieEilish (Dist: 0.79)
Found 2 faces in Test06.png.
  -> Face 1: Closest is BillieEilish (Dist: 0.54)
  -> Face 2: Closest is BillieEilish (Dist: 0.89)
Found 1 faces in Test07.png.
  -> Face 1: Closest is BillieEilish (Dist: 0.86)
Found 3 faces in Test08.png.
  -> Face 1: Closest is BraddPitt (Dist: 0.61)
  -> Face 2: Closest is BraddPitt (Dist: 0.36)
  -> Face 3: Closest is BraddPitt (Dist: 0.59)
Found 3 faces in Test09.png.
  -> Face 1: Closest is AlexandraDaddario (Dist: 0.66)
  -> Face 2: Closest is BraddPitt (Dist: 0.46)
  -> Face 3: Closest is AlexandraDaddario (Dist: 0.20)
Found 3 faces in Test10.png.
  -> Face 1: Closest is RobertDowneyJR (Dist: 0.59)
  -> Face 2: Closest is RobertDowneyJR (Dist: 0.82)
  -> Face 3: Closest is AlexandraDaddario (Dist: 0.28)
Found 2 faces in Test11.png.
  -> Face 1: Closest is BraddPitt (Dist: 0.37)
  -> Face 2: Closest is BraddPitt (Dist: 0.93)
Found 2 faces in Test12.png.
  -> Face 1: Closest is BraddPitt (Dist: 0.81)
  -> Face 2: Closest is BillieEilish (Dist: 0.33)
Found 3 faces in Test13.png.
  -> Face 1: Closest is AlexandraDaddario (Dist: 0.21)
  -> Face 2: Closest is BraddPitt (Dist: 0.90)
  -> Face 3: Closest is BillieEilish (Dist: 0.99)
Found 2 faces in Test14.png.
  -> Face 1: Closest is AlexandraDaddario (Dist: 0.71)
  -> Face 2: Closest is AlexandraDaddario (Dist: 1.04)
Found 2 faces in Test15.png.
  -> Face 1: Closest is BraddPitt (Dist: 1.03)
  -> Face 2: Closest is AlexandraDaddario (Dist: 0.20)
Found 2 faces in Test16.png.
  -> Face 1: Closest is BillieEilish (Dist: 0.78)
  -> Face 2: Closest is RobertDowneyJR (Dist: 0.40)
```



ทดสอบโมเดลด้วยภาพทั้งหมด 16 ภาพที่เตรียมไว้ใน test_images และใช้เป็นเกณฑ์ในการทดสอบประสิทธิภาพโมเดล โดยพบว่าจากทั้งหมด 16 ภาพ สามารถทำนายได้ถูก 14 ภาพ โดยมี 2 ภาพคือ Test09 และ Test10 ทำนายว่ามี Alexandra Daddario อยู่ในภาพถูกต้อง แต่ทำนายบุคคลที่อยู่ในภาพอีกคนหนึ่งผิดเป็น Bradd Pitt และอีกภาพที่ควรที่จะมี Robert Downey Jr กลับทำนายว่าไม่มี และได้ Distance ของ Robert Downey Jr เป็น 0.59 มองจากภาพรวมแล้วถือว่าโมเดลมีประสิทธิภาพที่ดีในระดับหนึ่ง แต่ความมั่นใจของโมเดลนี้ยังค่อนข้างต่ำ สังเกตได้จากภาพใบหน้าที่คำนวณถูกต้อง ยังคำนวณ Distance ได้เกือบเลย Recognization Threshold 0.5 ที่ตั้งไว้

เอกสารอ้างอิง

- datahacker. rs. (2021, November 9). *14 PyTorch tutorial - How to create Siamese Networks and create a face recognition system* [Video]. YouTube. https://www.youtube.com/watch?v=9hLcBgnY7cs
- datahacker.rs. (2021, November 7). *# 019 Siamese Network in PyTorch with application to face similarity*. https://datahacker.rs/019-siamese-network-in-pytorch-with-application-to-face-similarity/
- PyTorch. (n.d.). *TripletMarginLoss*. PyTorch Documentation. https://docs.pytorch.org/docs/stable/generated/torch.nn.TripletMarginLoss.html
- Sarıgöz, Y. (2022, March 24). *Triplet Loss – Advanced Intro*. Towards Data Science. https://towardsdatascience.com/triplet-loss-advanced-intro-49a07b7d8905/
- Schroff, F., Kalenichenko, D., & Philbin, J. (2015). *FaceNet: A Unified Embedding for Face Recognition and Clustering*. arXiv. https://arxiv.org/abs/1503.03832
- Face Recognition Dataset. https://www.kaggle.com/datasets/vasukipatel/face-recognition-dataset?resource=download

การแบ่งงาน

1. นายธนภัทร จำเริญ
   - ช่วยกันคิดหัวข้อและแนวทางแก้ปัญหา
   - ช่วยกันออกแบบสถาปัตยกรรม CNN
   - หาข้อมูลหลักการการใช้ Triplet Loss และ MTCNN
   - พัฒนาโค้ดส่วนการใช้ MTCNN ดึงหน้าจากภาพ
   - ช่วยกันพัฒนาโค้ดส่วนการสร้าง Triplet Dataloader และการเทรน
   - ทำรายงานส่วนการอธิบายโค้ด 5-8
   - ทำรายงานส่วนหน้าปก หลักการ Triplet Loss เอกสารอ้างอิง
2. นายภัทรพล ณ เนตร
   - ช่วยกันคิดหัวข้อและแนวทางแก้ปัญหา
   - ช่วยกันออกแบบสถาปัตยกรรม CNN
   - หา Dataset สำหรับการฝึก
   - พัฒนาโค้ดส่วนการดึงข้อมูลจาก Dataset
   - พัฒนาโค้ดรูปแบบโมเดล CNN
   - ช่วยกันพัฒนาโค้ดส่วนการสร้าง Triplet Dataloader และการเทรน
   - พัฒนาโค้ดส่วนทดสอบโมเดลจากภาพ test_images
   - ทำรายงานส่วนการอธิบายโค้ดข้อ 7-11
   - ทำรายงานส่วนที่มาและความสำคัญ การออกแบบสถาปัตยกรรม CNN