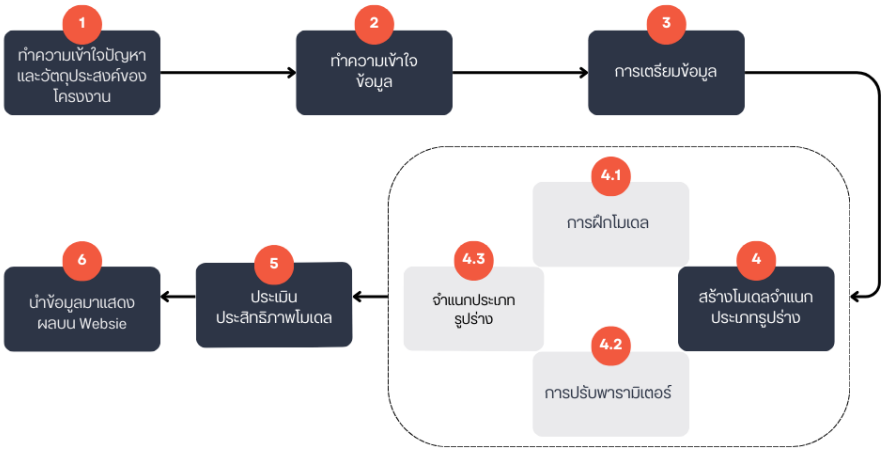
บทที่ 3
วิธีการดำเนินงาน

3.1 การทำความเข้าใจในธุรกิจ (Business Understanding)

โครงงานนี้มีวัตถุประสงค์ เพื่อพัฒนาวิธีการวิเคราะห์จำแนกประเภทรูปร่างแต่ละบุคคล โดยใช้ข้อมูลพื้นฐานนำไปวิเคราะห์จำแนกประเภทของรูปร่างอย่างมีประสิทธิภาพ นอกจากนี้ยังมุ่งเน้นการพัฒนาเว็บแอปพลิเคชันที่สามารถแสดงผลข้อมูลประเภทของรูปร่างที่แบ่งออกเป็น 4 ประเภท ได้แก่ หุ่นผู้หญิงทรงแอปเปิ้ล, หุ่นผู้หญิงทรงลูกแพร์, หุ่นผู้หญิงทรงนาฬิกาทราย และหุ่นผู้หญิงทรงสี่เหลี่ยมผืนผ้าแนะนำแนวทางการเลือกเสื้อผ้าที่เหมาะสม เพื่อช่วยให้ผู้ใช้งานสามารถเลือกเสื้อผ้าได้อย่างเหมาะสมกับรูปร่างของตนเอง จากการศึกษาวิธีการงานวิจัยที่เกี่ยวข้อง สามารถจัดเป็นขั้นตอนการวิจัยได้ดังนี้



รูปที่ 3.1 แสดงการดำเนินการวิจัย

3.2 ขั้นตอนการทำความเข้าใจข้อมูล (Data Understanding)

ทำความเข้าใจและวิเคราะห์ข้อมูลเพื่อจำแนกรูปร่างกลุ่มประชากรตัวอย่างจากนักศึกษามหาวิทยาลัยเทคโนโลยีราชมงคลล้านนาเชียงใหม่ กลุ่มตัวอย่างที่ต้องเก็บจำนวนทั้งหมด 380 คน จำนวนประชากรตัวอย่างดังกล่าวได้จากสูตรการคำนวณของ Yamane จากจำนวนนักศึกษาทุกระดับปริญญา มีจำนวนทั้งหมด 7,854 คน สามารถคำนวณได้ดังนี้

$$n = \frac{N}{1 + N(e^2)}$$

โดย

n คือ ขนาดกลุ่มตัวอย่างที่ต้องการ

N คือ จำนวนประชากรทั้งหมด (7,854 คน)

e คือ ค่าความคลาดเคลื่อนที่ต้องการ (5% หรือ 0.05)

แทนค่าลงในสมการ

$$n = \frac{7{,}854}{1 + 7{,}854(0.05^2)}$$

$$n = \frac{7{,}854}{1 + 7{,}854(0.05^2)}$$

$$n = \frac{7{,}854}{20.635}$$

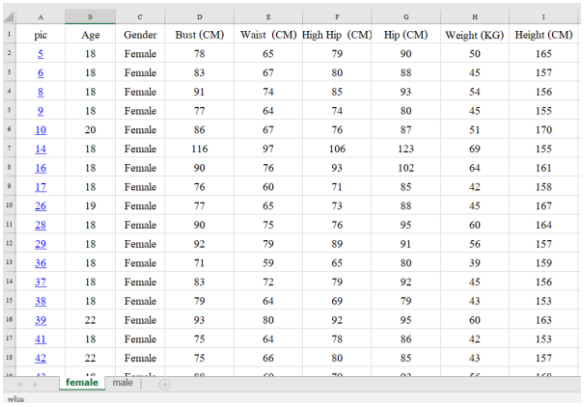
$$n \approx 380.69$$

ดังนั้น ขนาดกลุ่มตัวอย่างที่คำนวณได้คือ 380 คน โดยตัวอย่างนี้จะถูกแบ่งเป็นกลุ่มตัวอย่างเพศหญิงจำนวน 190 คน และกลุ่มตัวอย่างเพศชายจำนวน 190 คน ซึ่งเป็นขนาดกลุ่มตัวอย่างที่เหมาะสมในการศึกษา ทำการเก็บข้อมูลที่จำเป็นและคัดเลือกข้อมูลที่เหมาะสมเพื่อนำมาใช้ในการวิเคราะห์โมเดล ทำการเก็บข้อมูลจากกลุ่มตัวอย่าง โดยการสร้างแบบฟอร์มออนไลน์ผ่าน Google Form เพื่อรวบรวมข้อมูลที่จำเป็นทั้งหมด 9 แอตทริบิวต์ ดังตารางที่ 3.1

ตารางที่ 3.1 พจนานุกรมข้อมูล (Data Dictionary) จำนวน 9 แอตทริบิวท์

Field Name	Data Type	Description	Allowed Values	Units
Gender	Text	เพศของกลุ่มตัวอย่าง	Male, Female	–
Age	Integer	อายุ	0–100	ปี (Years)
Weight	Integer	น้ำหนัก	0–300	กิโลกรัม (kg)
Height	Integer	ส่วนสูง	0–300	เซนติเมตร (cm)
Bust	Integer	ขนาดรอบอก	0–200	เซนติเมตร (cm)
Waist	Integer	ขนาดรอบเอว	0–200	เซนติเมตร (cm)
Upper Hip	Integer	ขนาดรอบสะโพกบน	0–200	เซนติเมตร (cm)
Hip	Integer	ขนาดรอบสะโพก	0–200	เซนติเมตร (cm)
Image	Binary	ภาพถ่าย	JPG, PNG	–

โดยกลุ่มตัวอย่างทั้งหมดที่เก็บได้มีจำนวน 509 คน แบ่งเป็นเพศชาย 266 คน และเพศหญิง 243 คน ข้อมูลดังกล่าวจะถูกแยกไฟล์ตามเพศหญิงและเพศชาย เพื่อใช้ในการวิเคราะห์ต่อไป



pic	Age	Gender	Bust (CM)	Waist (CM)	High Hip (CM)	Hip (CM)	Weight (KG)	Height (CM)
5	18	Female	78	65	79	90	50	165
6	18	Female	83	67	80	88	45	157
8	18	Female	91	74	85	93	54	156
9	18	Female	77	64	74	80	45	155
10	20	Female	86	67	76	87	51	170
14	18	Female	116	97	106	123	69	155
16	18	Female	90	76	93	102	64	161
17	18	Female	76	60	71	85	42	158
26	19	Female	77	65	73	88	45	167
28	18	Female	90	75	76	95	60	164
29	18	Female	92	79	89	91	56	157
36	18	Female	71	59	65	80	39	159
37	18	Female	83	72	79	92	45	156
38	18	Female	79	64	69	79	43	153
39	22	Female	93	80	92	95	60	163
41	18	Female	75	64	78	86	42	153
42	22	Female	75	66	80	85	43	157

รูปที่ 3.2 ตารางแสดงข้อมูลจากการรวบรวมผ่าน Google From

3.3 ขั้นตอนทำการเตรียมข้อมูล (Data Preparation)

3.3.1 ทำการรวบรวมข้อมูลทั้งหมดที่เก็บมาจำนวน 509 คน ข้อมูลจะถูกแยกออกเป็น 2 ชุด ได้แก่ ข้อมูลเพศชายและข้อมูลเพศหญิง โดยถูกจัดเก็บเป็นไฟล์ Excel เพื่อความสะดวกในการจัดการและวิเคราะห์ในขั้นตอนต่อไป ข้อมูลในไฟล์จะประกอบไปด้วย 9 แอตทริบิวท์ ได้แก่ เพศ, อายุ, น้ำหนัก, ส่วนสูง, รอบอก, รอบเอว, รอบสะโพก, รอบสะโพกบน, และรูปภาพ เพื่อความเป็นระเบียบและการนำไปใช้ในการวิเคราะห์

3.3.2 ทำการทำความสะอาดข้อมูลเพื่อลบข้อมูลที่ขาดหาย ข้อมูลซ้ำซ้อน และข้อมูลที่ไม่ถูกต้อง เพื่อให้แน่ใจว่าข้อมูลที่นำไปใช้นั้นมีความสมบูรณ์ หลังจากทำความสะอาดข้อมูล คอลัมน์ที่เก็บรูปภาพ (pic) จะถูกเปลี่ยนให้แสดงชื่อไฟล์รูปภาพของกลุ่มตัวอย่างแต่ละคน เพื่อให้ข้อมูลตัวเลขและรูปภาพเชื่อมโยงกันอย่างถูกต้อง การเตรียมข้อมูลนี้มุ่งเน้นไปที่กลุ่มตัวอย่างเพศหญิง และหลังจากทำความสะอาดแล้ว กลุ่มตัวอย่างเพศหญิงที่เหลืออยู่มีจำนวน 240 คน ข้อมูลรูปภาพทั้งหมดจะถูกเบลอใบหน้าเพื่อป้องกันการระบุตัวตน

Apple Pear Hourglass Rectangle

รูปที่ 3.3 ตัวอย่างภาพถ่ายสี รูปร่างแต่ละประเภท

3.3.3 ทำการติดป้ายเลเบลให้กับข้อมูลเพื่อจัดกลุ่มและจำแนกรูปร่างออกเป็น 4 ประเภท ตามลักษณะสัดส่วนร่างกาย ได้แก่ หุ่นแอปเปิ้ล, หุ่นลูกแพร์, หุ่นนาฬิกาทราย, และหุ่นสี่เหลี่ยมผืนผ้า

ตารางที่ 3.2 รายชื่อ Class แต่ละประเภทรูปร่าง

Class	Class Name
Apple	หุ่นผู้หญิงทรงแอปเปิ้ล
Pear	หุ่นผู้หญิงทรงลูกแพร์
Hourglass	หุ่นผู้หญิงทรงนาฬิกาทราย
Rectangle	หุ่นผู้หญิงทรงสี่เหลี่ยมผืนผ้า

นำข้อมูลสัดส่วนที่เก็บรวบรวมไว้ ได้แก่ รอบอก, รอบเอว, รอบสะโพก, และรอบสะโพกบน ทำการคำนวณเพื่อติดป้ายเลเบลประเภทรูปร่างดังตาราง โดยใช้หลักการคำนวณการศึกษาวิจัย International Journal of Clothing Science and Technology (Lee et al, 2007) ซึ่งแบ่งสูตรการคำนวณรูปร่างของผู้หญิงออกเป็น 4 ประเภทดังนี้

1) หุ่นผู้หญิงทรงแอปเปิ้ล
   - If (bust – hips) ≥ 3.6" AND (bust – waist) < 9"
2) หุ่นผู้หญิงทรงลูกแพร์
   - If (hips – bust) ≥ 3.6" AND (hips – waist) < 9"
   - If (hips – bust) > 2" AND (hips – waist) ≥ 7" AND (high hip/waist) ≥ 1.193
3) หุ่นผู้หญิงทรงนาฬิกาทราย
   - If (bust – hips) ≤ 1" AND (hips – bust) < 3.6" AND (bust – waist) ≥ 9" OR (hips – waist) ≥ 10"
   - If (hips – bust) ≥ 3.6" AND (hips – bust) < 10" AND (hips – waist) ≥ 9" AND (high hip/waist) < 1.193
   - If (bust – hips) > 1" AND (bust – hips) < 10" AND (bust – waist) ≥ 9"
4) หุ่นผู้หญิงทรงสี่เหลี่ยมผืนผ้า
   - If (hips – bust) < 3.6" AND (bust – hips) < 3.6" AND (bust – waist) < 9" AND (hips – waist) < 10"

3.3.4 ทำการเลือกข้อมูลหรือฟีเจอร์ที่ต้องการใช้ในการสร้างโมเดลมีจำนวน 5 แอตทริบิวต์ ได้แก่ รหัสรูปภาพ, เพศ, อายุ, น้ำหนัก, ส่วนสูง, และเลเบลประเภทรูปร่าง บันทึกเป็นไฟล์ data_female.csv เพื่อทำการแปลงข้อมูลตัวเลขและรูปภาพให้ให้อยู่ในรูปแบบอาเรย์ (Array) เพื่อเตรียมพร้อมสำหรับการทำชุดข้อมูล (Dataset)

	A	B	C	D	E	F
1	**pic**	**Age**	**Gender**	**Weight (KG)**	**Height (CM)**	**Bodyshape**
2	5	18	Female	50	165	Pear
3	6	18	Female	45	157	Rectangle
4	8	18	Female	54	156	Rectangle
5	9	18	Female	45	155	Rectangle
6	10	20	Female	51	170	Rectangle
7	14	18	Female	69	155	Hourglass
8	16	18	Female	64	161	Pear
9	17	18	Female	42	158	Rectangle
10	26	19	Female	45	167	Hourglass
11	28	18	Female	60	164	Rectangle
12	29	18	Female	57	157	Rectangle
13	36	18	Female	39	159	Rectangle
14	37	18	Female	45	156	Rectangle
15	38	18	Female	43	153	Rectangle
16	39	22	Female	60	163	Rectangle
17	41	18	Female	42	153	Pear

รูปที่ 3.4 ตารางแสดงข้อมูลสำหรับทำชุด (Dataset)

3.3.5 ทำการรวมข้อมูล มีขั้นตอนดังต่อไปนี้

3.3.5.1 ทำการนำเข้าไลบรารีที่จำเป็นสำหรับการประมวลผลภาพ (OpenCV) การจัดการข้อมูล (Pandas, NumPy) และการแสดงผลภาพ (Matplotlib) และเชื่อมต่อกับ Google Drive เพื่อเข้าถึงไฟล์รูปภาพและไฟล์ข้อมูล CSV ที่เก็บไว้ใน Drive

```
#นำเข้าไลบารี่
import cv2
from google.colab import drive
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
```

รูปที่ 3.5 ตัวอย่างของไลบารี่นำมาที่ใช้งาน

3.3.5.2 ทำการนำเข้าไฟล์ข้อมูลตัวเลข data_female.csv ซึ่งเป็นไฟล์ที่เก็บข้อมูลตัวเลขที่จำเป็น เช่น อายุ, เพศ, น้ำหนัก, ส่วนสูงและเลเบลประเภทรูปร่างของแต่ละบุคคล จากนั้นทำการแปลงข้อมูลตัวเลขและรูปภาพให้ให้อยู่ในรูปแบบอาเรย์ (Array) เพื่อให้ข้อมูลทั้งหมดอยู่ในรูปแบบที่พร้อมสำหรับการประมวลผลในการสร้างโมเดล

```
#นำเข้าไฟล์ข้อมูล CSV
csv_file_path = '/content/drive/MyDrive/Project/data_female1.csv'
output_csv_path = '/content/drive/MyDrive/Project/merged_data.csv'
```

รูปที่ 3.6 ตัวอย่างนำเข้าไฟล์ data_female.csv

3.3.5.3 ทำการอัปโหลดรูปภาพของกลุ่มตัวอย่างที่ถูกเบลอใบหน้าแล้วขึ้นไปยัง Google Drive เพื่อแปลงเป็นรูปภาพขาวดำ (Grayscale) และทำการปรับขนาดของรูปภาพขาวดำให้เป็น 256x256 พิกเซล หลังจากทำการแปลงภาพเหล่านั้นให้เป็นอาเรย์ (Array) โดยที่แต่ละพิกเซลของภาพขาวดำจะถูกแทนด้วยค่าเชิงตัวเลข การแปลงภาพเป็นอาเรย์นี้ทำให้สามารถรวมเข้ากับข้อมูลตัวเลขอื่นๆ ได้อย่างสะดวก

3.3.5.4 ทำการรวมข้อมูลรูปภาพกับข้อมูลตัวเลข โดยการสร้าง list เพื่อจัดเก็บอาเรย์ของรูปภาพที่แปลงแล้วรวมเข้ากับข้อมูลตัวเลขในไฟล์ data_female.csv โดยจับคู่รูปภาพกับข้อมูลในไฟล์ CSV ตามรหัสรูปภาพ (ID) เพื่อให้ข้อมูลทั้งหมดถูกจัดกลุ่มเข้าด้วยกันอย่างถูกต้อง ทำการบันทึกข้อมูลทั้งหมดเป็นไฟล์ใหม่ชื่อว่า data_array.npy

```
#สร้าง list เก็บ Arrayของรูปภาพ grayscale ที่แปลงแล้ว
image_arrays = []
```

```
#วนลูปเพื่อแปลงรูปภาพเป็น grayscale อาเรย์ แล้วเพิ่มข้อมูลอาเรย์ลงใน DataFrame และบันทึก DataFrame ที่ปรับปรุงใหม่ลงในไฟล์ CSV
for filename in os.listdir(input_folder_path):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        # Extract the ID from the filename (assuming the filename structure matches the 'pic' column)
        pic_id = os.path.splitext(filename)[0]
        # Load image
        image_path = os.path.join(input_folder_path, filename)
        image = cv2.imread(image_path)
        # Convert to grayscale
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Convert grayscale image to 2D array
        gray_image_array = np.array(gray_image)
        # Find the row in the dataframe that matches the pic ID
        matched_row = df[df['pic'] == pic_id]
        if not matched_row.empty:
            # Add the numpy array to the list (optional, if you want to keep them in memory)
            image_arrays.append(gray_image_array)
            # Add the image array as a new column in the dataframe
            df.loc[df['pic'] == pic_id, 'image_array'] = [gray_image_array]
# Save the modified dataframe to a new CSV
df.to_csv(output_csv_path, index=False)
```

รูปที่ 3.7 ตัวอย่างข้อมูลอาเรย์รูปภาพรวมเข้ากับข้อมูลตัวเลข

3.3.5.5 ตรวจสอบข้อมูลไฟล์ data_array.npy ซึ่งประกอบด้วยข้อมูลตัวเลข, อาเรย์ของภาพขาวดำ, และเลเบลรูปร่างของกลุ่มตัวอย่างที่ถูกจัดเตรียมไว้ในขั้นตอนก่อนหน้านี้ เพื่อตรวจสอบจำนวนแถวและคอลัมน์ในข้อมูล

```
import numpy as np

# โหลดข้อมูลที่บันทึกในไฟล์ .npy ซึ่งประกอบด้วยข้อมูลตัวเลขและภาพที่ถูกจัดเตรียมไว้ในขั้นตอนก่อนหน้านี้
data = np.load('/content/drive/MyDrive/Project/data_array.npy', allow_pickle=True)
```

```
data.shape[0]
```

```
241
```

```
data[0, 6].shape
```

```
(256, 256)
```

รูปที่ 3.8 ตัวอย่างการนำเข้าไฟล์และตรวจสอบข้อมูล

3.3.5.6 ทำการแยกข้อมูลตามประเภท ได้แก่ ข้อมูลตัวเลข, อาเรย์ของภาพ, และเลเบลเลเบลรูปร่างทั้ง 4 ประเภท (หุ่นแอปเปิ้ล, หุ่นลูกแพร์, หุ่นนาฬิกาทราย, หุ่นสี่เหลี่ยมผืนผ้า) เพื่อตรวจสอบขนาดของอาเรย์ที่แยกออกมาแต่ละส่วน

```
#แยกข้อมูลออกเป็น 3 ส่วน: ข้อมูลตัวเลข, ภาพ และเลเบล
data_attributes = []
data_images = []
data_labels = []

for i in range(data.shape[0]):
  print(i)
  data_attributes.append(data[i, [1, 3, 4]])  # ข้อมูลตัวเลข เช่น เพศ, น้ำหนัก, ส่วนสูง
  data_images.append(data[i, [6]].tolist()[0].tolist())  # ภาพที่ถูกแปลงเป็น grayscale
  data_labels.append(data[i, [5]])  # เลเบลของคลาส (ประเภทของรูปร่าง)

np_attributes = np.array(data_attributes, dtype=np.float32)
np_images = np.array(data_images, dtype=np.float32)
# np_images = np_images.reshape(np_images.shape[0], np_images.shape[1], np_images.shape[2], 1)
np_labels = np.array(data_labels)
```

```
0
[5 18 'Female' 50 165]
['Pear']
[array([[124,  49, 118, ..., 162, 233, 234],
        [118,  68, 103, ..., 157, 233, 235],
        [119, 106,  50, ..., 165, 229, 231],
```

รูปที่ 3.9 ตัวอย่างข้อมูลอาเรย์ที่ถูกแยกเป็น 3 ส่วน

พบว่าข้อมูลกลุ่มตัวอย่าง หุ่นแอปเปิ้ลมีจำนวน 2 ตัวอย่าง, หุ่นนาฬิกาทรายมีจำนวน 13 ตัวอย่าง, หุ่นลูกแพร์มีจำนวน 89 ตัวอย่าง และหุ่นสี่เหลี่ยมผืนผ้ามีจำนวน 137 ตัวอย่าง ข้อมูลเกิดความไม่สมดุล (Imbalanced Data) อาจส่งผลกระทบต่อความแม่นยำของโมเดลในการจำแนกประเภทรูปร่างแต่ละบุคคล เนื่องจากโมเดลอาจมีแนวโน้มที่จะจำแนกรูปร่างที่มีตัวอย่างมากกว่าได้แม่นยำกว่า จึงต้องใช้เทคนิคการปรับสมดุลข้อมูลเพื่อให้ข้อมูลแต่ละประเภทมีความสมดุลมากขึ้น

```
# แสดงตัวอย่างของแต่ละคลาส
unique_labels, counts = np.unique(np_labels, return_counts=True)
for label, count in zip(unique_labels, counts):
    print(f"Class {label}: {count} samples")
```

```
Class Apple: 2 samples
Class Hourglass: 13 samples
Class Pear: 89 samples
Class Rectangle: 137 samples
```

รูปที่ 3.10 ตัวอย่างข้อมูลแต่ละประเภทที่ไม่สมดุล

3.3.5.7 ทำการแปลงเลเบลของคลาสให้อยู่ในรูปแบบ One-Hot Encoding ช่วยให้ข้อมูลอยู่ในรูปแบบที่สามารถใช้ในการฝึกโมเดลได้ โดยการแปลงค่า Original Value ไปเป็น Encoded Value ดังรูปนี้

```python
#แปลงเลเบลของคลาสให้อยู่ในรูปแบบ One-Hot Encoding เพื่อให้สามารถใช้ในการฝึกโมเดล
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
import pandas as pd

label_encoder = LabelEncoder()
# encoded_data = label_encoder.fit_transform(data_y)
encoded_data = label_encoder.fit_transform(np_labels)
one_encoded_data = to_categorical(encoded_data, num_classes=4)  # แปลงเป็น one-hot encoding

# สร้างตารางเพื่อดูค่าที่แปลงแล้วและค่าที่แท้จริง
label_mapping = pd.DataFrame({'Original Value': label_encoder.classes_, 'Encoded Value': range(len(label_encoder.classes_))})
print(label_mapping)
```

```
  Original Value  Encoded Value
0          Apple              0
1      Hourglass              1
2           Pear              2
3      Rectangle              3
/usr/local/lib/python3.10/dist-packa
```

รูปที่ 3.11 ตัวอย่างการแปลงเลเบลของคลาสเป็น One-Hot-Encoding

3.3.6 ทำการเพิ่มหรือลดจำนวนตัวอย่างของข้อมูลก่อนทำการสอนโมเดล เพื่อแก้ไขปัญหาข้อมูลไม่สมดุล (Imbalanced Data) ด้วยหลักการ 3 รูปแบบในการจัดการข้อมูลนี้ ได้แก่

3.3.6.1 การสุ่มเพิ่มข้อมูลแบบ Synthetic Minority Oversampling Technique (SMOTE) เป็นเทคนิคการเพิ่มข้อมูลในคลาสขนาดเล็ก เช่น หุ่นแอปเปิ้ลมีจำนวน 2 ตัวอย่าง, คลาสหุ่นนาฬิกาทรายมีจำนวน 13 ตัวอย่าง, หุ่นลูกแพร์มีจำนวน 89 ตัวอย่าง โดยทำการสังเคราะห์ตัวอย่างข้อมูลใหม่จากข้อมูลเดิม เพื่อเพิ่มข้อมูลเท่ากับคลาสที่มีตัวอย่างมากที่สุด คือ หุ่นแบบสี่เหลี่ยมผืนผ้าที่มีจำนวน 137 ตัวอย่าง ทำให้ข้อมูลในคลาสขนาดเล็กมีการกระจายตัวที่ดีขึ้น และช่วยให้ค่าความสำคัญของข้อมูลไม่สูญหาย ซึ่งจะช่วยลดปัญหาการเกิด Overfitting ได้

```python
#oversampling เทคนิค SMOTE เพื่อเพิ่มจำนวนตัวอย่างให้กับคลาสที่มีจำนวนน้อยกว่า
import numpy as np
from imblearn.over_sampling import SMOTE

# X คือ features (รวม attributes และ images) และ y คือ labels
X = np.hstack([np_attributes, np_images.reshape(241, -1)])
y = np.argmax(one_encoded_data, axis=1)

# ตรวจสอบจำนวนตัวอย่างก่อนการ oversampling
unique, counts = np.unique(y, return_counts=True)
print("ก่อน oversampling:", dict(zip(unique, counts)))

#กำหนด oversampling
n_samples = {0: 137, 1: 137, 2: 137}  # คลาส 3 (Rectangle) มีมากที่สุดแล้ว ไม่ต้องเพิ่ม
smote = SMOTE(sampling_strategy=n_samples, random_state=42, k_neighbors=1)
```

รูปที่ 3.12 ตัวอย่างการสุ่มเพิ่มข้อมูลแบบ SMOTE

```
# ทำ oversampling
X_resampled, y_resampled = smote.fit_resample(X, y)

# แยก attributes และ images ออกจากกัน
np_attributes_resampled = X_resampled[:, :3]
np_images_resampled = X_resampled[:, 3:].reshape(-1, 256, 256)

# สร้าง one-hot encoding สำหรับ labels ใหม่
one_encoded_data_resampled = np.eye(4)[y_resampled]

# ตรวจสอบจำนวนตัวอย่างหลัง oversampling
unique, counts = np.unique(y_resampled, return_counts=True)
print("หลัง oversampling:", dict(zip(unique, counts)))

np_attributes_resampled.shape, np_images_resampled.shape, one_encoded_data_resampled.shape
```

```
ก่อน oversampling: {0: 2, 1: 13, 2: 89, 3: 137}
หลัง oversampling: {0: 137, 1: 137, 2: 137, 3: 137}
((548, 3), (548, 256, 256), (548, 4))
```

รูปที่ 3.13 ตัวอย่างการสุ่มเพิ่มข้อมูลแบบ SMOTE (ต่อ)

3.3.6.2 การสุ่มเพิ่มข้อมูลแบบ Random Over Sampler (ROS) เป็นเทคนิคการเพิ่มข้อมูลในคลาสขนาดเล็ก เช่น หุ่นแอปเปิ้ลมีจำนวน 2 ตัวอย่าง, คลาสหุ่นนาฬิกาทรายมีจำนวน 13 ตัวอย่าง, หุ่นลูกแพร์มีจำนวน 89 ตัวอย่าง โดยทำการสุ่มตัวอย่างข้อมูลเดิมซ้ำเพื่อเพิ่มข้อมูลเท่ากับคลาสที่มีตัวอย่างมากที่สุดคือ หุ่นแบบสี่เหลี่ยมผืนผ้าที่มีจำนวน 137 ตัวอย่าง จะต้องระวังปัญหาการซ้ำซ้อนของข้อมูลจากการสุ่มซ้ำ ส่งผลให้โมเดลเกิดการ Overfitting ได้

```
#RandomOverSampler จะทำการสุ่มทำสำเนาตัวอย่างในคลาสที่มีจำนวนน้อยกว่าแทน
from imblearn.over_sampling import RandomOverSampler

# สมมติว่า X คือ features (รวม attributes และ images) และ y คือ labels
X = np.hstack([np_attributes, np_images.reshape(241, -1)])
y = np.argmax(one_encoded_data, axis=1)

# ตรวจสอบจำนวนตัวอย่างในแต่ละคลาสก่อน
unique, counts = np.unique(y, return_counts=True)
print("ก่อน oversampling:", dict(zip(unique, counts)))

# กำหนด n_samples ก่อนที่จะใช้
n_samples = {0: 137, 1: 137, 2: 137}  # คลาส 3 (Rectangle) มีมากที่สุดแล้ว ไม่ต้องเพิ่ม

# ใช้ RandomOverSampler สำหรับ oversampling
ros = RandomOverSampler(sampling_strategy=n_samples, random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)
```

รูปที่ 3.14 ตัวอย่างการสุ่มเพิ่มข้อมูลแบบ ROS

```
[42] # แยก attributes และ images ออกจากกัน
     np_attributes_resampled = X_resampled[:, :3]
     np_images_resampled = X_resampled[:, 3:].reshape(-1, 256, 256)

     # สร้าง one-hot encoding สำหรับ labels ใหม่
     one_encoded_data_resampled = np.eye(4)[y_resampled]

     # ตรวจสอบจำนวนตัวอย่างหลังจากการ Oversampling
     unique, counts = np.unique(y_resampled, return_counts=True)
     print("หลัง oversampling:", dict(zip(unique, counts)))

     np_attributes_resampled.shape, np_images_resampled.shape, one_encoded_data_resampled.shape

→ ก่อน oversampling: {0: 2, 1: 13, 2: 89, 3: 137}
  หลัง oversampling: {0: 137, 1: 137, 2: 137, 3: 137}
  ((548, 3), (548, 256, 256), (548, 4))
```

รูปที่ 3.15 ตัวอย่างการสุ่มเพิ่มข้อมูลแบบ ROS (ต่อ)

3.3.6.3 การสุ่มข้อมูลแบบผสม SMOTE Tomek เป็นเทคนิคผสมระหว่างการสุ่มเพิ่มข้อมูลด้วยเทคนิค SMOTE และการสุ่มลดข้อมูลด้วยเทคนิค Tomek's Link โดยมีการทำงานเป็นขั้นตอนดังนี้

1) ทำการสุ่มเพิ่มข้อมูลด้วย SMOTE เพิ่มข้อมูลในคลาสขนาดเล็ก เช่น หุ่นแอปเปิ้ลมีจำนวน 2 ตัวอย่าง, คลาสหุ่นนาฬิกาทรายมีจำนวน 13 ตัวอย่าง, หุ่นลูกแพร์มีจำนวน 89 ตัวอย่าง โดยทำการสังเคราะห์ตัวอย่างข้อมูลใหม่จากข้อมูลเดิม เพื่อเพิ่มข้อมูลเท่ากับคลาสที่มีตัวอย่างมากที่สุดคือ หุ่นแบบสี่เหลี่ยมผืนผ้าที่มีจำนวน 137 ตัวอย่าง

2) การสุ่มลดข้อมูลด้วย Tomek's Link หลังจากที่ทุกคลาสมีจำนวนตัวอย่างเท่ากับคลาสที่มีตัวอย่างมากที่สุดคือ หุ่นแบบสี่เหลี่ยมผืนผ้าที่มีจำนวน 137 ตัวอย่าง โดยทำการลบตัวอย่างที่อาจซ้ำซ้อนหรือมีลักษณะใกล้เคียงกันระหว่างคลาสต่าง ๆ เช่น คลาสหุ่นลูกแพร์ ซึ่งมีจำนวนตัวอย่างเพิ่มขึ้นเป็น 137 ตัวอย่างหลังการใช้ SMOTE จะถูกลดเหลือ 134 ตัวอย่าง และคลาสหุ่นแบบสี่เหลี่ยมผืนผ้าที่มี 137 ตัวอย่าง จะถูกลดจำนวนเหลือ 134 ตัวอย่าง

```
[43] #การสุ่มข้อมูลแบบผสม SMOTETomek
from imblearn.combine import SMOTETomek
from imblearn.over_sampling import SMOTE
import numpy as np

# สมมติว่า X คือ features (รวม attributes และ images) และ y คือ labels
X = np.hstack([np_attributes, np_images.reshape(241, -1)])
y = np.argmax(one_encoded_data, axis=1)

# ตรวจสอบจำนวนตัวอย่างในแต่ละคลาสก่อน
unique, counts = np.unique(y, return_counts=True)
print("ก่อนการใช้ SMOTETomek:", dict(zip(unique, counts)))

# สร้าง SMOTE object โดยกำหนดค่า k_neighbors
smote = SMOTE(k_neighbors=1, random_state=42)
# ใช้ SMOTETomek โดยนำ SMOTE ที่กำหนดไว้มาใช้
smote_tomek = SMOTETomek(smote=smote, random_state=42)
X_resampled, y_resampled = smote_tomek.fit_resample(X, y)

# แยก attributes และ images ออกจากกัน
np_attributes_resampled = X_resampled[:, :3]
np_images_resampled = X_resampled[:, 3:].reshape(-1, 256, 256)

# สร้าง one-hot encoding สำหรับ labels ใหม่
one_encoded_data_resampled = np.eye(4)[y_resampled]

# ตรวจสอบจำนวนตัวอย่างหลังจากการใช้ SMOTETomek
unique, counts = np.unique(y_resampled, return_counts=True)
print("หลังการใช้ SMOTETomek:", dict(zip(unique, counts)))

np_attributes_resampled.shape, np_images_resampled.shape, one_encoded_data_resampled.shape

ก่อนการใช้ SMOTETomek: {0: 2, 1: 13, 2: 89, 3: 137}
หลังการใช้ SMOTETomek: {0: 137, 1: 137, 2: 134, 3: 134}
((542, 3), (542, 256, 256), (542, 4))
```

รูปที่ 3.16 ตัวอย่างการสุ่มเพิ่มข้อมูลแบบ SMOTE Tomek

3.3.7 ทำการเตรียมข้อมูลด้วยหลักการ 3 รูปแบบในการจัดการข้อมูล ได้แก่ SMOTE ROS และ SMOTE Tomek เพื่อสร้างโมเดล ข้อมูลที่ใช้ในการสร้างโมเดลครั้งนี้ประกอบด้วยข้อมูลตัวเลข, ข้อมูลภาพขนาด 256x256 และ เลเบล 4 ประเภทรูปร่าง โดยแบ่งตัวอย่างข้อมูลออกเป็นชุดฝึก 60% (Training Set), ชุดทดสอบ 20% (Test Set) และชุดตรวจสอบ 20% (Validation Set) ดังรายละเอียดต่อไปนี้

1) การสุ่มเพิ่มข้อมูลแบบ SMOTE มีจำนวนตัวอย่างทั้งหมด 548 ถูกแบ่งออกเป็นชุดฝึกจำนวน 328 ตัวอย่าง, ชุดทดสอบจำนวน 110 ตัวอย่าง และชุดตรวจสอบจำนวน 110 ตัวอย่าง

รูปที่ 3.17 ตัวอย่างการแบ่งเป็นชุดฝึก, ชุดทดสอบ และชุดตรวจสอบ (SMOTE)

2) การสุ่มเพิ่มข้อมูลแบบ ROS มีจำนวนตัวอย่างทั้งหมด 542 ถูกแบ่งออกเป็นชุดฝึกจำนวน 325 ตัวอย่าง, ชุดทดสอบจำนวน 108 ตัวอย่าง และชุดตรวจสอบจำนวน 109 ตัวอย่าง

รูปที่ 3.18 ตัวอย่างการแบ่งเป็นชุดฝึก, ชุดทดสอบ และชุดตรวจสอบ (ROS)

3) การสุ่มเพิ่มข้อมูลแบบ SMOTETomek มีจำนวนตัวอย่างทั้งหมด 548 ถูกแบ่งออกเป็น ชุดฝึกจำนวน 328 ตัวอย่าง, ชุดทดสอบจำนวน 110 ตัวอย่าง และชุดตรวจสอบจำนวน 110 ตัวอย่าง

```
[36] from sklearn.model_selection import train_test_split

# แบ่งข้อมูลเป็น Train (60%) และ Temporary (40%)
attributes_train, attributes_temp, images_train, images_temp, labels_train, labels_temp = train_test_split(
    np_attributes_resampled, np_images_resampled, one_encoded_data_resampled, test_size=0.4, shuffle=True, random_state=42
)
attributes_test, attributes_val, images_test, images_val, labels_test, labels_val = train_test_split(
    attributes_temp, images_temp, labels_temp, test_size=0.5, shuffle=True, random_state=42
)
# แสดงขนาดของแต่ละชุดข้อมูล
print("Train set shapes:", attributes_train.shape, images_train.shape, labels_train.shape)
print("Test set shapes:", attributes_test.shape, images_test.shape, labels_test.shape)
print("Validation set shapes:", attributes_val.shape, images_val.shape, labels_val.shape)
```

```
Train set shapes: (325, 3) (325, 256, 256) (325, 4)
Test set shapes: (108, 3) (108, 256, 256) (108, 4)
Validation set shapes: (109, 3) (109, 256, 256) (109, 4)
```

รูปที่ 3.19 ตัวอย่างการแบ่งเป็นชุดฝึก, ชุดทดสอบ และชุดตรวจสอบ (SMOTETomek)

3.4 ขั้นตอนการออกแบบโมเดลและการสร้างโมเดล (Modeling)

3.4.1 การออกแบบและสร้างโมเดล เลือกใช้โมเดลแบบผสมผสานระหว่าง CNN ซึ่งเป็นโมเดลที่มีประสิทธิภาพสูงในการประมวลผลภาพและ Dense Layers สำหรับข้อมูลตัวเลขนี้ถูกนำมาใช้ร่วมกับ CNN เพื่อประมวลผลข้อมูลตัวเลข เช่น อายุ น้ำหนัก ส่วนสูง และสัดส่วนต่างๆ ของร่างกาย ซึ่งจะถูกเชื่อมต่อกับข้อมูลภาพในการสร้างโมเดลที่สามารถวิเคราะห์และจำแนกรูปร่างได้อย่างครอบคลุม

ตารางที่ 3.3 เปรียบเทียบ Convolutional Neural Networks (CNN), ResNet, VGG16 และ VGG18

คุณสมบัติ	Convolutional Neural Networks (CNNs)	ResNet	VGG16	VGG18
โครงสร้างพื้นฐาน	– ประกอบด้วยหลายชั้น: Conv, ReLU, Pooling, Fully Connected – สามารถมี Dropout layer และ Batch Normalization – มีการใช้ Conv layer ที่มีขนาด kernel ต่างกัน –ใช้ฟังก์ชันการแปลงเชิงเส้นในการปรับน้ำหนัก	– ใช้ Residual Learning ช่วยให้การฝึกโมเดลมีประสิทธิภาพ – ใช้ Identity shortcut connections มีความลึก 50, 101, 152 ชั้น – สามารถฝึกซ้ำได้โดยไม่เกิด vanishing gradient problem	– มีชั้น Convolution 13 ชั้น และ Fully Connected 3 ชั้น – ใช้ ReLU activation – มี Max Pooling layer และ Fully Connected layer – ใช้การขยายจำนวนชั้นเพื่อเพิ่มความซับซ้อนของโมเดล	– มีชั้น Convolution 15 ชั้น และ Fully Connected 3 ชั้น – ใช้ ReLU activation – มี Max Pooling layer และ Fully Connected layer – ใช้การขยายจำนวนชั้นเพื่อเพิ่มความซับซ้อนของโมเดล

ตารางที่ 3.3 เปรียบเทียบ Convolutional Neural Networks (CNN), ResNet, VGG16 และ VGG18 (ต่อ)

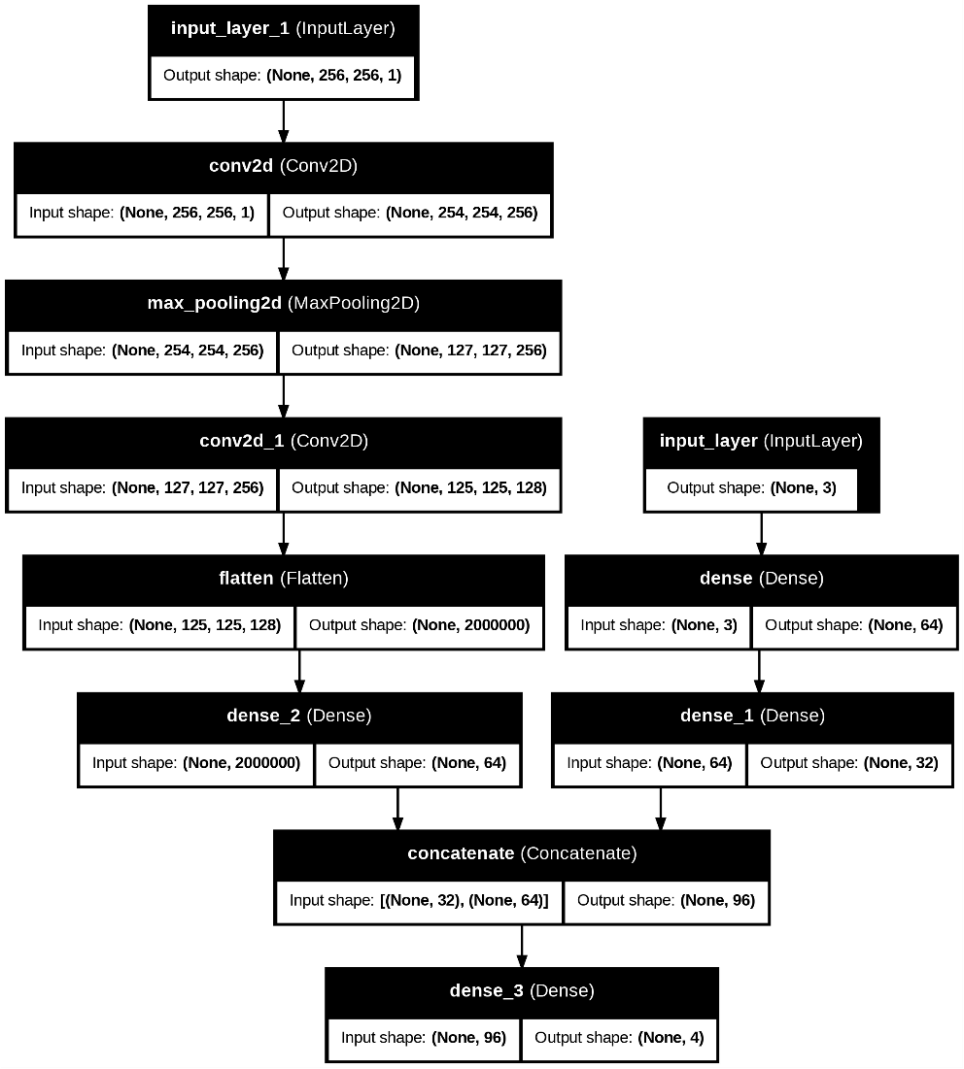
คุณสมบัติ	Convolutional Neural Networks (CNNs)	ResNet	VGG16	VGG18
ข้อดี	– สามารถเรียนรู้คุณลักษณะจากข้อมูลภาพ – การประมวลผลข้อมูลที่มีมิติสูงมีประสิทธิภาพ	ลดปัญหา vanishing gradient ทำให้การฝึกโมเดลง่ายขึ้น มีความแม่นยำสูงในการจำแนกประเภท	– ทำงานได้ดีในงานด้านการจำแนกประเภทภาพ – โครงสร้างที่เรียบง่ายและเข้าใจได้ง่าย	– ทำงานได้ดีในงานด้านการจำแนกประเภทภาพ – โครงสร้างที่เรียบง่าย
ข้อเสีย	– ความลึกของโมเดลมีข้อจำกัดในการเรียนรู้ – ปัญหา vanishing gradient ในโมเดลที่ลึก	– มีความซับซ้อนสูงในการออกแบบและฝึก – ต้องการหน่วยความจำมากในการฝึกโมเดล	– มีการใช้พารามิเตอร์มากทำให้การฝึกช้า – ไม่สามารถเรียนรู้คุณลักษณะที่ลึกซึ้งเหมือน ResNet	– มีการใช้พารามิเตอร์มากทำให้การฝึกช้า – ไม่สามารถเรียนรู้คุณลักษณะที่ลึกซึ้งเหมือน ResNet

จากตารางดังกล่าว เลือกใช้ Convolutional Neural Networks (CNNs) ที่ถูกพัฒนาขึ้นมาโดยเฉพาะเพื่อประมวลผลข้อมูลภาพ โดยมีความสามารถในการตรวจจับลักษณะเฉพาะและฟีเจอร์ที่สำคัญในภาพ เช่น รูปร่างและขอบเขตของวัตถุ ซึ่งช่วยให้การจำแนกและระบุวัตถุในภาพมีความแม่นยำสูง การทำงานของ CNN จะใช้กระบวนการที่เรียกว่า Convolution เพื่อวิเคราะห์ภาพในลักษณะเชิงพื้นที่ และใช้ Pooling เพื่อลดความซับซ้อนของข้อมูล ทำให้สามารถลดจำนวนพารามิเตอร์และประมวลผลได้รวดเร็วและมีประสิทธิภาพมากขึ้น นอกจากนี้ CNN ยังมีความยืดหยุ่นในการรับมือกับการเปลี่ยนแปลงของตำแหน่งและมุมมองของวัตถุในภาพ ทำให้สามารถตรวจจับได้แม้ว่าจะมีการเปลี่ยนแปลงเล็กน้อย โมเดลนี้ยังเหมาะสำหรับงานที่ต้องการการจำแนกประเภทและการวิเคราะห์ฟีเจอร์พื้นฐานในภาพ และสามารถปรับปรุงประสิทธิภาพได้ง่ายเมื่อข้อมูลมีความซับซ้อนเพิ่มขึ้น เช่น การเพิ่มเลเยอร์เพิ่มเติมในโครงสร้างของโมเดล

การเลือกใช้ Dense Layer เพื่อจัดการกับข้อมูลตัวเลขที่ไม่มีลักษณะเฉพาะ เช่น ข้อมูลที่เป็นตัวเลขในรูปแบบตาราง Dense Layer จะช่วยให้โมเดลสามารถเชื่อมโยงฟีเจอร์ต่าง ๆ เข้าด้วยกัน ทำให้สามารถจับความสัมพันธ์ในข้อมูลได้อย่างมีประสิทธิภาพ เหมาะสมกับการ

คาดการณ์หรือจำแนกประเภทของข้อมูล โดยเฉพาะข้อมูลที่มีความหลากหลาย Dense Layer มีความยืดหยุ่นในการปรับจำนวนของนิวรอนในแต่ละเลเยอร์และสามารถเพิ่มจำนวนเลเยอร์ได้ตามความต้องการ ทำให้สามารถใช้ในการวิเคราะห์ข้อมูลที่ซับซ้อนได้อย่างละเอียด และสามารถใช้ในขั้นตอนสุดท้ายของโมเดลเพื่อสรุปผลลัพธ์จากการประมวลผลที่ผ่านมาทั้งหมด

3.4.2 การออกแบบโมเดลเพื่อจำแนกประเภทรูปร่าง



รูปที่ 3.20 แสดงสถาปัตยกรรม CNN สำหรับการทำโมเดล

แบบจำลองการเรียนรู้เชิงลึก (Deep Learning Model) พัฒนาด้วยการผสมผสานระหว่าง Convolutional Neural Networks (CNN) และ Dense Layer โดยโครงสร้างของโมเดลนี้ถูกออกแบบมาเพื่อใช้ในการจำแนกประเภท (Classification) ทั้งจากข้อมูลภาพและข้อมูลตัวเลข

ได้อย่างมีประสิทธิภาพ CNN ถูกเลือกใช้เพราะสามารถจับลักษณะเฉพาะของภาพได้ดี ในขณะที่ Dense Layer ช่วยเชื่อมโยงฟีเจอร์ต่างๆ ของข้อมูลตัวเลข เพื่อทำให้การเรียนรู้และการจำแนกประเภทมีความแม่นยำมากยิ่งขึ้น โครงสร้างโมเดลประกอบด้วย

1) ชั้น Input Layer

– Input Layer ที่ 1 รับข้อมูลรูปภาพขาวดำขนาด 256x256 พิกเซล

– Input Layer ที่ 2 รับข้อมูลตัวเลข อายุ, น้ำหนัก, ส่วนสูง

2) ชั้น Convolutional Layers

– ชั้น conv2D ที่ 1 ใช้ตัวกรอง (filters) จำนวน 256 ซึ่งแต่ละตัวกรองจะสแกนภาพทีละส่วนเพื่อค้นหาคุณสมบัติต่างๆ ของภาพ เช่น ขอบ หรือลวดลายต่างๆ และให้ผลลัพธ์เป็นภาพที่มีขนาด 254x254 พิกเซล และมี 256 ช่องสัญญาณ

– ชั้น MaxPooling2D ทำการลดขนาดของภาพลงครึ่งหนึ่ง ช่วยลดจำนวนพารามิเตอร์และขนาดของข้อมูลที่ประมวลผล ผลลัพธ์ของชั้นนี้จะเป็นภาพที่มีขนาด 127x127 พิกเซล และมี 256 ช่องสัญญาณ

– ชั้น Conv2D ที่ 2 ใช้ตัวกรอง (filters) จำนวน 128 ตัว ซึ่งจะทำการประมวลผลในลักษณะคล้ายกับขั้นตอนแรก และจะได้ผลลัพธ์ที่มีขนาด 125x125 พิกเซล และมี 128 ช่องสัญญาณ

3) ชั้น Flatten Layer แปลงข้อมูลที่มีขนาด (125, 125, 128) หรือเรียกว่า 3D tensor ให้กลายเป็นเวกเตอร์แบบหนึ่งมิติ (1D vector) ที่มีขนาด 125*125*128 = 2,000,000 หน่วย เพื่อให้สามารถนำข้อมูลไปป้อนเข้าสู่ชั้น Dense Layer ซึ่งเป็นชั้นที่เชื่อมต่อแบบเต็ม (fully connected layer) ได้ง่ายขึ้น

4) Dense Layers ทำหน้าที่รวมผลลัพธ์จาก Dense Layer 1 ที่ประมวลผลข้อมูลภาพ 1 มิติ และ Dense Layer 2 ที่ประมวลผลข้อมูลตัวเลข (อายุ, น้ำหนัก, ส่วนสูง) จากนั้นทำการตัดสินใจขั้นสุดท้าย โดยการคำนวณผลรวมเชิงเส้น (linear combination) ซึ่งใช้น้ำหนักที่โมเดลได้เรียนรู้ และใช้ฟังก์ชันกระตุ้น เช่น ReLU, Sigmoid หรือ SoftMax เพื่อแปลงผลลัพธ์ให้เหมาะสมสำหรับการตัดสินใจ โดยในกรณีนี้โมเดลจะทำการจำแนกข้อมูลออกเป็น 4 ประเภทตามที่กำหนด

3.4.3 การสร้างโมเดลเพื่อจำแนกประเภทรูปร่าง

3.4.3.1 ทำการนำเข้าไลบารี่ TensorFlow และฟังก์ชันที่จำเป็นสำหรับการสร้างเลเยอร์ของโมเดล การแสดงภาพโครงสร้างของโมเดล และการคำนวณน้ำหนักของคลาสเพื่อแก้ปัญหาข้อมูลที่ไม่สมดุลของโมเดล

```python
#สร้างโมเดล CNN และ Dense Layer สำหรับการประมวลผล
import tensorflow as tf
from tensorflow.keras import layers, models # สร้างเลเยอร์และจัดการโดเมล
from tensorflow.keras.utils import plot_model #การสร้างภาพแสดงโครงสร้างของโมเดล
from sklearn.utils.class_weight import compute_class_weight
```

รูปที่ 3.21 ตัวอย่างไลบารี่และฟังก์ชันที่จำ

3.4.3.2 ทำการสร้างฟังก์ชันที่ทำหน้าสร้างและกำหนดค่าโมเดล Convolutional Neural Network (CNN) ที่รวมข้อมูลจากข้อมูลตัวเลขและข้อมูลภาพเพื่อทำการจำแนกประเภทรูปร่าง โดยมีการใช้ Convolutional Layers สำหรับการประมวลผลภาพ และ Dense Layers สำหรับการประมวลผลข้อมูลเชิงลักษณะ โมเดลทำการ Compile ด้วย optimizer Adam และ loss function categorical_crossentropy

```python
#สร้างโมเดล CNN และ Dense Layer สำหรับการประมวลผล
import tensorflow as tf
from tensorflow.keras import layers, models # สร้างเลเยอร์และจัดการโดเมล
from tensorflow.keras.utils import plot_model #การสร้างภาพแสดงโครงสร้างของโมเดล
from sklearn.utils.class_weight import compute_class_weight

#ฟังก์ชันนี้เป็นฟังก์ชันหลักที่สร้างโมเดลขึ้นมา
def create_model():
  #การกำหนด Input Layers:
  input_attributes = layers.Input(shape=(3,)) #ข้อมูลเชิงลักษณะ (เช่น อายุ, น้ำหนัก, ส่วนสูง)
  input_images = layers.Input(shape=(256, 256, 1))  #ข้อมูลภาพขนาด 256x256 พิกเซล ในรูปแบบ grayscale

  # การประมวลผลข้อมูลเชิงลักษณะถูกส่งผ่าน Dense Layers สองชั้น
  # โดยแต่ละชั้นมีการใช้ฟังก์ชันการกระตุ้นแบบ ReLUและมีจำนวนหน่วย 64 และ 32 ตามลำดับ
  x1 = layers.Dense(64, activation='relu')(input_attributes)
  x1 = layers.Dense(32, activation='relu')(x1)

  # การประมวลผลข้อมูลภาพสำหรับข้อมูลภาพจะถูกส่งผ่าน Convolutional Layers
  x2 = layers.Conv2D(256, (3, 3), activation='relu')(input_images)
  x2 = layers.MaxPooling2D((2, 2))(x2)
  x2 = layers.Conv2D(128, (3, 3), activation='relu')(x2)
  x2 = layers.Flatten()(x2)
  x2 = layers.Dense(64, activation='relu')(x2)
```

รูปที่ 3.22 การสร้างโมเดลและการกำหนดค่าโมเดล

```
#นำข้อมูลทั้ง 2 ทางรวมกันด้วยการใช้ฟังก์ชัน concatenate
combined = layers.concatenate([x1, x2])

#การสร้าง Output Layer:Output Layer จะมีหน่วยประมวลผล 4 หน่วย ซึ่งเหมาะสำหรับปัญหาที่มี 4 คลาส
#โดยใช้ฟังก์ชันการกระตุ้นแบบ Softmax เพื่อให้ค่าผลลัพธ์เป็นความน่าจะเป็นของแต่ละคลาส
output = layers.Dense(4, activation='softmax')(combined)

#สร้างโมเดลโดยกำหนด inputs และ outputs ตามที่กำหนดไว้ก่อนหน้านี้
#ทำการ Compile โมเดลโดยใช้ Optimizer แบบ Adam และ LossFunction แบบ categorical_crossentropy
#สำหรับการจัดการกับการจำแนกประเภท
model = models.Model(inputs=[input_attributes, input_images], outputs=output)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#แสดงรายละเอียดของโมเดลผ่าน model.summary()
model.summary()

#บันทึกโครงสร้างของโมเดลเป็นภาพ PNG ด้วย plot_model
plot_model(model, to_file='model_architecture.png', show_shapes=True, show_layer_names=True)
return model
```

รูปที่ 3.23 การสร้างโมเดลและการกำหนดค่าโมเดล (ต่อ)

3.4.4 การฝึกโมเดล Deep Learning ใช้ในการจำแนกรูปร่างของแต่ละบุคคล ประกอบด้วย Convolutional Neural Networks (CNN) สำหรับประมวลผลข้อมูลภาพ และ Dense Layers สำหรับจัดการข้อมูลตัวเลข การฝึกโมเดลแบ่งออกเป็น 4 ส่วนตามวิธีการสุ่มเพิ่มข้อมูลดังนี้

3.4.4.1 การฝึกโมเดลด้วยข้อมูลดั้งเดิมที่ยังไม่ผ่านการปรับสมดุลข้อมูล

1) นำเข้าข้อมูลที่ประกอบด้วย ข้อมูลภาพ และ ข้อมูลตัวเลข

2) แบ่งออกเป็น 3 ส่วน ดังนี้ ข้อมูลภาพ, ข้อมูลตัวเลข, และ เลเบลคลาส ซึ่งเป็นประเภทของรูปร่างที่ต้องการจำแนก

3) เลเบลของคลาสจะถูกแปลงเป็น One-Hot Encoding

4) ใช้เทคนิค SMOTE เพื่อเพิ่มจำนวนข้อมูลในคลาสที่มีตัวอย่างน้อย เพื่อให้ข้อมูลมีความสมดุลและโมเดลไม่เอียงในการเรียนรู้

5) แบ่งข้อมูลตัวอย่างออกเป็น 3 ชุด ได้แก่ ชุดฝึก 60%, ชุดทดสอบ 20% และชุดตรวจสอบ20%

6) เรียกใช้ฟังก์ชัน create_model เพื่อสร้างและกำหนดโครงสร้างของโมเดลที่จะใช้ในการฝึก

7) ทำการฝึกทั้งสิ้น 200 epochs โดยใช้ Early Stopping เพื่อหยุดการฝึกหากประสิทธิภาพของโมเดลเริ่มลดลง เพื่อป้องกันการเกิด Overfitting

```
model = create_model()
# การฝึกโมเดล (model.fit):
history = model.fit(
    [attributes_train, images_train], labels_train,
    epochs=200,
    batch_size=8,
    validation_data=([attributes_val, images_val], labels_val),
    callbacks=[early_stopping]
)
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 256, 256, 1)	0	-
conv2d (Conv2D)	(None, 254, 254, 256)	2,560	input_layer_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 127, 127, 256)	0	conv2d[0][0]
input_layer (InputLayer)	(None, 3)	0	-
conv2d_1 (Conv2D)	(None, 125, 125, 128)	295,040	max_pooling2d[0][0]
dense (Dense)	(None, 64)	256	input_layer[0][0]
flatten (Flatten)	(None, 2000000)	0	conv2d_1[0][0]
dense_1 (Dense)	(None, 32)	2,080	dense[0][0]
dense_2 (Dense)	(None, 64)	128,000,064	flatten[0][0]
concatenate (Concatenate)	(None, 96)	0	dense_1[0][0], dense_2[0][0]
dense_3 (Dense)	(None, 4)	388	concatenate[0][0]

```
Total params: 128,300,388 (489.43 MB)
Trainable params: 128,300,388 (489.43 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 27s 790ms/step - accuracy: 0.4679 - loss: 446.7224 - val_accuracy: 0.4898 - val_loss: 3.3384
Epoch 2/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 19s 135ms/step - accuracy: 0.5508 - loss: 8.6025 - val_accuracy: 0.5510 - val_loss: 2.2989
Epoch 3/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.7266 - loss: 1.0382 - val_accuracy: 0.5306 - val_loss: 1.2220
Epoch 4/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 99ms/step - accuracy: 0.7198 - loss: 0.6147 - val_accuracy: 0.5510 - val_loss: 1.4206
Epoch 5/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 100ms/step - accuracy: 0.9055 - loss: 0.3116 - val_accuracy: 0.6735 - val_loss: 2.0108
Epoch 6/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 101ms/step - accuracy: 0.9000 - loss: 0.2047 - val_accuracy: 0.6531 - val_loss: 2.3467
Epoch 7/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 102ms/step - accuracy: 0.9821 - loss: 0.0742 - val_accuracy: 0.6122 - val_loss: 2.9912
Epoch 8/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 99ms/step - accuracy: 1.0000 - loss: 0.0246 - val_accuracy: 0.5918 - val_loss: 3.4158
Epoch 9/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - accuracy: 1.0000 - loss: 0.0034 - val_accuracy: 0.5714 - val_loss: 3.6002
Epoch 10/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 100ms/step - accuracy: 1.0000 - loss: 0.0014 - val_accuracy: 0.5714 - val_loss: 3.9549
Epoch 11/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 99ms/step - accuracy: 1.0000 - loss: 5.2045e-04 - val_accuracy: 0.5714 - val_loss: 3.9930
Epoch 12/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 101ms/step - accuracy: 1.0000 - loss: 3.7242e-04 - val_accuracy: 0.5918 - val_loss: 4.0696
Epoch 13/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 101ms/step - accuracy: 1.0000 - loss: 2.3854e-04 - val_accuracy: 0.5918 - val_loss: 4.2493
Epoch 13: early stopping
Restoring model weights from the end of the best epoch: 3.
```

รูปที่ 3.24 การฝึกโมเดลด้วยข้อมูลดั้งเดิมที่ยังไม่ผ่านการปรับสมดุลข้อมูล

3.4.4.2 การฝึกโมเดลด้วยข้อมูลที่สุ่มเพิ่มแบบ SMOTE มีขั้นตอนดังนี้

1) นำเข้าข้อมูลที่ประกอบด้วย ข้อมูลภาพ และ ข้อมูลตัวเลข

2) แบ่งออกเป็น 3 ส่วน ดังนี้ ข้อมูลภาพ, ข้อมูลตัวเลข, และ เลเบลคลาส ซึ่งเป็นประเภทของรูปร่างที่ต้องการจำแนก

3) เลเบลของคลาสจะถูกแปลงเป็น One-Hot Encoding

4) ใช้เทคนิค SMOTE เพื่อเพิ่มจำนวนข้อมูลในคลาสที่มีตัวอย่างน้อย เพื่อให้ข้อมูลมีความสมดุลและโมเดลไม่เอียงในการเรียนรู้

5) แบ่งข้อมูลตัวอย่างออกเป็น 3 ชุด ได้แก่ ชุดฝึก 60%, ชุดทดสอบ 20% และชุดตรวจสอบ20%

6) เรียกใช้ฟังก์ชัน create_model เพื่อสร้างและกำหนดโครงสร้างของโมเดลที่จะใช้ในการฝึก

7) ทำการฝึกทั้งสิ้น 200 epochs โดยใช้ Early Stopping เพื่อหยุดการฝึกหากประสิทธิภาพของโมเดลเริ่มลดลง เพื่อป้องกันการเกิด Overfitting

```
model = create_model()
# การฝึกโมเดล (model.fit):
history = model.fit(
    [attributes_train, images_train], labels_train,
    epochs=200,
    batch_size=8,
    validation_data=([attributes_val, images_val], labels_val),
    callbacks=[early_stopping]
)
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 256, 256, 1)	0	-
conv2d (Conv2D)	(None, 254, 254, 256)	2,560	input_layer_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 127, 127, 256)	0	conv2d[0][0]
input_layer (InputLayer)	(None, 3)	0	-
conv2d_1 (Conv2D)	(None, 125, 125, 128)	295,040	max_pooling2d[0][0]
dense (Dense)	(None, 64)	256	input_layer[0][0]
flatten (Flatten)	(None, 2000000)	0	conv2d_1[0][0]
dense_1 (Dense)	(None, 32)	2,080	dense[0][0]
dense_2 (Dense)	(None, 64)	128,000,064	flatten[0][0]
concatenate (Concatenate)	(None, 96)	0	dense_1[0][0], dense_2[0][0]
dense_3 (Dense)	(None, 4)	388	concatenate[0][0]

รูปที่ 3.25 การฝึกโมเดล โดยการเพิ่มจำนวนตัวอย่างข้อมูลด้วยเทคนิค SMOTE

```
Total params: 128,300,388 (489.43 MB)
Trainable params: 128,300,388 (489.43 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 28s 382ms/step - accuracy: 0.4348 - loss: 1028.4535 - val_accuracy: 0.6000 - val_loss: 0.9644
Epoch 2/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 116ms/step - accuracy: 0.6700 - loss: 0.7372 - val_accuracy: 0.7909 - val_loss: 0.7850
Epoch 3/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 111ms/step - accuracy: 0.9369 - loss: 0.1916 - val_accuracy: 0.7818 - val_loss: 0.5053
Epoch 4/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 96ms/step - accuracy: 0.9885 - loss: 0.0482 - val_accuracy: 0.8000 - val_loss: 0.7117
Epoch 5/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 99ms/step - accuracy: 0.9843 - loss: 0.0319 - val_accuracy: 0.7909 - val_loss: 1.5259
Epoch 6/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 96ms/step - accuracy: 1.0000 - loss: 2.4989e-04 - val_accuracy: 0.8000 - val_loss: 1.4768
Epoch 7/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 97ms/step - accuracy: 1.0000 - loss: 5.7123e-05 - val_accuracy: 0.8000 - val_loss: 1.4171
Epoch 8/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 99ms/step - accuracy: 1.0000 - loss: 2.8311e-05 - val_accuracy: 0.8000 - val_loss: 1.4298
Epoch 9/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 97ms/step - accuracy: 1.0000 - loss: 5.7597e-05 - val_accuracy: 0.8000 - val_loss: 1.4436
Epoch 10/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 98ms/step - accuracy: 1.0000 - loss: 2.2373e-05 - val_accuracy: 0.8091 - val_loss: 1.4557
Epoch 11/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 100ms/step - accuracy: 1.0000 - loss: 2.5977e-05 - val_accuracy: 0.8091 - val_loss: 1.4697
Epoch 12/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 99ms/step - accuracy: 1.0000 - loss: 2.8148e-05 - val_accuracy: 0.8182 - val_loss: 1.4832
Epoch 13/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 98ms/step - accuracy: 1.0000 - loss: 2.7834e-05 - val_accuracy: 0.8182 - val_loss: 1.4962
Epoch 13: early stopping
Restoring model weights from the end of the best epoch: 3.
```

รูปที่ 3.26 การฝึกโมเดล โดยการเพิ่มจำนวนตัวอย่างข้อมูลด้วยเทคนิค SMOTE (ต่อ)

3.4.4.2 การฝึกโมเดลด้วยการสุ่มเพิ่มข้อมูลแบบ ROS มีขั้นตอนดังนี้

1) นำเข้าข้อมูลที่ประกอบด้วย ข้อมูลภาพ และ ข้อมูลตัวเลข

2) แบ่งออกเป็น 3 ส่วน ดังนี้ ข้อมูลภาพ, ข้อมูลตัวเลข, และ เลเบลคลาส ซึ่งเป็นประเภทของรูปร่างที่ต้องการจำแนก

3) เลเบลของคลาสจะถูกแปลงเป็น One-Hot Encoding

4) ใช้เทคนิค เพื่อสุ่มคัดลอกข้อมูลจากคลาสที่มีตัวอย่างน้อยเพื่อให้ข้อมูลมีความสมดุลและโมเดลไม่เอียงในการเรียนรู้

5) แบ่งข้อมูลตัวอย่างออกเป็น 3 ชุด ได้แก่ ชุดฝึก 60%, ชุดทดสอบ 20% และชุดตรวจสอบ20%

6) เรียกใช้ฟังก์ชัน create_model เพื่อสร้างและกำหนดโครงสร้างของโมเดลที่จะใช้ในการฝึก

7) ทำการฝึกทั้งสิ้น 200 epochs โดยใช้ Early Stopping เพื่อหยุดการฝึกหากประสิทธิภาพของโมเดลเริ่มลดลง เพื่อป้องกันการเกิด Overfitting

```
[59] model = create_model()
     # การฝึกโมเดล (model.fit):
     history = model.fit(
         [attributes_train, images_train], labels_train,
         epochs=200,
         batch_size=8,
         validation_data=([attributes_val, images_val], labels_val),
         callbacks=[early_stopping]
     )
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 256, 256, 1)	0	-
conv2d (Conv2D)	(None, 254, 254, 256)	2,560	input_layer_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 127, 127, 256)	0	conv2d[0][0]
input_layer (InputLayer)	(None, 3)	0	-
conv2d_1 (Conv2D)	(None, 125, 125, 128)	295,040	max_pooling2d[0][0]
dense (Dense)	(None, 64)	256	input_layer[0][0]
flatten (Flatten)	(None, 2000000)	0	conv2d_1[0][0]
dense_1 (Dense)	(None, 32)	2,080	dense[0][0]
dense_2 (Dense)	(None, 64)	128,000,064	flatten[0][0]
concatenate (Concatenate)	(None, 96)	0	dense_1[0][0], dense_2[0][0]
dense_3 (Dense)	(None, 4)	388	concatenate[0][0]

```
Total params: 128,300,388 (489.43 MB)
Trainable params: 128,300,388 (489.43 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 9s 161ms/step - accuracy: 0.2320 - loss: 2075.9385 - val_accuracy: 0.5455 - val_loss: 1.4206
Epoch 2/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 7s 118ms/step - accuracy: 0.5737 - loss: 1.2277 - val_accuracy: 0.7091 - val_loss: 0.8129
Epoch 3/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 103ms/step - accuracy: 0.8402 - loss: 0.5029 - val_accuracy: 0.7818 - val_loss: 0.9268
Epoch 4/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 101ms/step - accuracy: 0.9115 - loss: 0.1627 - val_accuracy: 0.8455 - val_loss: 0.8203
Epoch 5/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 6s 126ms/step - accuracy: 0.9758 - loss: 0.0554 - val_accuracy: 0.8000 - val_loss: 0.7253
Epoch 6/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 101ms/step - accuracy: 0.9964 - loss: 0.0401 - val_accuracy: 0.8364 - val_loss: 2.1264
Epoch 7/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 99ms/step - accuracy: 1.0000 - loss: 3.1326e-04 - val_accuracy: 0.8273 - val_loss: 1.8464
Epoch 8/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 100ms/step - accuracy: 1.0000 - loss: 3.6013e-04 - val_accuracy: 0.8273 - val_loss: 2.0923
Epoch 9/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 100ms/step - accuracy: 1.0000 - loss: 1.1999e-05 - val_accuracy: 0.8273 - val_loss: 2.1212
Epoch 10/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 99ms/step - accuracy: 1.0000 - loss: 1.6654e-05 - val_accuracy: 0.8273 - val_loss: 2.1500
Epoch 11/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 99ms/step - accuracy: 1.0000 - loss: 1.0159e-05 - val_accuracy: 0.8273 - val_loss: 2.1649
Epoch 12/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 110ms/step - accuracy: 1.0000 - loss: 7.7585e-06 - val_accuracy: 0.8273 - val_loss: 2.1783
Epoch 13/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 100ms/step - accuracy: 1.0000 - loss: 3.9394e-05 - val_accuracy: 0.8273 - val_loss: 2.1969
Epoch 14/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 99ms/step - accuracy: 1.0000 - loss: 4.4193e-06 - val_accuracy: 0.8273 - val_loss: 2.2063
Epoch 15/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 101ms/step - accuracy: 1.0000 - loss: 2.0369e-05 - val_accuracy: 0.8273 - val_loss: 2.2204
Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 5.
```

รูปที่ 3.27 การฝึกโมเดล โดยการเพิ่มจำนวนตัวอย่างข้อมูลด้วยเทคนิค ROS

3.4.4.1 การฝึกโมเดลด้วยข้อมูลที่สุ่มเพิ่มแบบ SMOTE Tomek มีขั้นตอนดังนี้

1) นำเข้าข้อมูลที่ประกอบด้วย ข้อมูลภาพ และ ข้อมูลตัวเลข

2) แบ่งออกเป็น 3 ส่วน ดังนี้ ข้อมูลภาพ, ข้อมูลตัวเลข, และ เลเบลคลาส ซึ่งเป็นประเภทของรูปร่างที่ต้องการจำแนก

3) เลเบลของคลาสจะถูกแปลงเป็น One-Hot Encoding

4) ใช้เทคนิค SMOTE Tomek เพื่อเพิ่มข้อมูลในคลาสที่มีตัวอย่างน้อย และลบข้อมูลซ้ำซ้อนในคลาสที่มีตัวอย่างมากเพื่อให้ข้อมูลมีความสมดุลและโมเดลไม่เอียงในการเรียนรู้

5) แบ่งข้อมูลตัวอย่างออกเป็น 3 ชุด ได้แก่ ชุดฝึก 60%, ชุดทดสอบ 20% และชุดตรวจสอบ20%

6) เรียกใช้ฟังก์ชัน create_model เพื่อสร้างและกำหนดโครงสร้างของโมเดลที่จะใช้ในการฝึก

7) ทำการฝึกทั้งสิ้น 200 epochs โดยใช้ Early Stopping เพื่อหยุดการฝึกหากประสิทธิภาพของโมเดลเริ่มลดลง เพื่อป้องกันการเกิด Overfitting

```
model = create_model()
# การฝึกโมเดล (model.fit):
history = model.fit(
    [attributes_train, images_train], labels_train,
    epochs=200,
    batch_size=8,
    validation_data=([attributes_val, images_val], labels_val),
    callbacks=[early_stopping]
)
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 256, 256, 1)	0	-
conv2d (Conv2D)	(None, 254, 254, 256)	2,560	input_layer_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 127, 127, 256)	0	conv2d[0][0]
input_layer (InputLayer)	(None, 3)	0	-
conv2d_1 (Conv2D)	(None, 125, 125, 128)	295,040	max_pooling2d[0][0]
dense (Dense)	(None, 64)	256	input_layer[0][0]
flatten (Flatten)	(None, 2000000)	0	conv2d_1[0][0]
dense_1 (Dense)	(None, 32)	2,080	dense[0][0]
dense_2 (Dense)	(None, 64)	128,000,064	flatten[0][0]
concatenate (Concatenate)	(None, 96)	0	dense_1[0][0], dense_2[0][0]
dense_3 (Dense)	(None, 4)	388	concatenate[0][0]

รูปที่ 3.28 การฝึกโมเดล โดยการเพิ่มจำนวนตัวอย่างข้อมูลด้วยเทคนิค SMOTE Tomek

```
Total params: 128,300,388 (489.43 MB)
Trainable params: 128,300,388 (489.43 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 32s 458ms/step - accuracy: 0.4336 - loss: 1518.5504 - val_accuracy: 0.6147 - val_loss: 0.7631
Epoch 2/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 113ms/step - accuracy: 0.7611 - loss: 0.5204 - val_accuracy: 0.7982 - val_loss: 0.4722
Epoch 3/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 96ms/step - accuracy: 0.9489 - loss: 0.1686 - val_accuracy: 0.8165 - val_loss: 0.4822
Epoch 4/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 98ms/step - accuracy: 1.0000 - loss: 0.0117 - val_accuracy: 0.8440 - val_loss: 0.5178
Epoch 5/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 97ms/step - accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 0.8532 - val_loss: 0.6258
Epoch 6/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 97ms/step - accuracy: 1.0000 - loss: 3.4107e-04 - val_accuracy: 0.8440 - val_loss: 0.7041
Epoch 7/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 99ms/step - accuracy: 1.0000 - loss: 2.1067e-04 - val_accuracy: 0.8532 - val_loss: 0.7420
Epoch 8/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 97ms/step - accuracy: 1.0000 - loss: 1.1779e-04 - val_accuracy: 0.8624 - val_loss: 0.7832
Epoch 9/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 97ms/step - accuracy: 1.0000 - loss: 7.0589e-05 - val_accuracy: 0.8624 - val_loss: 0.8131
Epoch 10/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 99ms/step - accuracy: 1.0000 - loss: 1.1464e-04 - val_accuracy: 0.8624 - val_loss: 0.8505
Epoch 11/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 5s 98ms/step - accuracy: 1.0000 - loss: 3.9810e-05 - val_accuracy: 0.8624 - val_loss: 0.8793
Epoch 12/200
41/41 ━━━━━━━━━━━━━━━━━━━━ 4s 98ms/step - accuracy: 1.0000 - loss: 5.0376e-05 - val_accuracy: 0.8624 - val_loss: 0.9051
Epoch 12: early stopping
Restoring model weights from the end of the best epoch: 2.
```

รูปที่ 3.29 การฝึกโมเดล โดยการเพิ่มจำนวนตัวอย่างข้อมูลด้วยเทคนิค SMOTE Tomek (ต่อ)

3.5 ขั้นตอนการประเมินประสิทธิภาพของโมเดล

การประเมินประสิทธิภาพของโมเดลเป็นขั้นตอนที่สำคัญ เพื่อวิเคราะห์ความสามารถของโมเดลในการพยากรณ์ข้อมูลที่หลากหลาย ข้อมูลดั้งเดิมที่ยังไม่ผ่านการปรับสมดุลข้อมูล และข้อมูลที่ถูกปรับด้วย 3 เทคนิคการสุ่มเพิ่มตัวอย่างที่หลากหลาย ได้แก่SMOTE, ROS และ SMOTE Tomek เพื่อเปรียบเทียบประสิทธิภาพและหาวิธีที่เหมาะสมที่สุดสำหรับการปรับปรุงคุณภาพของโมเดล

3.5.1 ตัวอย่างข้อมูลที่ไม่ผ่านการแก้ไขปัญหาข้อมูลไม่สมดุล

1) ค่าความสูญเสีย (Loss) 1.5057 บ่งชี้ว่าโมเดลทำผิดพลาดในกระบวนการพยากรณ์ โดยค่าความสูญเสียที่น้อยบ่งบอกถึงความผิดพลาดที่น้อยลงและโมเดลสามารถพยากรณ์ได้ใกล้เคียงกับค่าจริง

2) ค่าความแม่นยำ (Accuracy) 56.25% บ่งบอกเปอร์เซ็นต์ของการพยากรณ์ที่ถูกต้องเมื่อเปรียบเทียบกับข้อมูลจริง ค่าความแม่นยำที่สูงขึ้นหมายถึงโมเดลมีประสิทธิภาพในการจำแนกข้อมูลได้ถูกต้องมากขึ้น

```
# Evaluate the model on the test set
loss, accuracy = model.evaluate([attributes_test, images_test], labels_test)
print(f'Val Loss: {loss}')
print(f'Val Accuracy: {accuracy * 100:.2f}%')
```

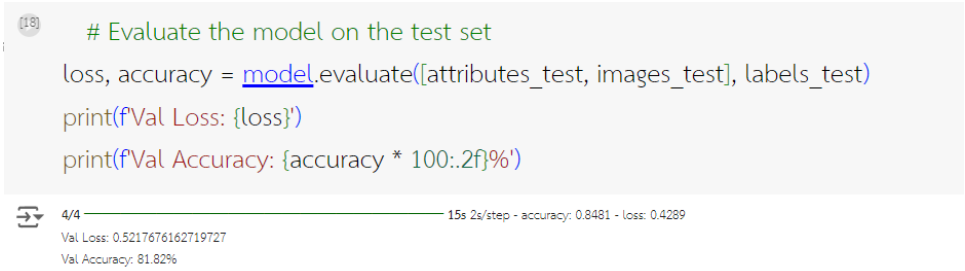
```
2/2 ━━━━━━━━━━━━━━━━━━━━ 15s 6s/step - accuracy: 0.5312 - loss: 1.5359
Val Loss: 1.5057672262191772
Val Accuracy: 56.25%
```

รูปที่ 3.30 ผลการประเมินโมเดล ตัวอย่างข้อมูลที่ไม่ผ่านการแก้ไขปัญหาข้อมูลไม่สมดุล

3.5.2 ตัวอย่างข้อมูลที่ปรับด้วยเทคนิค Synthetic Minority Oversampling Technique (SMOTE) ผลลัพธ์จากการประเมินดังนี้

1) ค่าความสูญเสีย (Loss) 0.5217 บ่งชี้ว่าโมเดลทำผิดพลาดในกระบวนการพยากรณ์ โดยค่าความสูญเสียที่น้อยบ่งบอกถึงความผิดพลาดที่น้อยลงและโมเดลสามารถพยากรณ์ได้ใกล้เคียงกับค่าจริง

2) ค่าความแม่นยำ (Accuracy) 81.82% บ่งบอกเปอร์เซ็นต์ของการพยากรณ์ที่ถูกต้องเมื่อเปรียบเทียบกับข้อมูลจริง ค่าความแม่นยำที่สูงขึ้นหมายถึงโมเดลมีประสิทธิภาพในการจำแนกข้อมูลได้ถูกต้องมากขึ้น

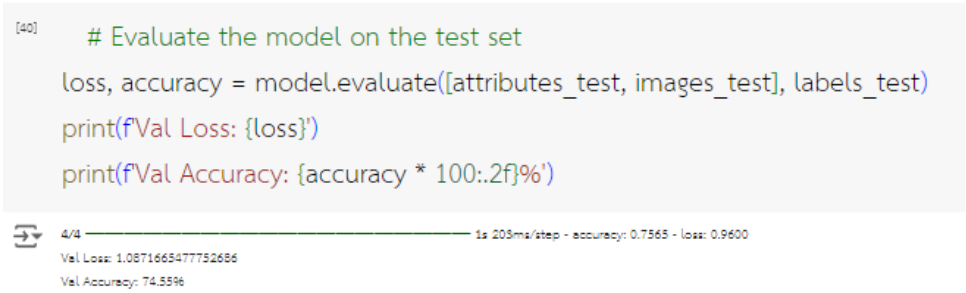


รูปที่ 3.31 ผลการประเมินโมเดล การเพิ่มจำนวนตัวอย่างข้อมูลด้วยเทคนิค SMOTE

3.5.3 ตัวอย่างข้อมูลที่ปรับด้วยเทคนิค RandomOverSampler (ROS)

1) ค่าความสูญเสีย (Loss) 1.0871 บ่งชี้ว่าโมเดลทำผิดพลาดในกระบวนการพยากรณ์ โดยค่าความสูญเสียที่น้อยบ่งบอกถึงความผิดพลาดที่น้อยลงและโมเดลสามารถพยากรณ์ได้ใกล้เคียงกับค่าจริง

2) ค่าความแม่นยำ (Accuracy) 74.55% บ่งบอกเปอร์เซ็นต์ของการพยากรณ์ที่ถูกต้องเมื่อเปรียบเทียบกับข้อมูลจริง ค่าความแม่นยำที่สูงขึ้นหมายถึงโมเดลมีประสิทธิภาพในการจำแนกข้อมูลได้ถูกต้องมากขึ้น

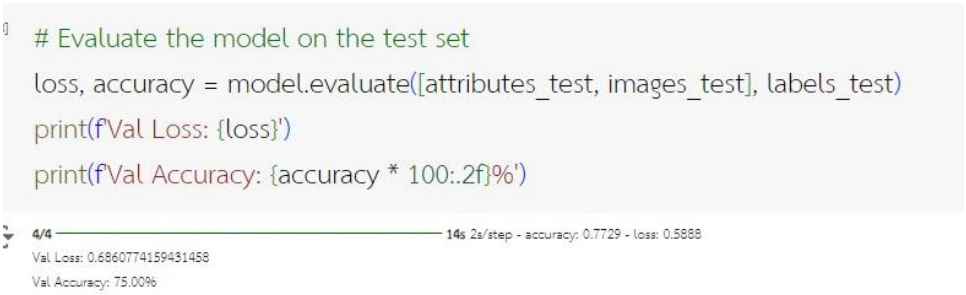


รูปที่ 3.32 ผลการประเมินโมเดลการเพิ่มจำนวนตัวอย่างข้อมูลด้วยเทคนิค ROS

3.5.4 ตัวอย่างข้อมูลที่ปรับด้วยเทคนิค SMOTE Tomek

1) ค่าความสูญเสีย (Loss) 0.6860 บ่งชี้ว่าโมเดลทำผิดพลาดในกระบวนการพยากรณ์ โดยค่าความสูญเสียที่น้อยบ่งบอกถึงความผิดพลาดที่น้อยลงและโมเดลสามารถพยากรณ์ได้ใกล้เคียงกับค่าจริง

2) ค่าความแม่นยำ (Accuracy) 75.00% บ่งบอกเปอร์เซ็นต์ของการพยากรณ์ที่ถูกต้องเมื่อเปรียบเทียบกับข้อมูลจริง ค่าความแม่นยำที่สูงขึ้นหมายถึงโมเดลมีประสิทธิภาพในการจำแนกข้อมูลได้ถูกต้องมากขึ้น



รูปที่ 3.33 ผลการประเมินโมเดล การเพิ่มจำนวนตัวอย่างข้อมูลด้วยเทคนิค SMOTE Tomek

3.6 ขั้นตอนการนำโมเดลไปใช้งานจริงโดยแสดงผลการวิเคราะห์ผ่านเว็บเบราว์เซอร์

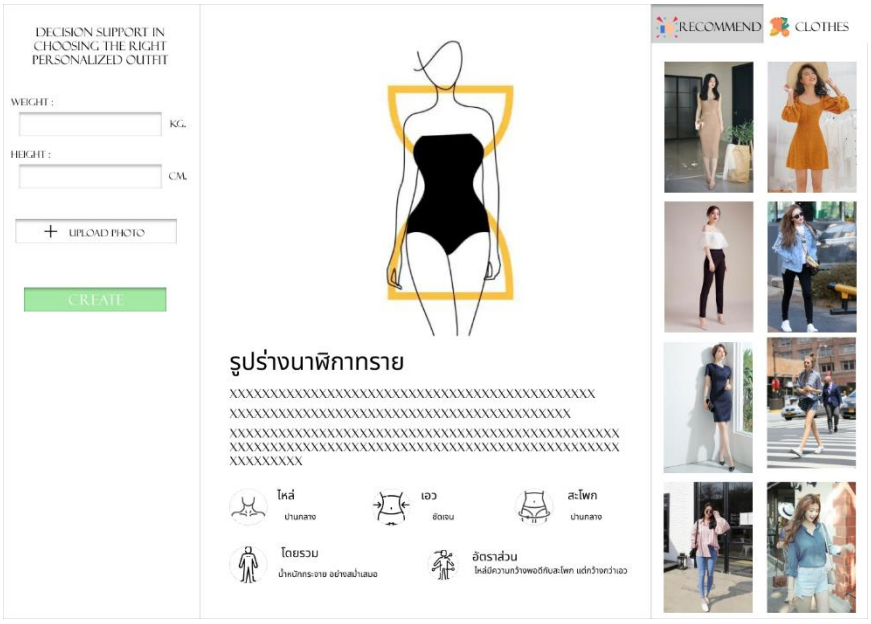
การนำโมเดลที่ได้รับการฝึกจากข้อมูลที่ปรับสมดุลด้วยเทคนิค SMOTE จนเสร็จแล้วมาประยุกต์ใช้ในระบบจริงผ่านเว็บเบราว์เซอร์ โดยมีการออกแบบและพัฒนาเว็บแอปพลิเคชันซึ่งแสดงผลการวิเคราะห์ข้อมูลผู้ใช้ มีการจำแนกรูปร่างและคำแนะนำการแต่งกายตามประเภทของรูปร่างเพื่อให้ผู้ใช้งานสามารถเข้าถึงได้ง่าย โดยสามารถแบ่งการพัฒนาออกเป็น 6 ขั้นตอนดังนี้

3.6.1 การวางแผนการพัฒนาเว็บแอปพลิเคชัน โดยกำหนดฟังก์ชันการทำงานของระบบ โดยผู้ใช้สามารถกรอกข้อมูลส่วนตัว เช่น น้ำหนัก ส่วนสูง รอบเอว และอัปโหลดรูปภาพ จากนั้นระบบจะวิเคราะห์ข้อมูลเหล่านี้และจำแนกรูปร่างของผู้ใช้ พร้อมทั้งแสดงคำแนะนำเกี่ยวกับการเลือกเสื้อผ้าที่เหมาะสม

3.6.2 การออกแบบหน้าเว็บไซต์ แสดงโครงสร้างส่วนต่างๆและเนื้อหาของรูปร่างนั้นๆ โดยแบ่งออกเป็น 3 ส่วน ดังนี้ส่วนฟอร์มกรอกข้อมูลส่วนตัวของผู้ใช้ ได้แก่ น้ำหนัก, ส่วนสูง, เพศ และอัปโหลดรูปภาพ ส่วนแสดงผลการจำแนก (Prediction Result) การวิเคราะห์จากโมเดล เมื่อผู้ใช้กรอกข้อมูลและอัปโหลดรูปภาพเสร็จสิ้น ระบบจะทำการประมวลผลและแสดงผลลัพธ์ประเภทของรูปร่างที่จำแนกได้เช่น รูปร่างแบบ Apple, Pear, Hourglass หรือ

Rectangle และส่วนแสดงคำแนะนำและตัวอย่างชุดที่เหมาะสม ระบบจะแสดงคำแนะนำเกี่ยวกับการเลือกเสื้อผ้าที่เหมาะสมและตัวอย่างเสื้อผ้าสำหรับประเภทของรูปร่างที่ได้รับการจำแนก

รูปที่ 3.34 แสดงโครงร่างการออกแบบหน้าแรกของเว็บไซต์และคำแนะนำ



รูปที่ 3.35 แสดงโครงร่างการออกแบบหน้าแรกของเว็บไซต์และชุดที่เหมาะสม

3.6.3 การพัฒนาเว็บไซต์ โดยสร้างเว็บสำหรับการแสดงผลข้อมูลบนเบราว์เซอร์โดยใช้ภาษาและเฟรมเวิร์ก เช่น HTML, CSS, JavaScript และ เฟรมเวิร์ก Flask เพื่อสร้างระบบที่สามารถรับข้อมูลจากผู้ใช้ ประมวลผลข้อมูลผ่านโมเดลการเรียนรู้เชิงลึก และแสดงผลการจำแนกรูปร่างพร้อมคำแนะนำการแต่งกายที่เหมาะสม

การออกแบบฟอร์มในส่วน Front-end ที่ให้ผู้ใช้กรอกข้อมูลส่วนตัว เช่น อายุ, น้ำหนัก, ส่วนสูง รวมถึงการอัปโหลดรูปภาพของตนเองเพื่อใช้ในการวิเคราะห์ ข้อมูลทั้งหมดจะถูกส่งผ่านฟอร์มที่พัฒนาโดยใช้ HTML, CSS และ JavaScript ซึ่งจะช่วยสร้างหน้าตาที่ใช้งานง่ายและรองรับการใช้งานผ่านหลากหลายอุปกรณ์

```
<div class="col-md-2" style="padding-left:20px; padding-top:20px;">
    <p class="font-head multi-shadow-text" style="padding-bottom: 20px;">
        DECISION SUPPORT FOR<br>CHOOSING THE RIGHT<br>PERSONALIZED OUTFIT<br>
    </p>
    <form id="predictionForm" action="/" method="POST" enctype="multipart/form-data">
        <label for="weight">WEIGHT :</label><br>
        <input type="text" id="weight" name="weight" required> KG<br><br>

        <label for="height">HEIGHT :</label><br>
        <input type="text" id="height" name="height" required> CM<br><br>

        <label for="age">AGE :</label><br>
        <input type="text" id="age" name="age" required> YEAR<br><br>

        <label for="file">UPLOAD PHOTO :</label><br><br>
        <input type="file" id="file" name="file" required><br><br><br>

        <input type="submit" value="PREDICT">
    </form>
</div>
```

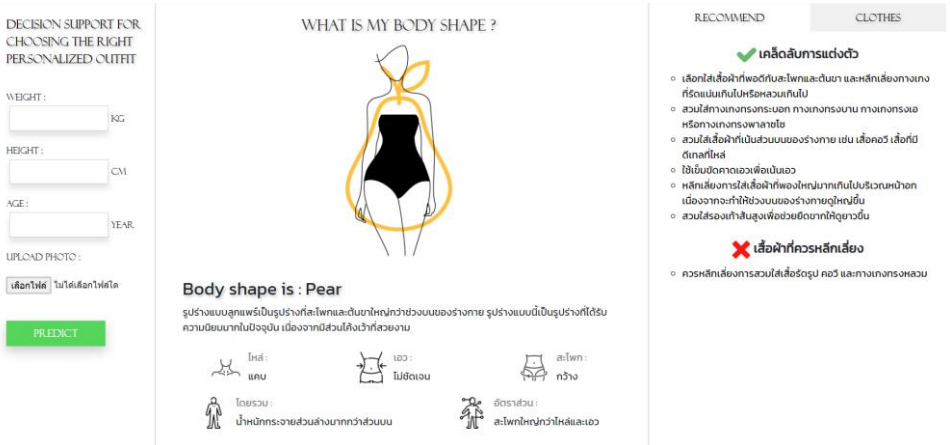
รูปที่ 3.36 หน้าฟอร์มกรอกข้อมูล

ส่วนของ Back-end ได้ใช้ Flask ซึ่งเป็นเฟรมเวิร์กของ Python ในการสร้าง API สำหรับรับข้อมูลจากฟอร์มผู้ใช้ ข้อมูลเหล่านี้ทั้งข้อมูลภาพและข้อมูลตัวเลขจะถูกส่งไปยังโมเดลที่พัฒนาขึ้นเพื่อทำการประมวลผล โดยโมเดลการเรียนรู้เชิงลึก (Deep Learning Model) ที่ผ่านการฝึกฝนแล้วจะวิเคราะห์ข้อมูลและจำแนกรูปร่างออกมาเป็นประเภท เช่น Apple, Pear, Hourglass, หรือ Rectangle จากนั้นระบบจะส่งผลลัพธ์กลับไปยังหน้าเว็บ

```python
# deep learning model
import tensorflow as tf
MODEL = tf.keras.models.load_model('models/my_model.h5')

label_mapping = ['Apple', 'Hourglass', 'Pear', 'Rectangle']

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads/'

# สร้างโฟลเดอร์สำหรับเก็บไฟล์ที่อัพโหลด ถ้ายังไม่มีโฟลเดอร์
if not os.path.exists(app.config['UPLOAD_FOLDER']):
    os.makedirs(app.config['UPLOAD_FOLDER'])

@app.route('/', methods=['GET', 'POST'])
def index():
    # ตั้งค่าเริ่มต้นทุกครั้งที่ฟังก์ชันถูกเรียกใช้งาน
    result_label = None
    result_image = url_for('static', filename='default.jpg')  # ตั้งค่า default.jpg ทุกครั้งที่โหลดหน้าเว็บ

    if request.method == 'POST':
        # รับข้อมูลจากฟอร์ม
        weight = request.form['weight']
        height = request.form['height']
        age = request.form['age']
```

รูปที่ 3.37 การขั้นตอนการสร้าง API ด้วย Flask

เมื่อการวิเคราะห์เสร็จสิ้น เว็บแอปพลิเคชัน จะแสดงผลการจำแนกรูปร่างที่ชัดเจนและเข้าใจง่าย พร้อมคำแนะนำการแต่งกายที่เหมาะสมสำหรับรูปร่างแต่ละประเภท เพื่อให้ผู้ใช้สามารถนำไปใช้ในการเลือกชุดที่เหมาะสมกับตนเองได้



รูปที่ 3.37 หน้าแสดงผลลัพธ์และคำแนะนำ

3.6.4 การทดสอบการทำงานของระบบ โดยทดสอบฟังก์ชันแต่ละส่วน เช่น การอัปโหลดรูปภาพและการประมวลผลข้อมูล จากนั้นทดสอบการทำงานร่วมกันระหว่าง Front-end และ Back-end ตั้งแต่การกรอกข้อมูลจนถึงการแสดงผลลัพธ์ และตรวจสอบความแม่นยำของโมเดลในการจำแนกรูปร่างจากข้อมูลผู้ใช้ เพื่อให้ได้ผลลัพธ์ที่ถูกต้อง

3.6.5 การเผยแพร่ โดยใช้ Git เพื่อส่งโค้ดไปยังแพลตฟอร์มที่เลือก เมื่อโค้ดถูกอัปโหลดไปแล้ว ระบบจะทำการ Build และเริ่มรันแอปพลิเคชันบนเซิร์ฟเวอร์จริง จากนั้นผู้ใช้งานสามารถเข้าถึงเว็บแอปพลิเคชันได้ผ่าน URL ที่กำหนด เช่น https://myapp.com

3.6.6 การดูแลบำรุงรักษาเว็บไซต์ โดยตรวจสอบและแก้ไขปัญหาที่เกิดขึ้นในระบบ ปรับปรุงระบบให้ปลอดภัยจากการโจมตีและปกป้องข้อมูลผู้ใช้