Mission 1. 패션 스타일 이미지 분류

Mission 1.1

1.1.1 구글 드라이브 연결과 필요한 라이브러리 / 모듈 로드

```
from google.colab import drive
drive.mount('/content/drive')                  # 구글 드라이브에 연결

import os                                      # 내 컴퓨터의 파일을 이용하기 위해 가져옴
import pandas as pd                            # 데이터프레임을 다루기 위해 가져옴
import numpy as np                             # 배열을 사용하기 위해 가져옴
import cv2                                     # 이미지 데이터를 로드, 저장하기 위해 가져옴
import shutil                                  # 이미지 데이터를 복사하기 위해 가져옴
import torch                                   # 전처리, 딥러닝을 위해 가져옴

from collections import defaultdict            # 딕셔너리를 사용하기 위해 가져옴
from PIL import Image                          # 이미지 데이터를 전처리하기 위해 가져옴
from IPython.display import display, HTML      # 데이터프레임 출력을 위해 가져옴
```

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

1.1.2 파일명을 분석하여 성별과 스타일을 추출하는 함수를 정의

```
def filename_anal(filename):                                                    # 함수 이름: filename_anal
    parts = filename.split('_')                                                 # 파일명을 _를 기준으로 나눔
    style = parts[3]                                                            # style은 파일명의 4번째 parts
    gender_code = parts[4].split('.')[0]                                        # 남/여 구분은 파일명의 5번째 parts + .jpg를 삭제하여 W/M만 추출
    gender = '여성' if gender_code == 'W' else '남성'                             # gender_code가 W라면 여성을, W가 아니면(=M이면) 남성을 할당
    return style, gender                                                        # 추출한 스타일과 성별을 반환
```

1.1.3 통계 데이터를 저장할 딕셔너리 생성

```
training_stats = defaultdict(lambda: defaultdict(int))                          # training
validation_stats = defaultdict(lambda: defaultdict(int))                        # validation
```

1.1.4 파일명에서 정보 추출 및 변환

```
training_dir = '/content/drive/MyDrive/Original_Data/training_image'
validation_dir = '/content/drive/MyDrive/Original_Data/validation_image'        # 파일명을 읽어올 데이터의 경로 지정

for filename in os.listdir(training_dir):                                       # 지정해둔 경로의 모든 파일명을 가져옴
        style, gender = filename_anal(filename)                                 # 정의해둔 함수 filename_anal을 이용해 style과 gender를 추출
        training_stats[gender][style] += 1                                      # 추출한 성별, 스타일에 맞는 이미지 수를 +1

for filename in os.listdir(validation_dir):                                     # validation 또한 반복
        style, gender = filename_anal(filename)
        validation_stats[gender][style] += 1
```

1.1.5 데이터프레임을 생성하는 함수를 정의

```
def create_dataframe(stats):                                                                                                    # 함수 이름: creat
    gender_styles = [[gender, style, count] for gender, style_data in stats.items() for style, count in style_data.items()]
    df_stats = pd.DataFrame(gender_styles, columns=['성별', '스타일', '이미지수'])                                                  # 열을 성별, 스타일
    df_stats = df_stats.query('이미지수 != 0')
    df_stats['성별'] = df_stats['성별'].mask(df_stats['성별'].duplicated(), '')                                                  # 중복 성별 빈칸 처
    return df_stats
```

1.1.6 데이터프레임 생성 및 수정

```
train_df = create_dataframe(training_stats)
val_df = create_dataframe(validation_stats)                                     # 데이터 프레임 생성
```

```
styled_df_train = train_df.style.set_table_styles(
    [
        {'selector': 'th', 'props': [('text-align', 'center'), ('border', '1px solid black')]},       # 헤더 셀 중앙 정렬 및 테두리
        {'selector': 'td', 'props': [('text-align', 'center'), ('border', '1px solid black')]},       # 데이터 셀 중앙 정렬 및 테두리
    ]
).hide(axis="index")                                                                                  # 인덱스 숨김


styled_df_val = val_df.style.set_table_styles(                                                        # validation에 반복
    [
        {'selector': 'th', 'props': [('text-align', 'center'), ('border', '1px solid black')]},
        {'selector': 'td', 'props': [('text-align', 'center'), ('border', '1px solid black')]},
    ]
).hide(axis="index")

# 제목 설정
train_title = HTML("<h2>Training 데이터 프레임</h2>")
val_title = HTML("<h2>Validation 데이터 프레임</h2>")
```

1.1.7 Training 데이터 프레임

```
display(train_title, styled_df_train)
```

Training 데이터 프레임

성별	스타일	이미지수
여성	ecology	64
	minimal	139
	military	33
	bodyconscious	95
	athleisure	67
	genderless	77
	feminine	154
	sportivecasual	157
	classic	77
	powersuit	120
	kitsch	91
	cityglam	67
	hippie	91
	grunge	31
	hiphop	48
	popart	41
	disco	37
	normcore	153
	space	37
	punk	65
	oriental	78
	lounge	45
	lingerie	55
남성	hiphop	274
	ivy	237
	metrosexual	278
	normcore	364
	sportivecasual	298
	hippie	260
	bold	268

1.1.8 Validation 데이터 프레임

```
display(val_title, styled_df_val)                    # Validation 데이터 프레임 출력
```

Validation 데이터 프레임

성별	스타일	이미지수
여성	feminine	44
	cityglam	18
	minimal	35
	hippie	14
	kitsch	22
	military	9
	powersuit	34
	sportivecasual	48
	genderless	12
	oriental	18
	grunge	10
	bodyconscious	23
	athleisure	14
	space	15
	lingerie	5
	classic	22
	punk	12
	disco	10
	normcore	20
	ecology	17
	popart	8
	hiphop	8
	lounge	8
남성	bold	57
	hiphop	66
	hippie	82
	ivy	79
	mods	80
	metrosexual	58
	normcore	51

Mission 1.2

1.2.1 YOLO v5l 로드 / GPU 연결

```
y5 = torch.hub.load('ultralytics/yolov5', 'yolov5l')                    # YOLOv5-Large 모델 로드
device = 'cuda' if torch.cuda.is_available() else 'cpu'                 # GPU를 사용하도록 설정
```

```
/usr/local/lib/python3.10/dist-packages/torch/hub.py:330: UserWarning: You are about to download and run code from an untrusted repository. In a
  warnings.warn(
Downloading: "https://github.com/ultralytics/yolov5/zipball/master" to /root/.cache/torch/hub/master.zip
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://docs.ultralytics.com/quickstart/
YOLOv5 🚀 2024-10-29 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
```

```
Downloading https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5l.pt to yolov5l.pt...
100%|██████████| 89.3M/89.3M [00:00<00:00, 98.8MB/s]

Fusing layers...
YOLOv5l summary: 367 layers, 46533693 parameters, 0 gradients, 109.0 GFLOPs
Adding AutoShape...
```

1.2.2 전처리 전/후 폴더 경로 설정

```
origin_folder = '/content/drive/MyDrive/Original_Data/training_image'          # 원본 이미지가 저장된 폴더 경로
prepro_folder = '/content/drive/MyDrive/Preprocessed_Data/prepro_training'     # 전처리된 이미지를 저장할 폴더 경로
```

1.2.3 이미지 크기 조정 함수 정의

```
def resize(image, target_size=(224, 224)):                                    # 함수 이름은 resize, 크기는 224*224
    img = Image.fromarray(image)                                               # numpy 배열을 Pillow 이미지 객체로 변환
    img.thumbnail(target_size, Image.LANCZOS)                                  # 비율을 유지하며 크기만 target_size로 조정

    img_gray = Image.new("RGB", target_size, (128, 128, 128))                  # 조정한 이미지를 붙일 새로운 회색 이미지 생
    img_gray.paste(img, ((target_size[0] - img.size[0]) // 2, (target_size[1] - img.size[1]) // 2))   # 회색 배경 위에 리사이즈된 이미지를 붙여넣
    return np.array(img_gray)                                                  # Pillow 이미지 객체를 다시 numpy 배열로 변환
```

1.2.4 이미지에서 사람 객체 추출+리사이즈 함수 정의

```
def preprocess(image_path):                                                    # 함수 이름: preprocess
    img = cv2.imread(image_path)                                               # 이미지 로드

    result = y5(img)                                                           # YOLOv5l 모델을 사용해 입력된 이미지에 대해 예측 수행
    prediction = result.xyxy[0]                                                # 예측 결과를 반환
    person_box = []                                                            # 사람 객체의 경계 정보를 저장할 리스트 초기화
    CONF_THRESHOLD = 0.5                                                       # 임계값 설정 (신뢰도가 이 값을 넘어야 사람으로 인정)

    for *box, conf, cls in prediction:                                         # YOLOv5l의 예측 결과로부터 좌표(box), 신뢰도(conf), 클래스(cls) 정
        if int(cls) == 0 and conf > CONF_THRESHOLD:                            # cls = 0(사람)이고, 신뢰도가 0.5 이상이면 사람 객체로 인식
            person_box.append(box)                                             # 인식된 사람 객체의 좌표를 리스트에 추가

    if len(person_box) > 0:                                                    # 이미지에서 한 명 이상의 사람이 감지되면 실행
        largest_person = max(person_box, key=lambda box: (box[2] - box[0]) * (box[3] - box[1]))   # max 함수로 person_box에서 가장 큰 바운딩 박스를
        x1, y1, x2, y2 = map(int, largest_person)                              # 가장 큰 사람의 좌표를 정수형으로 변환

        cropped_img = img[y1:y2, x1:x2]                                        # 바운딩 박스를 기준으로 이미지를 잘라냄
        prepro_img = resize(cropped_img)                                       # 잘린 이미지를 이전에 정의해둔 resize 함수로 비율 유지 + 크기 조정

    else:                                                                      # 이미지에서 사람 객체가 감지되지 않았으면 실행
        prepro_img = resize(img)                                               # resize만 진행

    return prepro_img                                                          # 전처리가 완료된 이미지를 반환
```

1.2.5 모든 이미지를 전처리하는 함수 정의

```
def prepro_img_in_folder(origin_folder, prepro_folder):                        # 함수 이름: prepro_img_in_folder

    image_files = [f for f in os.listdir(origin_folder) if f.endswith('.jpg')]   # 원본 이미지 폴더에서 모든 .jpg 파일의 이름을 리스트로 반환

    for image_file in image_files:                                             # 리스트 안의 이미지 파일들을 하나씩 처리

        input_path = os.path.join(origin_folder, image_file)                   # 원본 이미지가 저장된 경로 설정
        output_path = os.path.join(prepro_folder, image_file)                  # 전처리된 이미지들을 저장할 경로 설정

        preprocessed_image = preprocess(input_path)                            # input_path 경로의 모든 이미지를 함수 preprocess로 전처리
        cv2.imwrite(output_path, preprocessed_image)                           # 전처리된 이미지들을 output_path 경로에 저장

        print(f"{image_file} 전처리 완료")                                      # 셀 출력에 전처리가 완료될 때 마다 출력
```

1.2.6 training_image 전처리 실행

```
prepro_img_in_folder(origin_folder, prepro_folder)                             # training_image 전처리
```

숨겨진 출력 표시

1.2.7 validation_image 전처리 실행

```
# 경로를 validation으로 수정
origin_folder = '/content/drive/MyDrive/Original_Data/validation_image'
prepro_folder = '/content/drive/MyDrive/Preprocessed_Data/prepro_validation'

# validation_image 전처리
prepro_img_in_folder(origin_folder, prepro_folder)
```

숨겨진 출력 표시

1.2.8 분류 폴더 경로 설정

```
before_cate_dir_t = '/content/drive/MyDrive/Preprocessed_Data/prepro_training'                # training
after_cate_dir_t = '/content/drive/MyDrive/Preprocessed_Data/training_category'

before_cate_dir_v = '/content/drive/MyDrive/Preprocessed_Data/prepro_validation'               # validation
after_cate_dir_v = '/content/drive/MyDrive/Preprocessed_Data/validation_category'
```

1.2.9 남성/여성 스타일 딕셔너리 생성

```
categories = {
    "남성": ["metrosexual", "hippie", "normcore", "bold", "hiphop", "ivy", "sportivecasual", "mods"],
    "여성": ["disco", "hippie", "bodyconscious", "sportivecasual", "normcore", "oriental", "kitsch", "athleisure", "lounge",
             "minimal", "genderless", "space", "cityglam", "classic", "lingerie", "ecology", "powersuit", "feminine", "punk",
             "hiphop", "popart", "military", "grunge"]
}
```

1.2.10 '성별_스타일' 폴더를 생성하는 함수 정의

```
def create_folders(base_path, categories):                                    # 함수 이름: create_folders
    for category, styles in categories.items():
        for style in styles:
            combined_folder_path = os.path.join(base_path, f"{category}_{style}")
            if not os.path.exists(combined_folder_path):                      # 경로에 폴더가 존재하지 않으면 실행
                os.makedirs(combined_folder_path)                             # 폴더 생성
```

1.2.11 파일을 복사하는 함수 정의

```
def copy_data(before_dir, after_dir, categories):                             # 함수 이름: copy_data
    for filename in os.listdir(before_dir):
        src_path = os.path.join(before_dir, filename)                         # 분류 전 이미지 파일들의 경로 설정
        copied = False

        category = "남성" if "_M" in filename else "여성"                      # 파일 이름에 M이 있으면 남성, M이 없으면(=W가 있으면) 여성으로

        for style in categories[category]:
            if style in filename.lower():
                dest_path = os.path.join(after_dir, f"{category}_{style}")
                shutil.copy(src_path, dest_path)                              # shutil.copy를 이용해 복사
                copied = True
                print(f"{filename}을(를) {dest_path}로 복사했습니다.")
                break
```

1.2.12 분류 실행

```
create_folders(after_cate_dir_t, categories)
create_folders(after_cate_dir_v, categories)

copy_data(before_cate_dir_t, after_cate_dir_t, categories)                    # training 데이터 분류
copy_data(before_cate_dir_v, after_cate_dir_v, categories)                    # validation 데이터 분류
```

숨겨진 출력 표시

1.2.13 ResNet-18 디바이스 설정 및 모듈 시드 설정

```
import torch                                                    # 딥러닝, 데이터 처리, 모델 학습, 시각화에 필요한 라이브러리 불러오
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import os
import random
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

seed = 42                                                       # 시드 설정
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
torch.backends.cudnn.deterministic = True                       # 연산의 결정론적 수행 보장
torch.backends.cudnn.benchmark = False                          # CUDA의 비결정론적 최적화 비활성화

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")   # GPU를 사용하도록 설정
```

1.2.14 데이터 증강 및 형식 변환

```
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),                          # 무작위로 좌우 반전
    transforms.ToTensor(),                                      # 이미지를 텐서 형식으로 변환
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])   # 정규화
])
```

1.2.15 데이터셋 및 경로 설정

```
train_data_path = '/content/drive/MyDrive/Preprocessed_Data/training_category'          # 성별_스타일 별로 분류된 train 데이터
val_data_path = '/content/drive/MyDrive/Preprocessed_Data/validation_category'          # 성별_스타일 별로 분류된 validation 데

class ImageFolderWithPaths(datasets.ImageFolder):                                       # ImageFolder에서 파일 이름 추출을 위해
    def __getitem__(self, index):
        original_tuple = super(ImageFolderWithPaths, self).__getitem__(index)
        path = self.imgs[index][0]                                                      # 이미지 경로
        return original_tuple + (path,)                                                 # 이미지와 라벨, 경로 반환


# Training 및 Validation 데이터 로더를 ImageFolderWithPaths로 설정
train_dataset_with_paths = ImageFolderWithPaths(root=train_data_path, transform=transform)
train_loader_with_paths = DataLoader(train_dataset_with_paths, batch_size=32, shuffle=True, num_workers=2)

val_dataset_with_paths = ImageFolderWithPaths(root=val_data_path, transform=transform)
val_loader_with_paths = DataLoader(val_dataset_with_paths, batch_size=32, shuffle=False, num_workers=2)
```

1.2.16 ResNet-18 모델 정의

```
class CustomResNet18(nn.Module):
    def __init__(self, num_classes):                                    #  __init__ 함수 생성 : 클래스의 인스턴스를 생성할 때 호출
        super(CustomResNet18, self).__init__()
        self.resnet = models.resnet18(pretrained=False)                 # 무작위 초기화 (사전 학습 없음)
        in_features = self.resnet.fc.in_features                        # 원래의 출력 레이어의 입력 크기
        self.resnet.fc = nn.Linear(in_features, num_classes)            # 마지막 레이어를 클래스 수에 맞게 수정
        self.feature_extractor = nn.Sequential(*list(self.resnet.children())[:-1])      # 특징 추출기 생성

    def forward(self, x):                                               # forward 함수 생성 :  입력 데이터를 모델에 전달하는 순전파를
        features = self.feature_extractor(x)                            # 입력 데이터를 특징 추출기에 전달
        features = features.view(features.size(0), -1)                  # 2차원 형태로 변환
        output = self.resnet.fc(features)                               # 클래스별 출력값 생성
        return features, output                                         # 추출된 특징 벡터와 최종 분류 결과를 반환

model = CustomResNet18(num_classes=len(train_dataset_with_paths.classes)).to(device)    # 모델 생성 및 device로 이동
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and ma
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights
  warnings.warn(msg)
```

1.2.17 손실 함수 및 옵티마이저 설정

```
criterion = nn.CrossEntropyLoss()                                   # 다중 클래스 분류 문제를 위한 손실 함수
optimizer = optim.Adam(model.parameters(), lr=0.001)                # Adam 옵티마이저 설정 (학습률 0.001)
```

1.2.18 학습 함수 정의

```
def train(model, train_loader, criterion, optimizer, device):       # train 함수 생성 : 각 에포크 동안 학습 데이터의 여러 배치를 사용하
    model.train()                                                   # 학습 모드로 설정
    total_loss = 0                                                  # 초기값 설정
    correct = 0
    total = 0

    for images, labels, _ in train_loader:                          # 학습 데이터셋에서 배치 단위로 데이터 로드
        images, labels = images.to(device), labels.to(device)       # 데이터를 device로 이동

        _, outputs = model(images)                                  # 모델 출력값

        loss = criterion(outputs, labels)                           # 손실 함수 계산
        optimizer.zero_grad()                                       # 이전의 기울기 초기화
        loss.backward()                                             # 역전파를 통해 기울기 계산
        optimizer.step()                                            # 가중치 업데이트

        total_loss += loss.item()                                   # 누적 손실 계산
        _, predicted = torch.max(outputs, 1)                        # 예측된 클래스 계산
        total += labels.size(0)                                     # 전체 샘플 수
        correct += (predicted == labels).sum().item()               # 정확하게 예측한 샘플 수

    train_loss = total_loss / len(train_loader)                     # 평균 학습 손실 계산
    accuracy = 100 * correct / total                                # 정확도 계산

    print(f'Train Loss: {train_loss:.4f}, Accuracy: {accuracy:.2f}%')  # 손실과 정확도 출력
    return train_loss                                               # train_loss 반환
```

1.2.19 검증 함수 정의

```
def validate(model, val_loader, criterion, device):                 # validate 함수 생성 : 각 에포크 동안 검증 데이터의 배치를 사용해 모
    model.eval()                                                    # 평가 모드로 설정
    total_loss = 0                                                  # 초기값 설정
    correct = 0
    total = 0

    with torch.no_grad():                                           # 검증 시에는 역전파 계산을 하지 않음
        for images, labels, _ in val_loader:                        # 검증 데이터셋에서 배치 단위로 데이터 로드
            images, labels = images.to(device), labels.to(device)   # 데이터를 device로 이동

            _, outputs = model(images)                              # 모델 출력값

            loss = criterion(outputs, labels)                       # 손실 계산

            total_loss += loss.item()                               # 누적 손실 계산
            _, predicted = torch.max(outputs, 1)                    # 예측된 클래스 계산
            total += labels.size(0)                                 # 전체 샘플 수
            correct += (predicted == labels).sum().item()           # 정확하게 예측한 샘플 수

    val_loss = total_loss / len(val_loader)                         # 평균 검증 손실 계
    accuracy = 100 * correct / total                                # 정확도 계산

    print(f'Validation Loss: {val_loss:.4f}, Accuracy: {accuracy:.2f}%')  # 손실과 정확도 출력
    return val_loss, accuracy                                       # 손실과 정확도 반환
```

1.2.20 학습 및 검증 + Early Stopping 추가, 시각화를 위한 데이터 저장

```
# 결과를 저장할 리스트들을 초기화
train_losses = []
val_losses = []
val_accuracies = []

num_epochs = 200                            # 에포크 수 설정
patience = 10                               # Early Stopping 대기 횟수 설정
best_val_loss = float('inf')                # 검증 손실 초기값을 무한대로 설정
early_stop_counter = 0                      # Early Stopping 카운터 초기값 설정

for epoch in range(1, num_epochs + 1):      # 각 에포크 반복
    print(f'Epoch {epoch}/{num_epochs}')    # 에포크 정보 출력
```

```
    train_loss = train(model, train_loader_with_paths, criterion, optimizer, device)                    # 학습 손실 반환
    val_loss, val_accuracy = validate(model, val_loader_with_paths, criterion, device)                  # 검증 손실 및 검증 정확도 반환

    train_losses.append(train_loss)                                                                     # 계산된 학습 손실을 리스트에 저장
    val_losses.append(val_loss)                                                                         # 계산된 검증 손실을 리스트에 저장
    val_accuracies.append(val_accuracy)                                                                 # 계산된 검증 정확도를 리스트에 저장

    if val_loss < best_val_loss:                                                                        # 현재 에포크의 검증 손실이 최저 손실보다 작
        best_val_loss = val_loss
        early_stop_counter = 0
        torch.save(model.state_dict(), '/content/drive/MyDrive/Mission 3-2/resnet18_Real.pth')          # 모델의 가중치 저장
        print(f"Validation loss improved, saving model...")
    else:                                                                                               # 검증 손실이 개선되지 않았으면 실행
        early_stop_counter += 1                                                                         # Early Stopping 카운터 +1
        print(f"Validation loss did not improve. Early stop counter: {early_stop_counter}/{patience}")

    if early_stop_counter >= patience:                                                                  # Early Stopping 카운터가 patience 설정값 C
        print("Early stopping triggered. Stopping training.")
        break                                                                                           # 학습 중단

torch.save(model.state_dict(), '/content/drive/MyDrive/Mission 3-2/resnet18.pth')        # 최종 모델의 가중치 저장
```

```
Validation loss did not improve. Early stop counter: 2/10
Epoch 21/200
Train Loss: 1.5912, Accuracy: 49.68%
Validation Loss: 2.3984, Accuracy: 37.75%
Validation loss improved, saving model...
Epoch 22/200
Train Loss: 1.3090, Accuracy: 58.65%
Validation Loss: 2.7608, Accuracy: 35.44%
Validation loss did not improve. Early stop counter: 1/10
Epoch 23/200
Train Loss: 1.0434, Accuracy: 68.60%
Validation Loss: 2.6738, Accuracy: 42.17%
Validation loss did not improve. Early stop counter: 2/10
Epoch 24/200
Train Loss: 0.8419, Accuracy: 75.23%
Validation Loss: 2.3572, Accuracy: 49.42%
Validation loss improved, saving model...
Epoch 25/200
Train Loss: 0.5980, Accuracy: 82.14%
Validation Loss: 2.4090, Accuracy: 53.52%
Validation loss did not improve. Early stop counter: 1/10
Epoch 26/200
Train Loss: 0.4533, Accuracy: 87.15%
Validation Loss: 2.4773, Accuracy: 53.31%
Validation loss did not improve. Early stop counter: 2/10
Epoch 27/200
Train Loss: 0.4074, Accuracy: 88.85%
Validation Loss: 2.4992, Accuracy: 56.57%
Validation loss did not improve. Early stop counter: 3/10
Epoch 28/200
Train Loss: 0.2925, Accuracy: 92.19%
Validation Loss: 2.4566, Accuracy: 58.99%
Validation loss did not improve. Early stop counter: 4/10
Epoch 29/200
Train Loss: 0.2622, Accuracy: 93.14%
Validation Loss: 2.5081, Accuracy: 59.83%
Validation loss did not improve. Early stop counter: 5/10
Epoch 30/200
Train Loss: 0.1809, Accuracy: 95.68%
Validation Loss: 2.6044, Accuracy: 59.94%
Validation loss did not improve. Early stop counter: 6/10
Epoch 31/200
Train Loss: 0.1388, Accuracy: 96.90%
Validation Loss: 2.4900, Accuracy: 60.78%
Validation loss did not improve. Early stop counter: 7/10
Epoch 32/200
Train Loss: 0.1360, Accuracy: 96.76%
Validation Loss: 2.5081, Accuracy: 60.25%
Validation loss did not improve. Early stop counter: 8/10
Epoch 33/200
Train Loss: 0.1001, Accuracy: 98.08%
Validation Loss: 2.5290, Accuracy: 61.72%
Validation loss did not improve. Early stop counter: 9/10
Epoch 34/200
Train Loss: 0.0927, Accuracy: 98.30%
Validation Loss: 2.5024, Accuracy: 61.83%
Validation loss did not improve. Early stop counter: 10/10
Early stopping triggered. Stopping training.
```

1.2.21 결과값 시각화

```
sns.set(style="whitegrid")                          # 시각화를 위한 스타일 설정

epochs = range(1, len(train_losses) + 1)            # 에포크 범위 설정
```

```
plt.figure(figsize=(14, 5))                                  # 전체 그래프의 크기를 너비 14, 높이 5로 설정

# 훈련 및 검증 손실 그래프
plt.subplot(1, 2, 1)
plt.plot(epochs, train_losses, label='Train Loss')           # 학습 손실에 대한 그래프
plt.plot(epochs, val_losses, label='Validation Loss')        # 검증 손실에 대한 그래프
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Train and Validation Loss')
plt.legend()

# 검증 정확도 그래프
plt.subplot(1, 2, 2)
plt.plot(epochs, val_accuracies, label='Validation Accuracy')    # 검증 정확도에 대한 그래프
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Validation Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
plt.close()
```