

```
from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

▼ Mission 1: 패션 스타일 이미지 분류

1-1. 주어진 이미지 데이터의 파일명은 아래와 같은 형식이다.

```
'{W/T}_{이미지ID}_{시대별}_{스타일별}_{성별}.jpg'
```

이에 기반하여 "이미지ID" 수 기준으로 성별 & 스타일 통계를 도출한다.

```
import os
# 파일 시스템을 탐색하여 디렉토리 내 파일 목록을 가져오는데 사용
import pandas as pd
# 통계 데이터를 표 형태로 저장하고 출력하는 데 사용
from collections import defaultdict
# 중첩된 딕셔너리 형태로 데이터를 쉽게 저장하기 위해 사용

# 이미지 파일명에서 성별 및 스타일을 추출하는 함수
def extract_info_from_filename(filename):
    # 파일명 예시: "W_00237_60_popart_W.jpg"
    #           [0] [1] [2] [3] [4]
    parts = filename.split("_")
    if len(parts) < 4:
        return None, None
    # 형식이 맞지 않는 파일명은 무시
    style = parts[3]
    # 스타일 정보는 파일명의 구분 중 앞에서 네 번째 요소
    gender = '여성' if parts[-1].startswith('W') else '남성'
    # 성별 정보는 파일명의 구분 중 뒤에서 첫 번째 요소
    return gender, style

# 디렉토리 내 파일명으로 통계 정보를 추출하는 함수
def generate_statistics(directory):
    # 성별 & 스타일별 이미지 수를 저장할 딕셔너리
    stats = defaultdict(lambda: defaultdict(int))

    # 디렉토리 내 모든 파일명에 대해 성별과 스타일 정보 추출
    for filename in os.listdir(directory):
        if filename.endswith(".jpg"):
            gender, style = extract_info_from_filename(filename)
            if gender and style:
                stats[gender][style] += 1

    # 통계 정보를 DataFrame으로 변환
    stats_list = []
    for gender, style_dict in stats.items():
        for style, count in style_dict.items():
            stats_list.append((gender, style, count))

    stats_df = pd.DataFrame(stats_list, columns=['성별', '스타일', '이미지 수'])
    return stats_df.head(1000)
# 이미지 수의 내림차순으로 데이터프레임 출력

# Training 및 Validation 데이터 경로

# /PATH/TO/ 부분을 실사 환경에 맞게 수정해주세요
training_image_dir = '/PATH/TO/Dataset/training_image'
validation_image_dir = '/PATH/TO/Dataset/validation_image'

# Training Image 데이터 통계 도출
generate_statistics(training_image_dir)

# Validation Image 데이터 통계 도출
generate_statistics(validation_image_dir)
```

	성별	스타일	이미지 수
0	남성	hippie	260
1	남성	normcore	364
2	남성	mods	269
3	남성	sportivecasual	298
4	남성	ivy	237
5	남성	hiphop	274
6	남성	metrosexual	278
7	남성	bold	268
8	여성	sportivecasual	157
9	여성	powersuit	120
10	여성	feminine	154
11	여성	oriental	78
12	여성	normcore	153
13	여성	popart	41
14	여성	classic	77
15	여성	punk	65
16	여성	hiphop	48
17	여성	hippie	91
18	여성	kitsch	91
19	여성	ecology	64
20	여성	minimal	139
21	여성	lingerie	55
22	여성	disco	37
23	여성	cityglam	67
24	여성	space	37
25	여성	genderless	77
26	여성	lounge	45
27	여성	bodyconscious	95
28	여성	athleisure	67
29	여성	grunge	31
30	여성	military	33

	성별	스타일	이미지 수
0	여성	oriental	18
1	여성	cityglam	18
2	여성	military	9
3	여성	grunge	10
4	여성	genderless	12
5	여성	powersuit	34
6	여성	kitsch	22
7	여성	sportivecasual	48
8	여성	feminine	44
9	여성	hippie	14
10	여성	bodyconscious	23
11	여성	minimal	35
12	여성	space	15
13	여성	athleisure	14
14	여성	classic	22
15	여성	lingerie	5
16	여성	punk	12
17	여성	disco	10
18	여성	normcore	20
19	여성	ecology	17
20	여성	popart	8
21	여성	hiphop	8
22	여성	lounge	8
23	남성	hippie	82
24	남성	mods	80
25	남성	hiphop	66
26	남성	ivy	79
27	남성	normcore	51
28	남성	bold	57
29	남성	metrosexual	58
30	남성	sportivecasual	52

▼ 1-2. ResNet-18를 활용하여 “성별 & 스타일” 단위로 클래스 분류를 수행하고 Validation 데이터에 대한 정확도를 제시한다.

- ResNet-18의 parameters는 무작위로 초기화하여 사용한다.(즉, pretrained weights는 사용할 수 없음)
- 성능을 높이기 위해 object detection, image cropping 등의 다양한 데이터 전처리 기법을 활용해도 무방하다. (데이터 전처리 단계에 한해서 는 외부 라이브러리 활용 가능)

```
import torch
import os
import numpy as np
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms, models
from PIL import Image
import torch.optim as optim
import torch.nn as nn
import warnings
import matplotlib.pyplot as plt
```

```
# FutureWarning 무시
warnings.filterwarnings("ignore", category=FutureWarning)
```

```
# GPU 사용 설정
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
if torch.cuda.is_available():
    print(f'Using GPU: {torch.cuda.get_device_name(0)}')
else:
    print("Using CPU")
```

🔄 Using GPU: Tesla T4

```
# Custom Dataset Class 정의
class GenderStyleDataset(Dataset):
    def __init__(self, image_folder, label_mapping, transform=None):
        self.image_folder = image_folder
        self.image_files = [f for f in os.listdir(image_folder) if f.endswith('.jpg')]
        self.transform = transform
        self.label_mapping = label_mapping

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

    def __getitem__(self, idx):
        img_name = self.image_files[idx]
        img_path = os.path.join(self.image_folder, img_name)
        image = Image.open(img_path)

        # 파일명에서 성별과 스타일 추출
        parts = img_name.split('.')
        style = parts[3] # 예: sportivecasual
        gender_code = parts[-1][0] # M or W

        if gender_code == 'M':
            gender = 'man'
        elif gender_code == 'W':
            gender = 'woman'

        # 성별 & 스타일을 결합한 클래스 라벨
        class_label = f'{gender}_{style}'

        # Transform 적용
        if self.transform:
            image = self.transform(image)

        # 문자열 라벨을 숫자 라벨로 변환
        label = self.label_mapping[class_label]

        return image, label
```

```
# 라벨 인덱스 매핑 생성
def create_label_mapping(image_folder):
    label_set = set()
    for filename in os.listdir(image_folder):
        parts = filename.split('.')
        style = parts[3]
        gender_code = parts[-1][0]
```

```
if gender_code == 'M':
    gender = 'man'
elif gender_code == 'W':
    gender = 'woman'
class_label = f'{gender}_{style}'
label_set.add(class_label)
label_mapping = {label: idx for idx, label in enumerate(sorted(label_set))}
return label_mapping
```

데이터 경로 및 변환 설정, RemBG Output 90으로 나오게 된 것을 대상

```
# /PATH/TO/ 부분을 실사 환경에 맞게 수정해주세요
train_output_folder = '/PATH/TO/Dataset/bg_remove/training_image_no_bg'
val_output_folder = '/PATH/TO/Dataset/bg_remove/validation_image_no_bg'
```

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

```
# 라벨 매핑 생성
label_mapping = create_label_mapping(train_output_folder)
num_classes = len(label_mapping)
```

```
# 학습 루프 및 검증 루프
num_epochs = 50 # 에폭 수 설정
batch_size = 128
learning_rate = 0.001
train_losses = []
val_losses = []
val_accuracies = []
```


```
# 데이터셋 및 DataLoader 생성
train_dataset = GenderStyleDataset(image_folder=train_output_folder, label_mapping=label_mapping, transform=transform)
val_dataset = GenderStyleDataset(image_folder=val_output_folder, label_mapping=label_mapping, transform=transform)
```

train_loader = DataLoader(train_dataset, batch_size, shuffle=True, num_workers=6, pin_memory=True) #이 부분에서 batch_size와 num_workers를 수정하며 진행
val_loader = DataLoader(val_dataset, batch_size, shuffle=False, num_workers=6, pin_memory=True)

```
# 데이터셋 및 DataLoader 생성
train_dataset = GenderStyleDataset(image_folder=train_output_folder, label_mapping=label_mapping, transform=transform)
val_dataset = GenderStyleDataset(image_folder=val_output_folder, label_mapping=label_mapping, transform=transform)
```

train_loader = DataLoader(train_dataset, batch_size, shuffle=True, num_workers=6, pin_memory=True) #이 부분에서 batch_size와 num_workers를 수정하며 진행
val_loader = DataLoader(val_dataset, batch_size, shuffle=False, num_workers=6, pin_memory=True)

```
# ResNet-18 모델 정의 및 GPU로 전송
model = models.resnet18(pretrained=False)
model.fc = nn.Linear(model.fc.in_features, num_classes) # 마지막 레이어 수정
model = model.to(device)
```

 /usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
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' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to pass
warnings.warn(msg)

```
# 손실 함수와 옵티마이저 설정
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), learning_rate) #이 부분에서도 lr을 수정해가며 진행
```

```
for epoch in range(num_epochs):
    # 에폭 시작 정보 출력
    print(f'Epoch {epoch + 1}/{num_epochs} 시작')

    # 학습 단계
    model.train()
    running_loss = 0.0
    for i, (inputs, labels) in enumerate(train_loader):
        inputs, labels = inputs.to(device), labels.to(device) # GPU로 전송

        # Forward pass
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)

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

        running_loss += loss.item()

    avg_train_loss = running_loss / len(train_loader)
    train_losses.append(avg_train_loss)
    print(f'Epoch [{epoch + 1}/{num_epochs}] 학습 완료, Average Loss: {avg_train_loss:.4f}')
```

```
# 검증 단계
model.eval() # 모델 평가 모드로 설정
correct = 0
total = 0
val_loss = 0.0
with torch.no_grad():
    for val_inputs, val_labels in val_loader:
        val_inputs, val_labels = val_inputs.to(device), val_labels.to(device) # GPU로 전송

        val_outputs = model(val_inputs)
        val_loss += criterion(val_outputs, val_labels).item()

        _, predicted = torch.max(val_outputs, 1)
        total += val_labels.size(0)
        correct += (predicted == val_labels).sum().item()

avg_val_loss = val_loss / len(val_loader)
val_accuracy = 100 * correct / total
val_losses.append(avg_val_loss)
val_accuracies.append(val_accuracy)
print(f'Validation Loss: {avg_val_loss:.4f}, Validation Accuracy: {val_accuracy:.2f}%')
```

```
# 다시 학습 모드로 전환
model.train()
```

 Epoch 1/50 시작
Epoch [1/50] 학습 완료, Average Loss: 3.2107
Validation Loss: 3.8030, Validation Accuracy: 5.36%
Epoch 2/50 시작
Epoch [2/50] 학습 완료, Average Loss: 3.0484
Validation Loss: 3.3185, Validation Accuracy: 10.52%
Epoch 3/50 시작
Epoch [3/50] 학습 완료, Average Loss: 2.9448
Validation Loss: 4.6742, Validation Accuracy: 8.41%
Epoch 4/50 시작
Epoch [4/50] 학습 완료, Average Loss: 2.8418
Validation Loss: 4.5283, Validation Accuracy: 7.66%
Epoch 5/50 시작
Epoch [5/50] 학습 완료, Average Loss: 2.7282
Validation Loss: 3.1028, Validation Accuracy: 13.46%
Epoch 6/50 시작
Epoch [6/50] 학습 완료, Average Loss: 2.5921

```
Validation Loss: 3.8571, Validation Accuracy: 9.46%
Epoch 7/50 시작
Epoch [7/50] 학습 완료, Average Loss: 2.4487
Validation Loss: 2.7849, Validation Accuracy: 21.24%
Epoch 8/50 시작
Epoch [8/50] 학습 완료, Average Loss: 2.2563
Validation Loss: 3.3357, Validation Accuracy: 11.57%
Epoch 9/50 시작
Epoch [9/50] 학습 완료, Average Loss: 2.0441
Validation Loss: 5.0088, Validation Accuracy: 10.09%
Epoch 10/50 시작
Epoch [10/50] 학습 완료, Average Loss: 1.7123
Validation Loss: 10.5202, Validation Accuracy: 10.62%
Epoch 11/50 시작
Epoch [11/50] 학습 완료, Average Loss: 1.3117
Validation Loss: 4.3981, Validation Accuracy: 21.03%
Epoch 12/50 시작
Epoch [12/50] 학습 완료, Average Loss: 0.8409
Validation Loss: 3.4838, Validation Accuracy: 26.92%
Epoch 13/50 시작
Epoch [13/50] 학습 완료, Average Loss: 0.4419
Validation Loss: 3.2433, Validation Accuracy: 34.17%
Epoch 14/50 시작
Epoch [14/50] 학습 완료, Average Loss: 0.2161
Validation Loss: 2.4907, Validation Accuracy: 51.95%
Epoch 15/50 시작
Epoch [15/50] 학습 완료, Average Loss: 0.1045
Validation Loss: 2.0603, Validation Accuracy: 56.57%
Epoch 16/50 시작
Epoch [16/50] 학습 완료, Average Loss: 0.0573
Validation Loss: 1.9627, Validation Accuracy: 62.04%
Epoch 17/50 시작
Epoch [17/50] 학습 완료, Average Loss: 0.0358
Validation Loss: 1.8747, Validation Accuracy: 60.99%
Epoch 18/50 시작
Epoch [18/50] 학습 완료, Average Loss: 0.0286
Validation Loss: 2.0286, Validation Accuracy: 62.46%
Epoch 19/50 시작
Epoch [19/50] 학습 완료, Average Loss: 0.0232
Validation Loss: 1.8876, Validation Accuracy: 60.99%
```

```
# 프리 트레인드 모델 저장
# /PATH/TO/ 부분을 임시 환경에 맞게 수정해주세요
torch.save(model.state_dict(), '/PATH/TO/Mission1/resnet18_gender_style_pretrained.pth') # 학습된 가중치 저장
# 'resnet18_gender_style_pretrained.pth' 파일에 학습된 가중치를 저장하여 이후 재사용할 수 있도록 함.
```

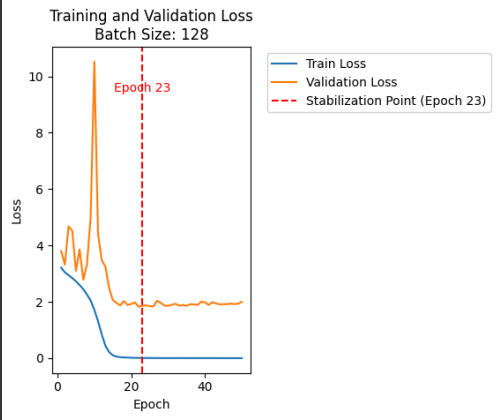
```
# 안정된 예측 설정 (평균 손실 값의 변화가 미미한 시점을 자동으로 판별)
stabilization_epoch = next((epoch for epoch in range(1, num_epochs) if abs(train_losses[epoch] - train_losses[epoch - 1]) < 0.025 and abs(val_losses[epoch] - val_losses[epoch - 1]) < 0.025), num_epochs)
```

```
# Correcting the variable names and plotting the graphs with legends outside and including stabilization epoch value on the vertical line
plt.figure(figsize=(12, 5))
```

```
<Figure size 1200x500 with 0 Axes>
<Figure size 1200x500 with 0 Axes>
```

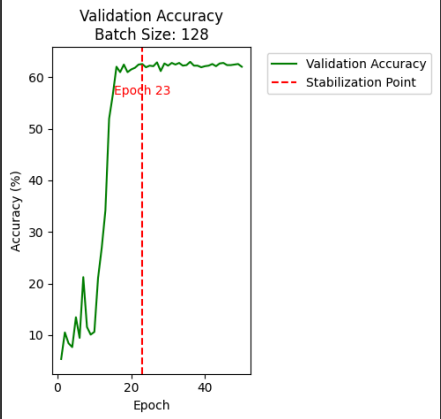
```
# Loss Graph
plt.subplot(1, 2, 1)
plt.plot(range(1, len(train_losses) + 1), train_losses, label='Train Loss')
plt.plot(range(1, len(val_losses) + 1), val_losses, label='Validation Loss')
plt.axvline(x=stabilization_epoch, color='red', linestyle='--', label=f'Stabilization Point (Epoch {stabilization_epoch})')
plt.text(stabilization_epoch, max(max(train_losses), max(val_losses)) * 0.9, f'Epoch {stabilization_epoch}', color='red', ha='center')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title(f'Training and Validation Loss\nBatch Size: 128')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
```

```
<matplotlib.legend.Legend at 0x7b3912db410>
```



```
# Accuracy Graph
plt.subplot(1, 2, 2)
plt.plot(range(1, len(val_accuracies) + 1), val_accuracies, label='Validation Accuracy', color='green')
plt.axvline(x=stabilization_epoch, color='red', linestyle='--', label='Stabilization Point')
plt.text(stabilization_epoch, max(val_accuracies) * 0.9, f'Epoch {stabilization_epoch}', color='red', ha='center')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title(f'Validation Accuracy\nBatch Size: 128')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
```

```
<matplotlib.legend.Legend at 0x7b382c8d8550>
```



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

