2024 데이터 크리에이터 캠프

문제: 인공지능은 사람의 마음을 이해할수 있을까?

Mission3. 패션스타일 선호 여부 예측

Mission 3-1.

추천 시스템의 기본인 협업 필터링 (Collaborative Filtering)은 크게 user-based filtering,
item-based filtering 방식으로 나뉘어져 있다. 각각에 대해서 이해하고, 2-2에서 구해 본 응답자의
"스타일 선호 정보표"를 토대로 Validation 데이터 내 응답자의 "스타일 선호 여부 예측" 문제를
2가지 기법으로 어떻게 적용해 볼 수 있고, 서로 비교하여 어떤 장단점을 갖는지 설명한다.
※ 설명을 용이하게 하기 위해 응답자의 스타일 선호도 예시를 들어서 설명해도 무방하다.

응답자의 "스타일 선호 정보표"를 토대로 validation 데이터 내 응답자의 스타일 선호 여부를
예측해볼 수 있습니다. 먼저 item based filtering을 적용한 방식에 대해 설명하겠습니다.

우선 평가가 이루어지지 않은 스타일에 대해서는 전부 제거해줍니다. 예를 들어 응답자 A는 비선호하는
스타일만 있을 뿐 선호하는 아이템이 없다면 선호하는 스타일이 있는지 확인해볼 수 없기 때문에
결측치는 전부 제거해준 뒤에 예측을 진행합니다.

다음으로 각 스타일의 특성을 고려한 벡터값을 resnet18 모델에 이미지를 넣어 구합니다. 이렇게 구한
특징 벡터는 스타일 간의 유사도를 계산하는데 사용됩니다. 각 응답자마다 선호, 비선호하는 스타일이
나뉘어져 있을 뿐만 아니라 선호하는 스타일의 수 또한 상이합니다. 어떤 응답자는 선호하는 스타일이 하나만
있거나 2개 이상의 아이템을 선호하는 사용자가 있기도 합니다. 또는 비선호하는 스타일이 여러개인 응답자도
존재한 반면 비선호하는 스타일이 하나만 있는 경우도 존재합니다.

이렇게 선호, 비선호하는 스타일의 수가 제각기 다르기 때문에 이를 하나의 특성 벡터로 만들어줍니다.
이 때 평균을 활용해서 응답자마다 선호, 비선호하는 스타일의 특징 벡터값을 도출합니다.

이제 마지막 단계로 validation 데이터의 특징 벡터값과 training 데이터의 특징 벡터값의 유사도 계산을
한 후에 선호하는 스타일과 유사도가 높으면 선호로 예측하고 비선호하는 스타일과 유사도가 높으면 비선호로
예측하는 방식으로 스타일 선호 여부를 예측합니다. 유사도 임계치는 사용자 임의로 설정할 수 있습니다.

마지막으로 예측한 값은 실제 정답과 비교를 통해 얼마나 정확하게 예측했는지는 confusion matrix에서
accuracty 값으로 확인할 수 있습니다. user-based filtering을 적용한 방식에 대해 설명하겠습니다.

우선, item-based filtering과 마찬가지로 해당 사용자가 평가하지 않은 아이템은 제거합니다.
다음으로, 각 사용자 간의 유사성을 분석하기 위해 사용자 간의 평가 데이터를 기반으로 유사도 매트릭스를 생성합니다.
이 과정에서는 코사인 유사도, 피어슨 상관계수 등의 방법을 사용하여 사용자의 선호 패턴을 비교합니다.
이를 통해 비슷한 취향을 가진 사용자 그룹을 식별할 수 있습니다.

각 사용자의 선호도는 다를 수 있으며, 이에 따라 선호하는 아이템의 수 또한 차이가 있습니다.
어떤 사용자는 선호하는 아이템이 많을 수도 있고, 어떤 사용자는 적을 수도 있습니다.
이러한 다양한 선호도를 반영하기 위해, 유사한 사용자들의 선호도를 평균하여 특정 사용자에 대한 예측을 생성합니다.
예를 들어, 유사한 사용자들이 높은 평점을 준 아이템은 해당 사용자에게도 추천될 가능성이 높습니다.

이제 마지막 단계로, validation 데이터에서 각 사용자의 선호도 벡터와 training 데이터에서 도출한
사용자 벡터 간의 유사도를 계산합니다. 유사도가 높은 사용자들로부터 추천 아이템을 선택하여, 해당 아이템이
선호될 가능성이 높은지 예측합니다. 이때 유사도 임계치는 사용자가 임의로 설정할 수 있습니다.

마지막으로, 예측한 값은 실제 정답과 비교하여 얼마나 정확하게 예측했는지를 confusion matrix를
통해 accuracy 값으로 확인할 수 있습니다.

사용자 기반 협업 필터링과 아이템 기반 협업 필터링을 비교해보자면 우선 사용자 기반 협업 필터링의 경우
각 사용자의 선호도를 기반으로 추천을 제공하므로, 사용자 개개인에 맞춘 개인화된 결과를 생성할 수 있습니다.
또한 유사한 취향을 가진 여러 사용자들의 정보를 활용하므로 다양한 아이템을 추천할 수 있습니다.
그러나 사용자 수가 너무 많고 각 사용자가 평가한 아이템의 수가 너무 적을 경우 유사한 사용자 찾기가 어려워
추천의 질이 떨어질 수 있으며 사용자 수가 늘어날수록 계산 복잡도가 높아져 성능이 저하될 수 있습니다.

반면 아이템 기반 협업 필터링은 아이템 간의 유사성을 기반으로 하기 때문에, 사용자 수가 적더라도 추천이 가능합니다.
또한 사용자가 선호하는 아이템과 유사한 아이템을 추천하기 때문에 사용자는 쉽게 이해하고 수용할 수 있습니다.
단점으로는 아이템의 특성에만 집중하기 때문에 개별 사용자의 취향을 충분히 반영하지 못한다는 점이 있습니다.
또한 사용자 취향이 변화할 경우 아이템 간의 유사성 만으로는 추천을 하기 어려울 수 있습니다.

Mission 3-2.

3-1에서 살펴 본 기법 중, item-based filtering을 직접 구현해본다. "이미지 간 유사도" (image2image)만을 활용하여
Validation 데이터 내 응답자의 "스타일 선호 여부 예측" 문제를 수행하고 성능을 측정한다.
※ Hint. 1-2에서 학습한 ResNet-18의 중간 layer 값을 활용하여 각 이미지의 feature vector를 구하고,

벡터 연산을 통해 이미지 간 유사도를 구해볼 수 있다.
※ 예측 문제에서 활용한 파라미터 및 임계 값 등의 수치를 정확하게 제시한다.

라이브러리 불러오기

In [53]:
```
import os
import torch
import json
import numpy as np
import pandas as pd
from torch import Tensor
import torch.nn as nn
import torch.optim as optim
import torchvision.models as models
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
from PIL import Image
from sklearn.metrics import accuracy_score
from typing import Type
from collections import defaultdict
import torchvision.transforms as transforms
from sklearn.metrics.pairwise import cosine_similarity
```

Resnet

In [54]:
```
class BasicBlock(nn.Module):
    def __init__(
        self,
        in_channels: int,
        out_channels: int,
        stride: int = 1,
        expansion: int = 1,
        downsample: nn.Module = None
    ) -> None:
        super(BasicBlock, self).__init__()
        self.expansion = expansion
        self.downsample = downsample
        self.conv1 = nn.Conv2d(
            in_channels,
            out_channels,
            kernel_size=3,
            stride=stride,
            padding=1,
            bias=False
        )
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(
            out_channels,
            out_channels*self.expansion,
            kernel_size=3,
            padding=1,
            bias=False
        )
        self.bn2 = nn.BatchNorm2d(out_channels*self.expansion)

    def forward(self, x: Tensor) -> Tensor:
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)
        return  out
```

In [55]:
```
class ResNet(nn.Module):
    def __init__(
        self,
        img_channels: int,
        num_layers: int,
        block: Type[BasicBlock],
        num_classes: int  = 1000
    ) -> None:
        super(ResNet, self).__init__()
        if num_layers == 18: # ResNet18 만을 본 대회에서 사용함으로 18층만 구현
            layers = [2, 2, 2, 2]
            self.expansion = 1

        self.in_channels = 64
```

```
        self.conv1 = nn.Conv2d(
            in_channels=img_channels,
            out_channels=self.in_channels,
            kernel_size=7,
            stride=2,
            padding=3,
            bias=False
        )
        self.bn1 = nn.BatchNorm2d(self.in_channels)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512*self.expansion, num_classes)

    def _make_layer(
        self,
        block: Type[BasicBlock],
        out_channels: int,
        blocks: int,
        stride: int = 1
    ) -> nn.Sequential:
        downsample = None
        if stride != 1:
            downsample = nn.Sequential(
                nn.Conv2d(
                    self.in_channels,
                    out_channels*self.expansion,
                    kernel_size=1,
                    stride=stride,
                    bias=False
                ),
                nn.BatchNorm2d(out_channels * self.expansion),
            )
        layers = []
        layers.append(
            block(
                self.in_channels, out_channels, stride, self.expansion, downsample
            )
        )
        self.in_channels = out_channels * self.expansion

        for i in range(1, blocks):
            layers.append(block(
                self.in_channels,
                out_channels,
                expansion=self.expansion
            ))
        return nn.Sequential(*layers)

    def forward(self, x: Tensor) -> Tensor:
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        # print('Dimensions of the last convolutional feature map: ', x.shape)

        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)

        return x
```

ResNet-18 모델 정의

In [56]:
```
# ResNet-18 모델 정의
class ResNet18FeatureExtractor(nn.Module):
    def __init__(self):
        super(ResNet18FeatureExtractor, self).__init__()
        self.resnet18 = models.resnet18(pretrained=True) # 사전 학습된 가중치
        self.features = nn.Sequential(*list(self.resnet18.children())[:-1])  # 마지막 FC 레이어 제외

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)  # Flatten
        return x
```

In [57]:
```
# CSV 파일 불러오기
mission2_result = pd.read_csv('../dataset/mission2-2_result_all.csv')
```

```
# 데이터프레임의 일부 출력
mission2_result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3480 entries, 0 to 3479
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   응답자 ID     3480 non-null   int64 
 1   train 선호    2215 non-null   object
 2   train 비선호   2830 non-null   object
 3   valid 선호    1048 non-null   object
 4   valid 비선호   1354 non-null   object
dtypes: int64(1), object(4)
memory usage: 136.1+ KB
```

In [58]:
```
mission2_result.head()
```

Out[58]:

	응답자 ID	train 선호	train 비선호	valid 선호	valid 비선호
0	58049	NaN	T_00253_60_popart_W.jpg	NaN	T_00253_60_popart_W.jpg
1	62192	NaN	T_00253_60_popart_W.jpg	NaN	T_00253_60_popart_W.jpg
2	64213	NaN	T_00253_60_popart_W.jpg	NaN	T_00253_60_popart_W.jpg
3	66592	T_00253_60_popart_W.jpg, T_00893_90_hiphop_W.j...	T_07452_50_classic_W.jpg, W_02170_50_feminine_...	T_00253_60_popart_W.jpg, W_10028_50_classic_W....	W_02170_50_feminine_W.jpg, W_19352_50_feminine...
4	66721	W_05960_70_hippie_W.jpg	T_00253_60_popart_W.jpg	NaN	T_00253_60_popart_W.jpg

- train 데이터에서 선호/비선호 여부에 따라 데이터 있는경우에 추천이 가능하다고 판단하였고
- valid 데이터의 또한 선호/비선호 데이터가 모두 있는 경우를 사용
  -> valid 데이터의 경우 선호/비선호 값에 속하는 이미지가 모두 존재해야되지는 않지만 통일성을 위해 데이터가 모두 존재하는 상황에서 아이템 기반 필터링(Item-Based Filtering) 방식 적용

In [59]:
```
# 모든 열의 값이 채워져 있는 행만 선택
filtered_mission2_result = mission2_result.dropna()

# 인덱스 재설정
filtered_mission2_result = filtered_mission2_result.reset_index(drop=True)

# 결과 출력
filtered_mission2_result
```

Out[59]:

	응답자 ID	train 선호	train 비선호	valid 선호	valid 비선호
0	66592	T_00253_60_popart_W.jpg, T_00893_90_hiphop_W.j...	T_07452_50_classic_W.jpg, W_02170_50_feminine_...	T_00253_60_popart_W.jpg, W_10028_50_classic_W....	W_02170_50_feminine_W.jpg, W_19352_50_feminine...
1	66469	T_00456_10_sportivecasual_M.jpg, T_00588_10_sp...	T_02958_19_normcore_M.jpg, T_06076_60_mods_M.j...	T_00456_10_sportivecasual_M.jpg, T_01123_90_hi...	W_24553_70_hippie_M.jpg, W_24647_70_hippie_M.j...
2	66513	T_05088_19_normcore_W.jpg, T_07416_19_lounge_W...	T_08306_10_sportivecasual_W.jpg, T_08918_19_no...	W_14828_50_classic_W.jpg	T_06910_50_classic_W.jpg, W_10984_50_feminine_...
3	58251	T_08242_10_sportivecasual_W.jpg, T_08663_19_no...	T_11058_90_lingerie_W.jpg, T_12089_80_powersui...	T_14538_00_cityglam_W.jpg, W_00716_60_minimal_...	W_04101_19_lounge_W.jpg, W_08886_10_athleisure...
4	48094	T_14538_00_cityglam_W.jpg	W_09479_70_hippie_W.jpg	T_14538_00_cityglam_W.jpg	W_09479_70_hippie_W.jpg
...	...	...	...	...	...
463	64621	W_29596_10_sportivecasual_M.jpg	W_24250_90_hiphop_M.jpg	W_28925_90_hiphop_M.jpg	W_24250_90_hiphop_M.jpg
464	66842	W_27765_60_mods_M.jpg	W_24352_70_hippie_M.jpg	W_27765_60_mods_M.jpg	W_16485_00_metrosexual_M.jpg
465	64772	W_32595_60_mods_M.jpg, W_48687_60_mods_M.jpg	W_24470_70_hippie_M.jpg, W_24590_60_mods_M.jpg...	W_33329_50_ivy_M.jpg, W_48687_60_mods_M.jpg	W_24590_60_mods_M.jpg
466	64828	W_35091_80_powersuit_W.jpg	W_34027_10_sportivecasual_W.jpg	W_22783_70_hippie_W.jpg, W_35091_80_powersuit_...	W_34027_10_sportivecasual_W.jpg
467	67669	W_71920_60_mods_M.jpg	W_52548_50_ivy_M.jpg, W_52586_50_ivy_M.jpg	W_52578_50_ivy_M.jpg	W_52521_50_ivy_M.jpg

468 rows × 5 columns

이미지 전처리

ResNet-18 모델을 사용하여 이미지의 feature vector를 추출 및 저장

In [ ]:
```
import os
import torch
from torchvision import transforms
from PIL import Image
```

```python
# 이미지 전처리
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5498, 0.5226, 0.5052], std=[0.2600, 0.2582, 0.2620]),
])

# 이미지 feature vector 추출 함수
def extract_features(image_path, model, transform):
    if not os.path.exists(image_path):
        print(f"File not found: {image_path}")
        return None
    image = Image.open(image_path).convert('RGB')
    image = transform(image).unsqueeze(0)  # Add batch dimension
    with torch.no_grad():
        features = model(image)
    return features.numpy().flatten()

# ResNet-18 모델 초기화
class ResNet18FeatureExtractor(torch.nn.Module):
    def __init__(self):
        super(ResNet18FeatureExtractor, self).__init__()
        self.resnet18 = torch.hub.load('pytorch/vision:v0.10.0', 'resnet18', pretrained=False)
        self.resnet18.fc = torch.nn.Identity()  # 마지막 FC 레이어를 Identity로 대체

    def forward(self, x):
        return self.resnet18(x)

# 저장된 모델 전체 로드
model_path = r'../model/model_path.pth'
model = torch.load(model_path, map_location=torch.device('cpu'))
model.eval()
```

Out[ ]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=31, bias=True)
)
```

이미지 간 유사도를 계산 및 Validation 데이터 내 응답자의 스타일 선호 여부를 예측

In [61]:
```
import os
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
```

```
# 학습 이미지와 검증 이미지의 디렉토리 경로 설정
train_image_directory = '../dataset/training_image'
valid_image_directory = '../dataset/validation_image'

# 유사도 계산 함수
def calculate_similarity(feature1, feature2):
    return cosine_similarity([feature1], [feature2])[0][0]

# 응답자별 학습 이미지와 검증 이미지의 feature vector 추출 및 저장
def extract_respondent_features(mission2_result, model, transform):
    respondent_train_features = {}
    respondent_valid_features = {}

    for index, row in mission2_result.iterrows():
        respondent_id = row['응답자 ID']
        train_images = []
        valid_images = []

        if not pd.isna(row['train 선호']):
            train_images.extend([(img, '선호') for img in row['train 선호'].split(', ')])
        if not pd.isna(row['train 비선호']):
            train_images.extend([(img, '비선호') for img in row['train 비선호'].split(', ')])
        if not pd.isna(row['valid 선호']):
            valid_images.extend(row['valid 선호'].split(', '))
        if not pd.isna(row['valid 비선호']):
            valid_images.extend(row['valid 비선호'].split(', '))

        respondent_train_features[respondent_id] = {}
        respondent_valid_features[respondent_id] = {}

        for train_image, preference in train_images:
            image_path = os.path.join(train_image_directory, train_image).replace('\\', '/')
            if not os.path.exists(image_path):
                print(f"File not found: {image_path}")
                continue
            image_id = train_image.split('_')[1]
            feature = extract_features(image_path, model, transform)
            if feature is not None:
                respondent_train_features[respondent_id][image_id] = (feature, preference)

        for valid_image in valid_images:
            image_path = os.path.join(valid_image_directory, valid_image).replace('\\', '/')
            if not os.path.exists(image_path):
                print(f"File not found: {image_path}")
                continue
            image_id = valid_image.split('_')[1]
            feature = extract_features(image_path, model, transform)
            if feature is not None:
                respondent_valid_features[respondent_id][image_id] = feature

    return respondent_train_features, respondent_valid_features

# 응답자별 학습 이미지와 검증 이미지의 feature vector 추출
respondent_train_features, respondent_valid_features = extract_respondent_features(filtered_mission2_result, model, transform)

# Validation 데이터 내 응답자의 스타일 선호 여부 예측
def predict_preference(respondent_train_features, respondent_valid_features, threshold=0.5):
    predictions = {}
    for respondent_id, valid_features in respondent_valid_features.items():
        respondent_predictions = {}
        for valid_image_id, valid_feature in valid_features.items():
            similarities = {'선호': [], '비선호': []}
            for train_image_id, (train_feature, preference) in respondent_train_features[respondent_id].items():
                similarity = calculate_similarity(valid_feature, train_feature)
                similarities[preference].append(similarity)
            # 모든 유사도를 사용하여 평균 계산
            if len(similarities['선호']) == 0 and len(similarities['비선호']) == 0:
                preference_score = 0  # 기본값 설정
            else:
                preference_score = (sum(similarities['선호']) - sum(similarities['비선호'])) / (len(similarities['선호']) + len(similarities[
            respondent_predictions[valid_image_id] = '선호' if preference_score > threshold else '비선호'
        predictions[respondent_id] = respondent_predictions
    return predictions
```

In [62]:
```
predictions = predict_preference(respondent_train_features, respondent_valid_features, threshold=0.5)
```

성능 확인

In [67]:
```
# 성능 측정 (예시로 정확도 계산)
def calculate_accuracy(predictions, mission2_result):
    correct = 0
    total = 0
    for index, row in mission2_result.iterrows():
        respondent_id = row['응답자 ID']
        if respondent_id not in predictions:
            continue
        for valid_image in row['valid 선호'].split(', ') if not pd.isna(row['valid 선호']) else []:
            valid_image_id = valid_image.split('_')[1]
            if valid_image_id in predictions[respondent_id]:
```

```
                total += 1
                if predictions[respondent_id][valid_image_id] == '선호':
                    correct += 1
        for valid_image in row['valid 비선호'].split(', ') if not pd.isna(row['valid 비선호']) else []:
            valid_image_id = valid_image.split('_')[1]
            if valid_image_id in predictions[respondent_id]:
                total += 1
                if predictions[respondent_id][valid_image_id] == '비선호':
                    correct += 1
    return correct / total if total > 0 else 0

# 정확도 계산
accuracy = calculate_accuracy(predictions, mission2_result)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.55
```

.