Fashion-How 발표자료

Sub-Task 1

박상하

CONTENTS

1. 문제 정의

문제 정의

과제 목표

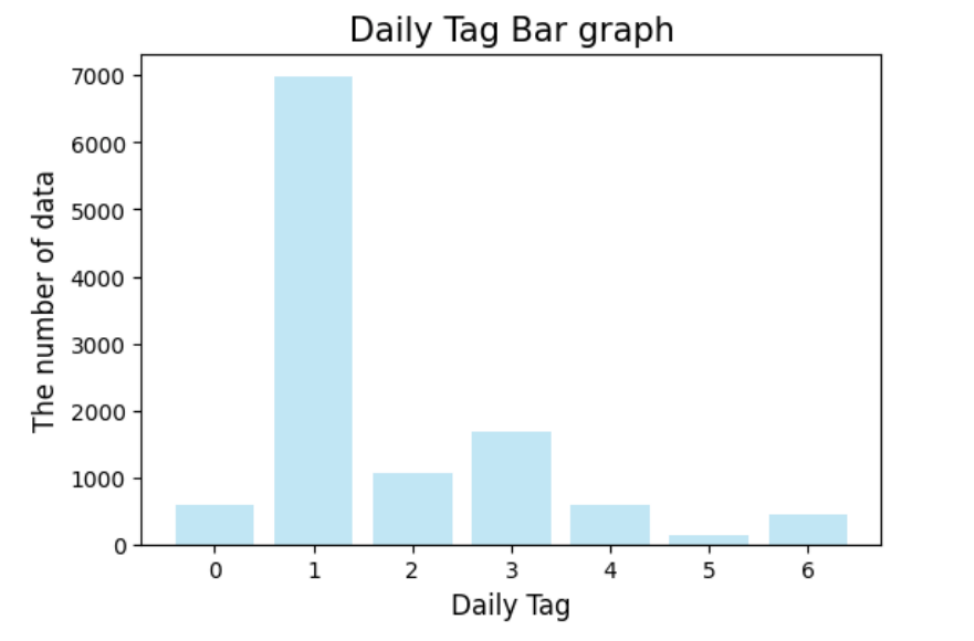
1. 문제 정의 및 과제 목표

1. 문제 정의
    1. 14000의 패션 이미지를 위치 정보 bounding box를 참고해서 각각의 아이템을 분류한다.
    2. 각 아이템을 3가지의 감성 특징(일상성, 성, 장식성)을 분류한다.
2. 과제 목표
    1. **이미지가 입력되면 이 이미지를 일상성, 성, 장식성 총 3개의 감성 특징을 각 특징별로 분류할 수 있는 모델을 만든다.**

2. 데이터 분석

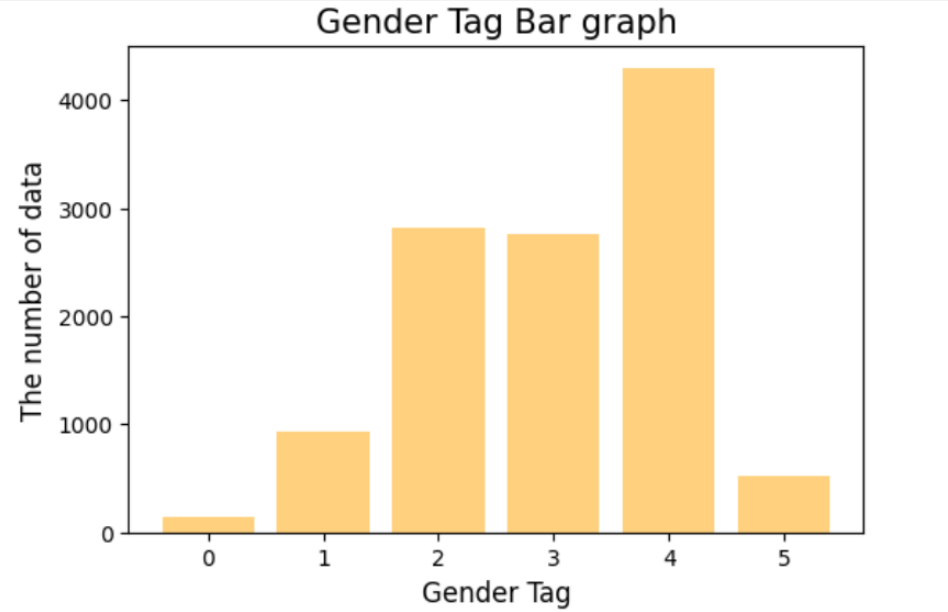
1. 데이터 분석

1. 일상성에 대한 분류



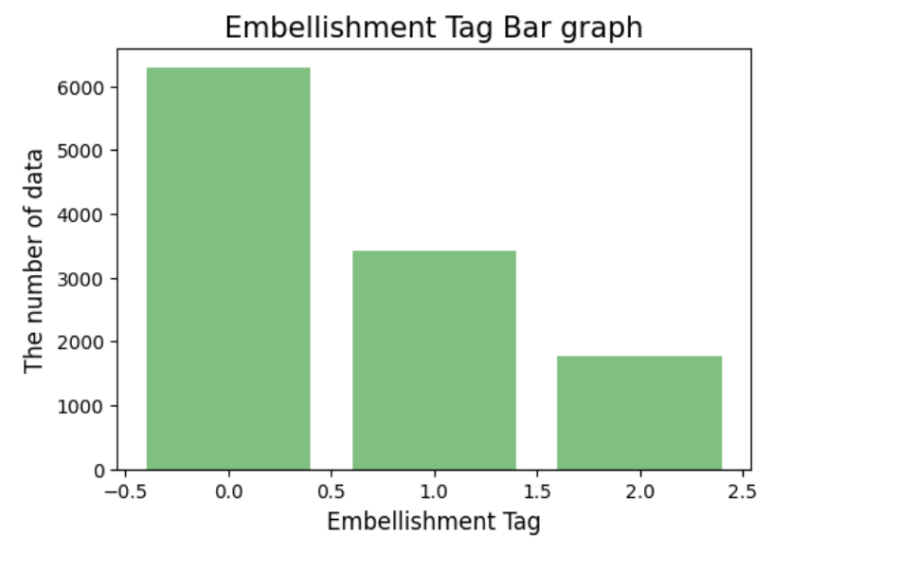
1. 일상성 Tag는 총 7개의 클래스가 존재한다.
2. 각 Tag간의 데이터 불균형이 심하다
    - 1번 클래스의 데이터 수는 대략 7000개이지만 5번 클래스의 데이터 수는 1000개가 되지 않는다.

2. 성에 대한 분류



1. 성 Tag는 총 6개의 클래스가 존재한다.
2. 일상성과 마찬가지로 각 Tag간의 데이터 불균형이 심하다

3. 장식성에 대한 분류



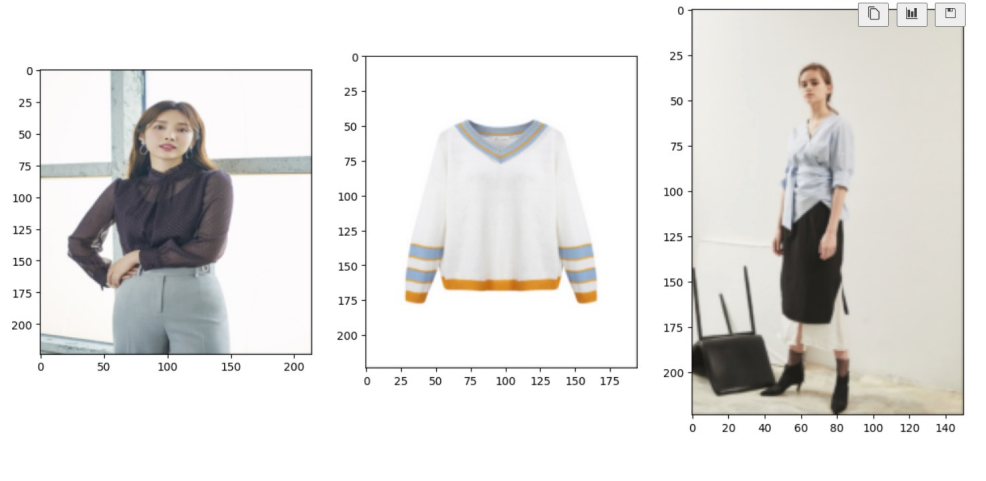
1. 장식성에 대한 Tag는 총 3가지가 존재
2. 데이터 불균형이 존재하지만 일상성, 성 분류에 비하면 적은 편

- **모든 클래스에 대해서 데이터 불균형이 존재하므로 이 문제에 대해서 적절한 접근이 필요하다.**

3. 데이터 전처리

1. 데이터 로딩
2. 데이터 전처리
3. 데이터 증강

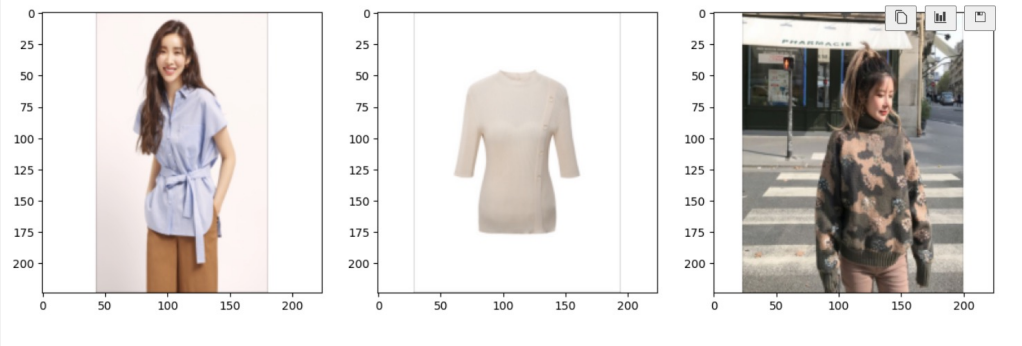
1. 데이터 로딩



```
img_name = row["image_name"]
img = io.imread(os.path.join(self._dir_path,img_name))
if img.shape[2] != 3:
    img = color.rgba2rgb(img)

x_min = int(row["BBox_xmin"])
y_min = int(row["BBox_ymin"])
x_max = int(row["BBox_xmax"])
y_max = int(row["BBox_ymax"])

height = y_max - y_min
width = x_max - x_min

img_ = copy.deepcopy(img)
img_ = img_[y_min:y_max,x_min:x_max]

if height != width :
    if height > width :
        new_h = self._img_size
        new_w = int(width * self._img_size / height)
    else :
        new_w = self._img_size
        new_h = int(height * self._img_size / width)
else :
    new_h = self._img_size
    new_w = self._img_size

img_ = transform.resize(img_, (new_h, new_w), mode='constant')
```

1. 원본의 이미지에서 Bounding Box 부분을 잘라서 불러온다.
2. 불러온 이미지의 높이 및 너비의 비율에 맞추어서 이미지 크기를 재조정한다.
    1. 높이, 너비 중 긴 것의 길이를 224로 맞춘다.
    2. 높이, 너비 중 짧은 것의 길이는 기존의 높이, 너비의 비율에 맞게 줄인 길이를 계산해서 맞춘다.

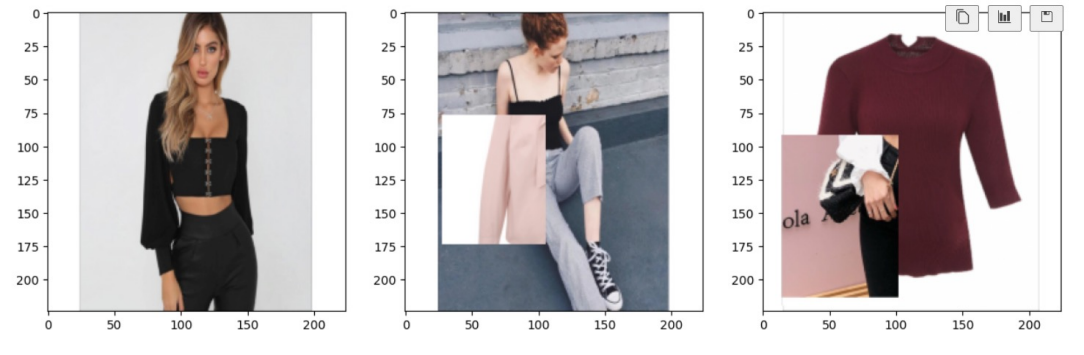
2. 데이터 전처리



```
for data in tqdm(dataset) :
    img = data["image"]
    height, width, _ = img.shape

    new_img = np.ones((self._img_size, self._img_size, 3))
    if height != width :
        if width == self._img_size :
            height_start = int((self._img_size - height) / 2)
            new_img[height_start:height_start+height, :, :] = img
        else :
            width_start = int((self._img_size - width) / 2)
            new_img[:, width_start:width_start+width, :] = img
    else :
        new_img = img

    data["image"] = new_img

return dataset
```

1. 높이 및 너비의 길이를 균일하게 맞춘다.
    - 모델 학습에는 224로 통일
2. 원래 이미지를 중앙에 두고 나머지 부분을 값을 1로 Padding을 진행

3. 데이터 증강



```
lam = np.random.uniform(0, 1)
val = np.math.sqrt(1 - lam)
r_y = np.random.randint(0, org_h)
r_h = val * org_h

r_x = np.random.randint(0, org_w)
r_w = val * org_w

# width
x1 = round(r_x - (r_w / 2))
x1 = x1 if x1 > 0 else 0
x2 = round(r_x + (r_w / 2))
x2 = x2 if x2 < org_w else org_w

# height
y1 = round(r_y - (r_h / 2))
y1 = y1 if y1 > 0 else 0
y2 = round(r_y + (r_y / 2))
y2 = y2 if y2 < org_h else org_h

new_img = copy.deepcopy(org_img)
new_img[y1:y2, x1:x2, :] = tar_img[y1:y2, x1:x2, :]
```

1. CutMix 방법을 통해서 데이터 증강을 진행.
    - 기존 이미지 → 원본 이미지 + 4개의 증강된 이미지
    - 학습된 이미지 수 : 14000 → 70000개
2. 기존 이미지에 선정된 **이미지의 부분을 교체**
    - **이미지의 부분 비율을 임의로 선정**
    - **부분이 되는 y, x 지점을 임의로 선정**
    - **기존 정답에서 다른 이미지의 부분 만큼 정답이 변경**

4. 모델 설계

1. Backbone 선정
2. 모델 구조 선정
3. Classification Head 선정

1. Backbone 선정

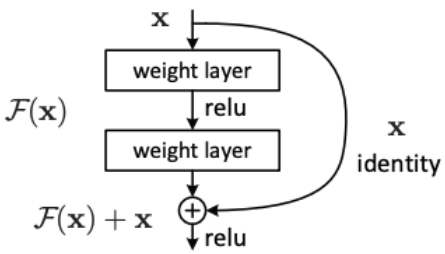
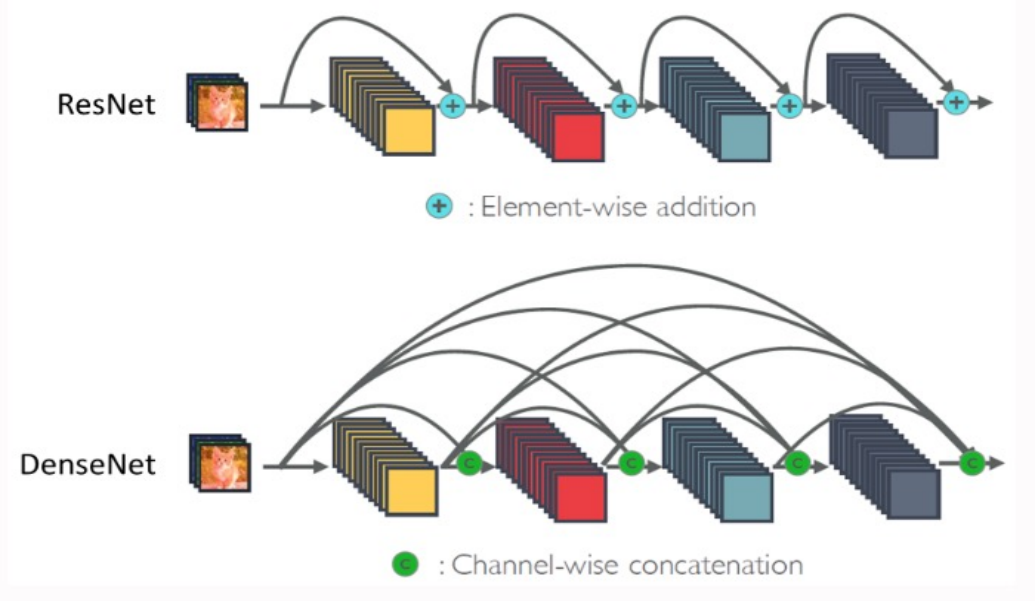
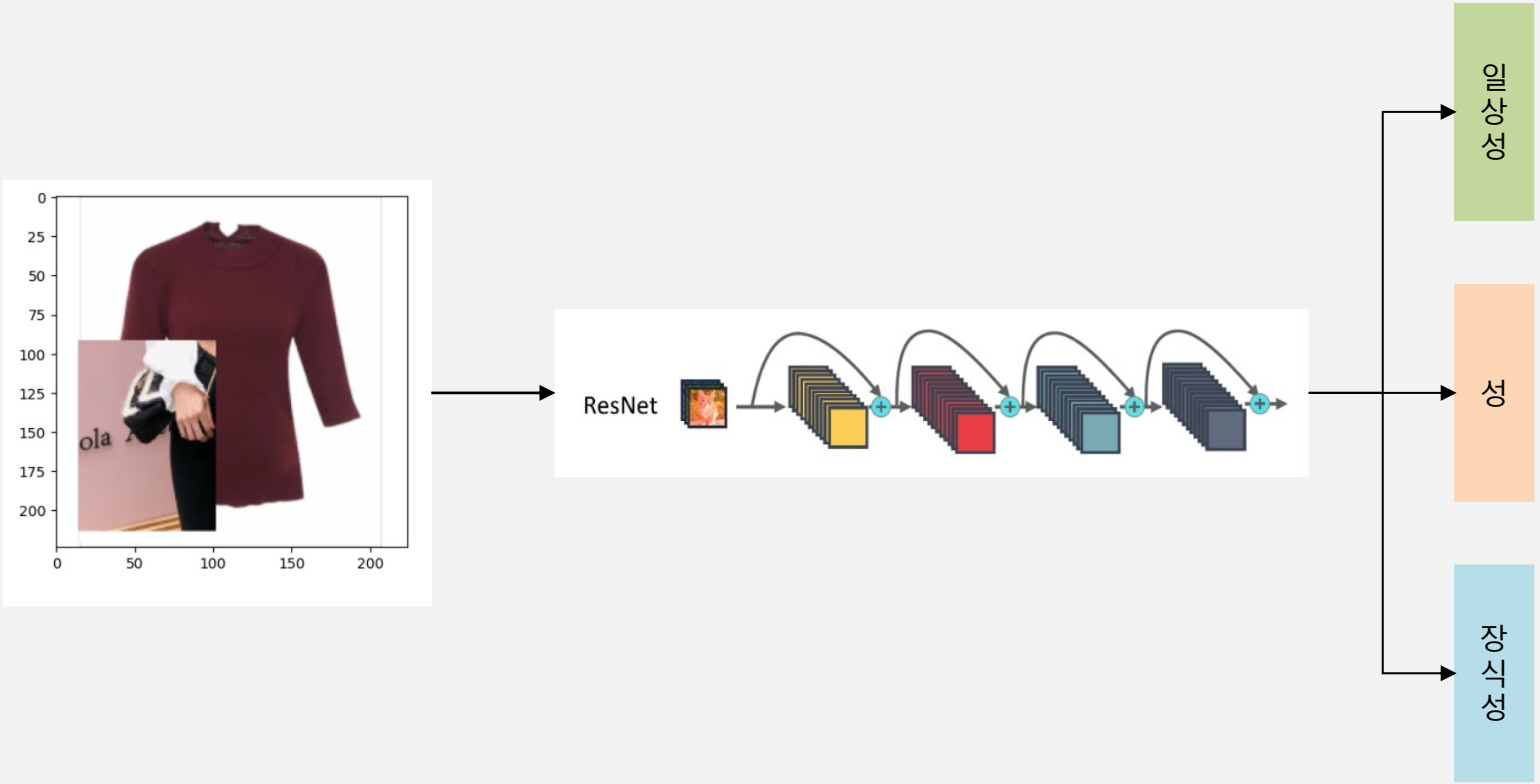


Figure 2. Residual learning: a building block.



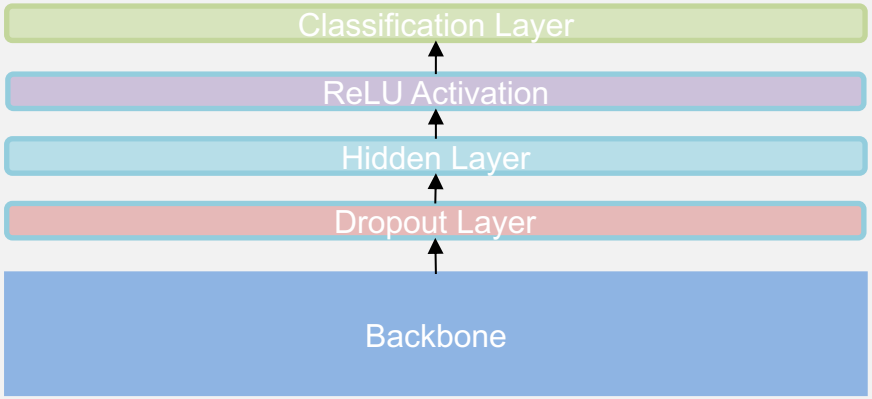
1. ResNet 및 DenseNet 모델을 선정
    - 모델의 깊이가 특정 깊이 이상 깊어지면 gradient vanishing 문제로 인해서 모델의 성능이 감소하게 되는 현상이 발생
    - 이 문제점을 방지하기 위해서 Residual Learning 이라는 방법을 모델 구조에서 사용
2. ResNet 과 DenseNet 과의 차이점
    - ResNet은 이전의 결과와 현재의 결과를 Sum하는 구조
    - DenseNet은 이전의 모든 결과들과 현재의 결과를 Concatenate하는 구조
3. **최종 Backbone 선정**
    - **ResNet 152 & DenseNet 161**

2. 모델 구조 선정



1. (224, 224, 3) 이미지가 모델에 입력된다.
2. ResNet 혹은 DenseNet을 통해서 이미지로부터 Feature를 추출한다.
3. 해당 Feature를 일상성, 성, 장식성 Classification Head으로 보내서 분류를 진행한다.
    1. **일상성, 성, 장식성 총 3가지에 대해서 Softmax Loss를 각각 구한다.**
    2. 3개의 Loss의 평균을 가지고서 모델의 학습을 진행한다.

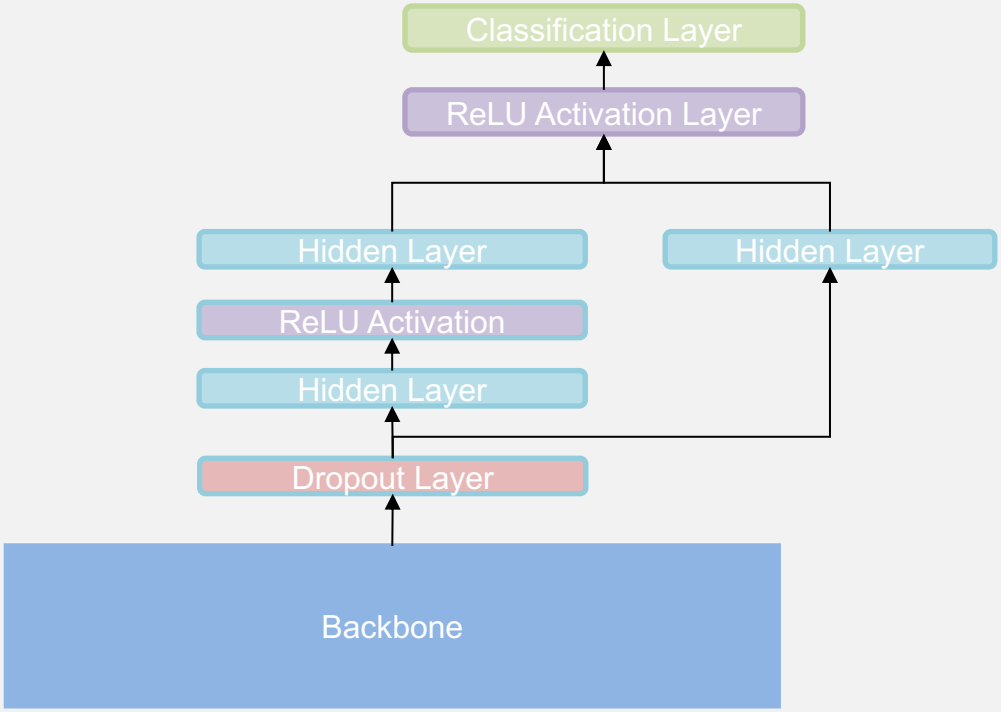
3. Classification Head 선정




```python
class DenseNetFeedForwardModel(nn.Module):
    """
    DenseNet161 161를 backbone으로 해서 이미지가 입력이되면 daily, gender, embellishment를 분류하는 모델입니다.
    """
    def __init__(self,
        hidden_size,
        class1_size,
        class2_size,
        class3_size,
        dropout_prob,
        pretrained=True
    ):
        super(DenseNetFeedForwardModel, self).__init__()

        if pretrained == True :
            self._backbone = densenet161(pretrained=True, progress=True)
        else :
            self._backbone = densenet161(pretrained=False, progress=False)

        self._feature_size = 1000
        self._hidden_size = hidden_size
        self._dropout_prob = dropout_prob
        self._class1_size = class1_size
        self._class2_size = class2_size
        self._class3_size = class3_size

        self._drop = nn.Dropout(self._dropout_prob)
        self._classifier1 = nn.Sequential(
            nn.Linear(self._feature_size, self._hidden_size),
            nn.ReLU(),
            nn.Linear(self._hidden_size, self._class1_size)
        )

        self._classifier2 = nn.Sequential(
            nn.Linear(self._feature_size, self._hidden_size),
            nn.ReLU(),
            nn.Linear(self._hidden_size, self._class2_size)
        )

        self._classifier3 = nn.Sequential(
            nn.Linear(self._feature_size, self._hidden_size),
            nn.ReLU(),
            nn.Linear(self._hidden_size, self._class3_size)
        )
```

1. ResNet 혹은 DenseNet을 통해서 이미지의 Feature를 추출
2. 해당 Feature를 Dropout Layer로 먼저 전달한다.
3. Feature를 **Hidden Layer, ReLU Activation Layer, Classification Layer**로 차례대로 전달하면서 최종 Output을 생성한다.

3. Classification Head 선정



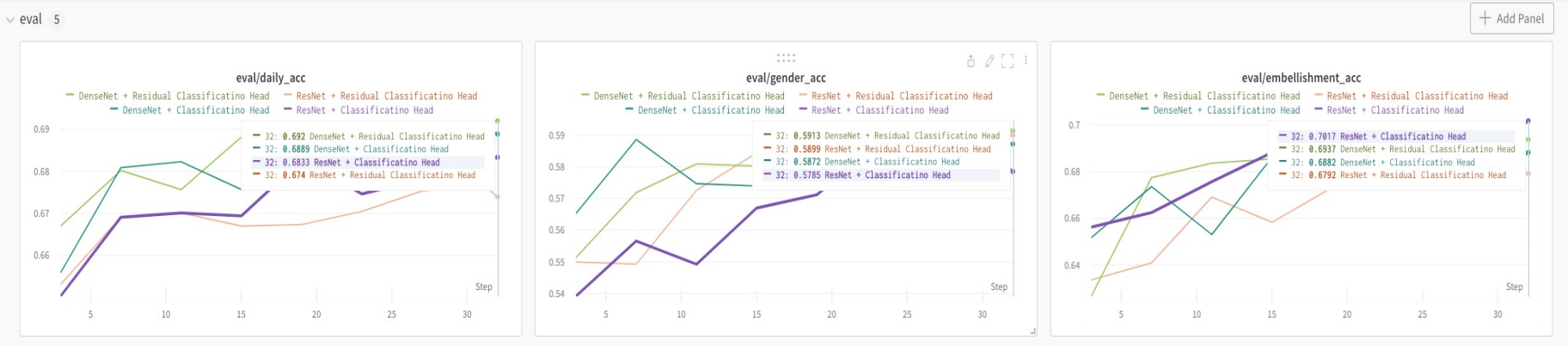

```python
class ResidualClassifier(nn.Module) :
    """
    ResidualConnection을 응용해서 만든 클래스입니다.
    long network와 short network가 있는데 이를 통과하면
    representation size가 동일하게 되는데 이를 더한 이후에 classifier로 전달이 됩니다.
    """
    def __init__(self, feature_size, hidden_size, class_size) :
        super(ResidualClassifier, self).__init__()

        self._feature_size = feature_size
        self._hidden_size = hidden_size
        self._class_size = class_size

        self._long = nn.Sequential(
            nn.Linear(self._feature_size, self._hidden_size),
            nn.ReLU(),
            nn.Linear(self._hidden_size, self._hidden_size)
        )

        self._short = nn.Linear(self._feature_size, self._hidden_size)
        self._act = nn.ReLU()
        self._classifier = nn.Linear(self._hidden_size, self._class_size)

    def forward(self, x) :
        h1 = self._long(x)
        h2 = self._short(x)

        h = self._act(h1 + h2)
        o = self._classifier(h)
        return o
```

1. Residual Learning을 참고해서 Classification Head를 더 깊게 쌓아서 설계
2. **깊이가 긴 Layer 1개, 깊이가 짧은 Layer 1개를 생성하고 각각의 출력 Feature을 더한다.**
3. 더한 Feature을 ReLU Activation Layer을 전달한 이후에 Classification Layer로 전달해서 최종 Output을 구한다.

5. 학습 및 성능 평가

1. 학습 과정
2. 모델 성능 평가

1. 학습 과정 및 모델 성능 평가

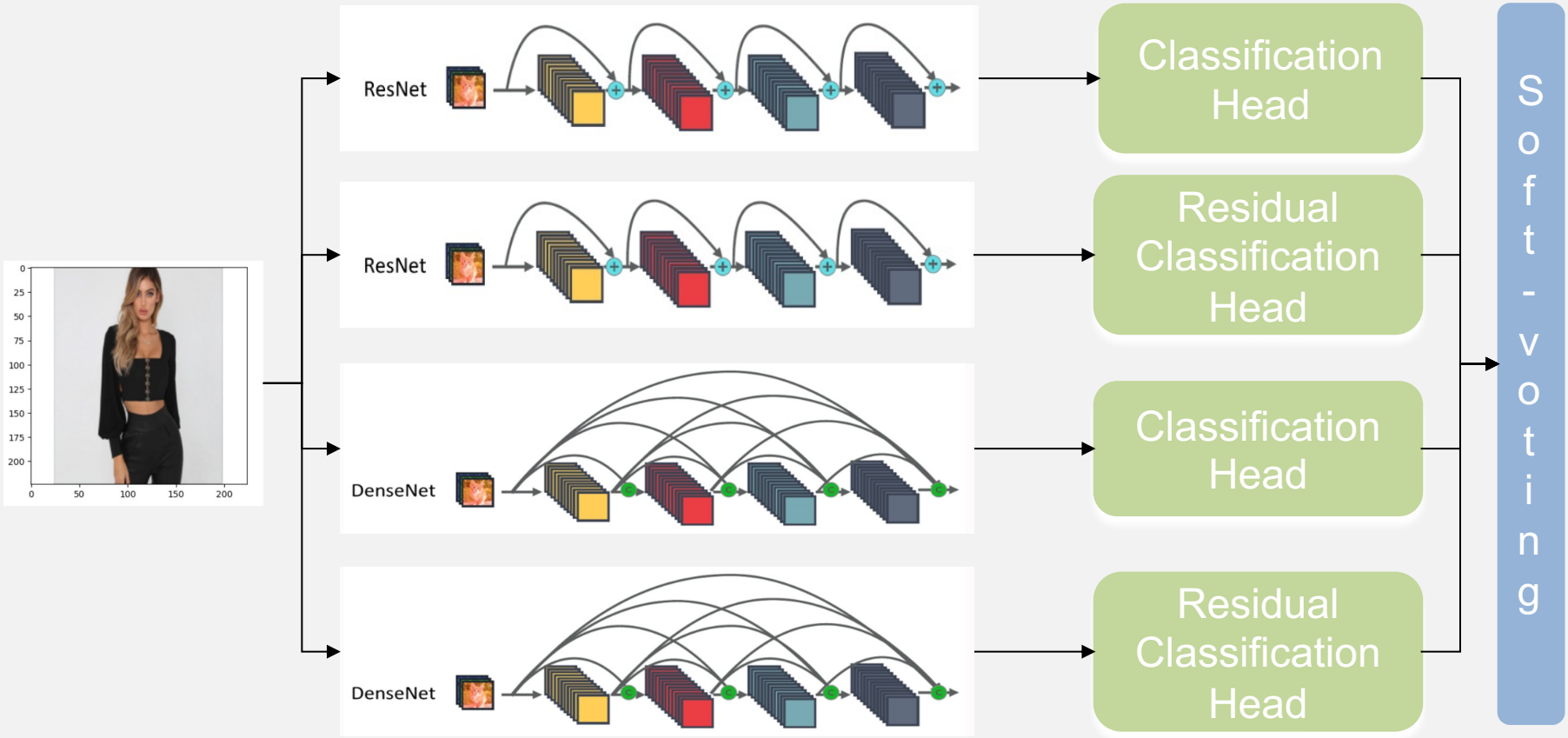


1. 훈련 데이터를 75 : 25의 비율로 나누어서 모델의 성능 평가를 진행
2. 모델의 구조를 다르게 하고 Hyperparameter는 동일한 조건에서 모델을 학습 및 평가
3. Hyperparameter
    - Optimizer : AdamW
    - Scheduler : Linear Warmup Scheduler
        - Warmup Ratio : 0.05
    - Epoch : 3
    - CutMix Augmentation Size : 4
    - Batch Size : 64
    - Weight Decay : 1e-4

6. 최종 모델 선정

1. 앙상블

1. 앙상블



- 4개의 모델을 평가 때와 동일한 Hyperparameter에서 전체 데이터 대상으로 학습
- **학습된 모델의 출력 값들을 Soft-Voting**을 하는 방법으로 최종 추론 결과를 제출