창의개인연구 결과 보고서

2020학년도 2학기 창의 개인 연구 결과 보고서

K-Means Clustering을 통한 Labeled Image로부터 Class 추출 및 Image Segmentation 결과 분석

경기북과학고등학교
1학년 5반 2번 성명: 강태구
지도교사 : 임석영 (확인)

초록(Abstract)

Image Segmentation은 computer vision에서 매우 중요하고 활발히 연구가 진행되고 있는 분야이다. 본 연구에서는 K-Means Clustering을 이용해 Image Segmentation 데이터셋에서 각 클래스를 대표하는 RGB 값을 찾아내는 것, K-Means Clustering을 거친 layer로부터 사용자가 정의한 RGB값으로 새롭게 labeled data를 생성하는 것을 진행하였다. 이에 Cityscapes Image Pairs Dataset을 이용해 13개의 클러스터로 K-Means Clustering을 진행해 클래스를 구분하였고, 사용자가 정의한 RGB값으로 새롭게 labeled data를 생성할 수 있는 툴을 개발하였다. 또한 Semantic Segmentation 모델을 거친 후 생성 된 마스킹 된 이미지를 통해 원하는 클래스 부분에 해당하는 마스크를 K-Means 클러스터링을 이용해 높은 정확도로 생성할 수 있었다. 본 연구를 통해 Image Segmentation 데이터 전처리, 학습결과 분석 전반에 비지도 학습 방법인 K-Means Clustering이 매우 유용함을 확인하였다.

1. 연구동기 및 목적

1-1. 연구 동기

본 연구자는 항공 사진 semantic segmentation을 진행하여 착륙지점을 알려주는 프로젝트를 진행하면서 데이터셋의 라벨 디테일이 제공되지 않는 경우가 많아 일일이 RGB 값들을 찾아내어 다시 클래스를 구분해 라벨링하는 작업을 거쳤었다. 이 과정은 번거롭고 비효율적이며 색상이 유사한 것은 구분하기 힘들다. 이렇게 RGB 값을 찾아서 HSV이미지로 변환하여 마스크를 생성할 때 색상 범위를 제대로 지정하지 못하면 마스크를 제대로 생성할 수 없게 된다. 따라서 라벨 디테일이 제공되지 않는 데이터셋에서 라벨 디테일을 찾아내는 과정은 필요하며 라벨 디테일을 사용자 편의로 설정할 수 있으면 개발 입장에서 편리하겠다고 생각이 들었다. 또한 Image Segmentation 결과에서의 원하는 클래스 부분에 해당하는 마스크를 찾아내는 과정을 효과적으로 찾을 수 있는 부분이 필요하다고 생각하였다.

1-2. 연구 목적

(1) 라벨 디테일 (클래스와 RGB값)이 제공되지 않는 Image Segmentation용 데이터셋으로부터 K-Means Clustering을 이용해 각 클래스를 대표하는 RGB 값을 찾아낸다.

(2) 기존 데이터셋에서 제공하는 labeled data가 아닌 K-Means Clustering을 거친 layer로부터 사용자가 정의한 RGB값으로 새롭게 labeled data를 생성한다.

(3) Image Segmentation 결과 생긴 masked image에서 원하는 클래스 부분에 해당하는 마스크를 찾아낸다.

2. 연구 내용 및 방법

2.1. K-Means Clustering

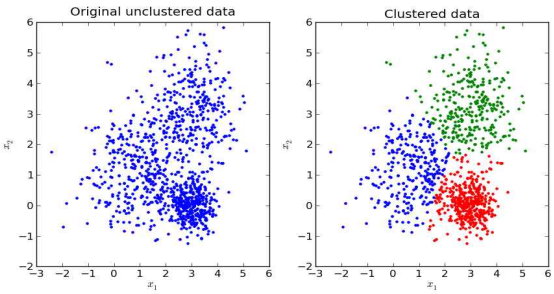


그림 1 k=3 인 경우의 K-Means Clustering의 결과

주어진 데이터를 k개의 클러스터로 묶는 알고리즘으로 각 클러스터와 거리 차이의 분산을 최소화하는 방식으로 동작하는 비지도학습의 방법이다.

$$\min_{\{r_{nk}\},\{\boldsymbol{\mu}_k\}} J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

그림 2 K-means의 비용함수 J를 최소화하는 r_{nk}, μ_k 를 찾음.

N은 데이터의 개수, K는 클러스터의 개수, μ_k는 k번째 클러스터의 중심, r_{nk}는 n번째 데이터가 k번째 클러스터에 속하면 1, 속하지 않으면 0을 가지는 이진 변수이다.
K-means algorithm을 구하는 방법은 Expectation과 Maximization 로 나누어진다. 먼저 μ_k에 랜덤 값을 사용하여 임의의 초깃값을 설정한다.
E(Expectation) : μ_k를 고정하면서 J를 최소화하는 r_{nk}값을 지정하는데 r_{nk}는 모든 데이터 n에 대해 모든 데이터 n에 대해 모든 클러스터 중에서 $\|x_n - \mu_k\|$값이 가장 작은 클러스터에 할당한다.
M(Maximization) : 새롭게 얻은 r_{nk}를 고정하고 μ_k는 k 번째 클러스터의 mean을 계산한다.
위 두 과정을 반복하여 J가 적당한 범위 내로 수렴할 때까지 E,M 과정을 반복한다.
본 연구에서는 여기에서 아이디어를 얻어 labeled image에서 가장 중요한 색상들을 찾고 유사한 색상을 식별하기 위해 KMeans Clustering을 이용하고자 하였다.

2.2. 연구 Dataset 선정 및 original photo, labeled image 분류

그림 3 Cityscapes Image Pairs Dataset

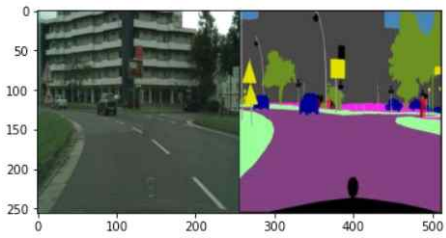


그림 4 256x512로 이루어짐

Kaggle에서 Cityscapes Dataset의 경량화 버전인 Cityscapes Image Pairs 데이터셋을 Semantic Segmentation을 위한 데이터셋으로 선정함. 2975개의 training 이미지 파일이 있고 500개의 validation 이미지 파일이 있고, 각 파일은 256x512 pixel로 이루어어져 있고 왼쪽 절반은 original photo이고 오른쪽 절반은 labeled image임.

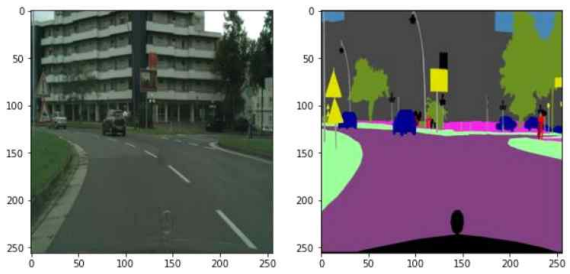


그림 5 256x256의 2개의 이미지로 나눔.

```
def divide_img(img):
    img = np.array(img)
    image = img[:,:256]
    mask = img[:,256:]
    return image, mask

real_img = []
mask_img = []

for i in range(len(train_fns)):
    sample_image = os.path.join(train_dir,train_fns[i])
    sample_image = Image.open(sample_image).convert("RGB")
    cityscape, label= divide_img(sample_image)
    real_img.insert(i,cityscape)
    mask_img.insert(i,label)
```
그림 6 이미지를 나누는 코드

2.3. K-Means Clustering을 이용해 각 클래스를 대표하는 RGB 값을 찾기.

```
from sklearn.cluster import KMeans
```
sklearn 라이브러리를 이용하여 K-Means Clustering 구현할 수 있음
ㄱ. train 이미지에서 labeled image 중 랜덤하게 10개를 선택하여 행,열 관계 없이 모든

rgb 값을 받기 위해서 (-1,3)의 shape 로 reshape하여 colors에 저장한다.
ㄴ. colors를 (-1,3)으로 reshape하여 rgb값 만을 뽑아낸다.

```
for i in range(len(train_fns)):
    sample_image = os.path.join(train_dir,train_fns[i])
    sample_image = Image.open(sample_image).convert("RGB")
    cityscape, label= divide_img(sample_image)
    real_img.insert(i,cityscape)
    mask_img.insert(i,label)
```

```
colors = []
for i in np.random.choice(len(train_fns),10):
    colors.append(mask_img[i].reshape(-1,3))
colors = np.array(colors)
print(colors.shape)
colors = colors.reshape(-1,3)
print(colors.shape)
```

```
(10, 65536, 3)
(655360, 3)
```

그림 8 ㄱ,ㄴ 구현 부분

ㄷ. 클래스의 수를 13으로 생각해 클러스터의 개수를 13으로 하여 K-Means Clustering을 실행한다.
ㄹ. 클러스터들의 중심을 출력하여 실제 labeled image들이 무슨 색으로 색칠되었는지 알 수 있다.

```
km = KMeans(13)
km.fit(colors)
```

```
KMeans(n_clusters=13)
```

```
print(km.cluster_centers_)
```

```
[[ 83.01625141   3.38398845  74.07313133]
 [106.88990483 140.18192449  36.77622691]
 [128.15951638  62.91103777 126.87000313]
 [  4.76332021   3.23744148   7.05079141]
 [235.97293031  40.8163523  224.42478889]
 [ 70.48906344  70.12116273  69.42762155]
 [ 73.41495235 129.50261506 175.70966992]
 [211.79429776  22.02779545  57.76029903]
 [  3.78224193   1.9471692  135.81363219]
 [134.84306015 115.37644868 104.85871965]
 [216.16610738 196.42980984  42.84535794]
 [160.60984747 243.70430827 156.77950227]
 [201.32821851 153.49884916 168.77198097]]
```

그림 9 ㄷ,ㄹ 구현

```
s = mask_img[0].reshape(-1,3)
s = km.predict(s)
s = s.reshape(256,256)
print(s)
```

```
[[12  1 12 ... 12  1 12]
 [ 1  2  2 ...  9  9  1]
 [ 1  2  3 ...  6  6  1]
 ...
 [ 1  0  0 ...  0  0  7]
 [ 1  9  9 ...  9  9  1]
 [12  1 12 ... 12  1 12]]
```

그림 10. 256x256 labeled 이미지에서 각 픽셀에 해당하는 영역의 클래스

2.4 기존 데이터셋에서 제공하는 labeled image가 아닌 clustering을 거친 layer를 사용자가 알고 있는 RGB값으로 변환

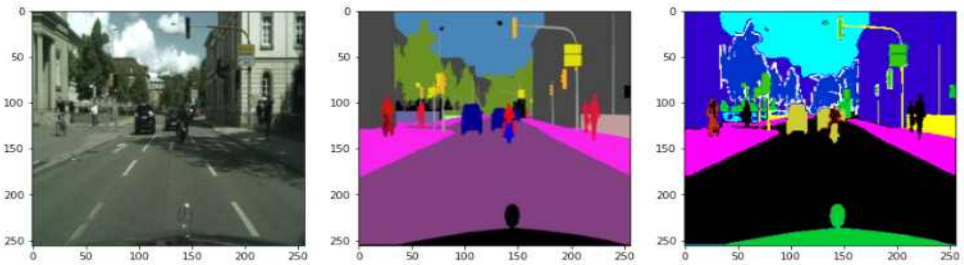


그림 11 왼쪽부터 순서대로 원본, 라벨링된 데이터, 클러스터링을 거친 데이터

ㄱ. FindClassLoc() 는 색상 클러스터링의 출력을 13차원 레이어 표현으로 변환한다. 이것은 후에 머신러닝 과정에서 유용하게 쓰일 수 있다.

ㄴ. 클러스터링 후 이미지에는 13개의 서로 다른 레이어에 대한 정보가 포함되는데 RGB 형태만 시각화할 수 있으므로 MaskOwnColor()는 13개의 레이어 표현을 RGB 형태로 변환해준다.

```
def FindClassLoc(mask):
    s = mask.reshape(-1,3)
    s = km.predict(s)
    s = s.reshape(256,256)
    n = len(km.cluster_centers_)
    cls = np.zeros((256,256,n))
    for i in range(n):
        m = np.copy(s)
        m[m!=i] = 0
        m[m!=0] = 1
        cls[:,:,i] = m
    return cls
```

그림 12 클래스별 픽셀 존재 유무를 포함하는 (256,256,13)의 13차원 layer를 반환

```
def MaskOwnColor(cls):
    colors = [(255,0,0), (0,255,0), (0,0,255),
              (255,255,0), (255,0,255), (0,255,255),
              (255,255,255), (200,50,0),(50,200,0),
              (50,0,200), (200,200,50), (0,50,200),
              (0,200,50), (0,0,0)]
    nimg = np.zeros((256,256,3))
    for i in range(cls.shape[2]):
        c = cls[:,:,i]
        col = colors[i]

        for j in range(3):
            nimg[:,:,j]+=col[j]*c
    nimg = nimg/255.0
    return nimg
```

그림 13 새로운 colors를 통해 nimg의 새롭게 라벨링된 (256,256,3)의 RGB 이미지를 반환한다.

2.5 Image Segmentation 결과에서의 원하는 클래스 부분에 해당하는 마스크를 찾기

```
def find_class(class_num,mask):
    s = mask.reshape(-1,3)
    s = km.predict(s)
    s = s.reshape(256,256)
    m = np.copy(s)
    m[m!=class_num] = 0
    m[m!=0] = 1
    m = (m*255)
    return m
```

그림 14 class에 해당하는 mask를 구하는 함수 find_class()

```
fig,axes = plt.subplots(1,4,figsize=(5*4,5))
i = 232
axes[0].imshow(real_img[i])
axes[1].imshow(mask_img[i])
axes[2].imshow(find_class(2,mask_img[i]))
axes[3].imshow(find_class(4,mask_img[i]))
```

그림 15 도로와 인도에 해당하는 mask 이미지를 나타낸다.

masked image와 찾고자 하는 class를 입력받아 s를 이루는 픽셀이 K-Means 결과 생성된 클러스터의 index를 가지는 s를 얻어 m으로 복제한다. m에서 입력받은 클러스터와 같은 값을 갖는 픽셀은 255로 아닌 값은 0으로 바꾸어 (255,255)의 이미지를 반환하였다. 밑에 예시로 둔 2,4번 클래스는 도로, 인도를 나타낸다.

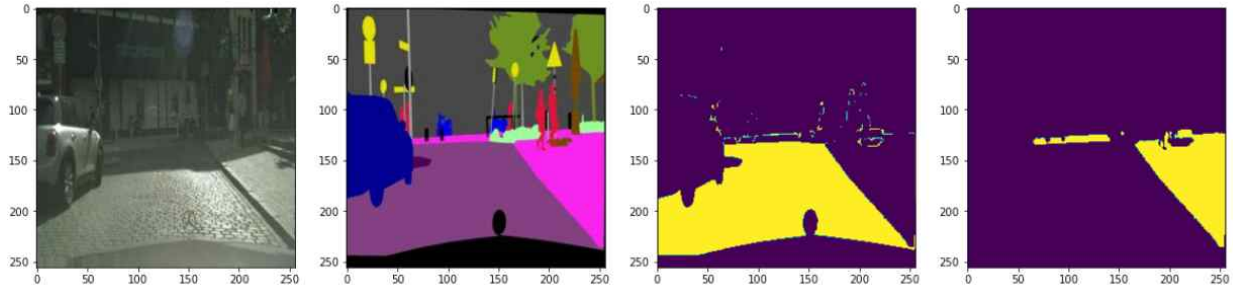
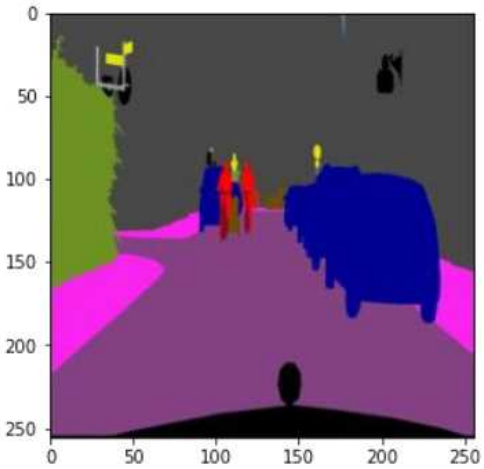


그림 16 Masked Image로부터 도로와 인도 영역을 K-Means 클러스터링을 이용해 얻은 사진.

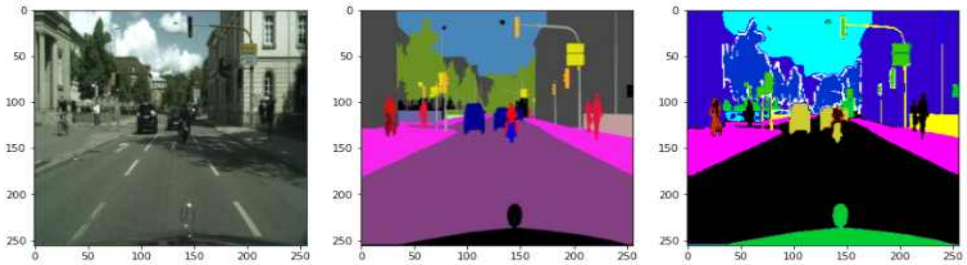
3. 연구 결과

3.1. K-Means Clustering을 이용해 각 클래스를 대표하는 RGB 값을 찾아내기

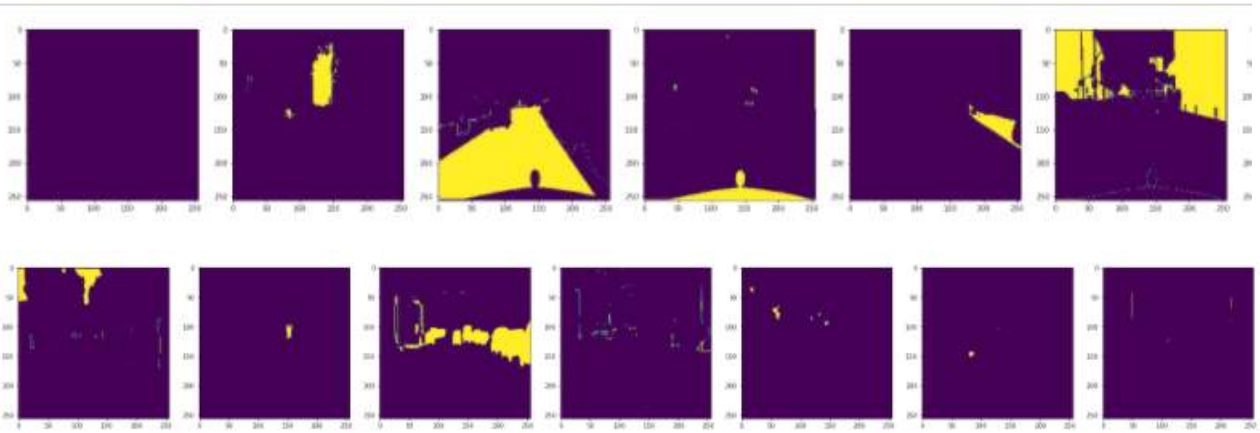


```
[[ 83.01625141   3.38398845  74.07313133]
 [106.88990483 140.18192449  36.77622691]
 [128.15951638  62.91103777 126.87000313]
 [  4.76332021   3.23744148   7.05079141]
 [235.97293031  40.8163523  224.42478889]
 [ 70.48906344  70.12116273  69.42762155]
 [ 73.41495235 129.50261506 175.70966992]
 [211.79429776  22.02779545  57.76029903]
 [  3.78224193   1.9471692  135.81363219]
 [134.84306015 115.37644868 104.85871965]
 [216.16610738 196.42980984  42.84535794]
 [160.60984747 243.70430827 156.77950227]
 [201.32821851 153.49884916 168.77198097]]
```

3.2. K-Means Clustering을 거친 layer로 부터 사용자가 정의한 RGB값으로 새롭게 labeled data를 생성



3.3. Image Segmentation 결과 생긴 masked image에서 원하는 클래스 부분에 해당하는 마스크를 찾기



4. 연구 결론 및 제언

- K-Means Clustering을 활용하여 라벨 디테일 (클래스와 RGB값)이 제공되지 않는 Image Segmentation용 데이터셋의 각 클래스를 대표하는 Centroid 즉 RGB 값을 추출함에 성공하였다. 이는 segmented image만 있는 상황에서도 직접 클래스를 구분지을 수 있게 되며, Image Segmentation 데이터셋을 전처리하는 과정에 활용 될 수 있다.

- 기존 데이터셋에서 제공하는 labeled data가 아닌 K-Means Clustering을 거친 layer로부터 사용자가 정의한 RGB값으로 새롭게 labeled data를 생성에 성공하였다.
- 이 과정에서 데이터셋의 개수를 조절할 수 있으며, 클래스별 정보를 가지고 있는 layer로 모델학습에 사용할 수 있으며, 후에 데이터 분석 과정에 매우 유용하게 사용될 수 있다.
- Image Segmentation 결과 생긴 masked image에서 원하는 클래스 부분에 해당하는 마스크를 찾아낼 때 K-Means 클러스터링을 이용하여 매우 높은 정확도로 찾을 수 있다. 이는 HSV 공간 상에서 Masking을 하는 것보다 간단한 조작으로 Semantic Segmenation의 결과를 효율적으로 분석 할 수 있으며 Image Segmentation과 관련된 데이터 분석에 매우 유용하게 활용될 수 있다.
- 이 연구에 사용된 대부분의 소스코드는 아래 github에서 확인해 볼 수 있다.
  https://github.com/taegukang35/kmeans_segmentation/blob/main/semanticseg.ipynb

5. 참고 문헌

가. K-Means Clustering 알고리즘 설명 및 구현 https://brunch.co.kr/@elicecoding/36

나. Cityscapes-Image-Pairs https://www.kaggle.com/dansbecker/cityscapes-image-pairs

다. U-Net Semantic Segmentation https://arxiv.org/abs/1505.04597

라. Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning