Pytorch Computer Vision Cookbook

2021210088 허지혜

Chapter4. Single-Object Detection

Object Detection은 객체 검출, 감지로 이미지 안에서 특정 물체의 위치를 찾는 방법이다. 이미지 안의 물체의 수에 따라 single-object나 multi-object로 detection 문제를 나눌 수 있다. 여기에서는 single-object detection을 Pytorch로 구현한다. single-object detection은 하나의 주어진 이미지 안에서 하나의 물체 찾기를 시도하고, 그 물체의 위치를 bounding box로 정의할 수 있다.

- bounding box 표현
[x0,y0,w,h]
[x0,y0,x1,y1]
[xc,yc,w,h]
이에 대해 간략하게 설명하면 다음과 같다.
x0, y0 : bounding box 좌표의 왼쪽 위 값
x1, y1 : bounding box의 오른쪽 아래 값
w,h : bounding box의 width(너비)와 height(높이)
xc, yc : bounding box의 centroid(중심)

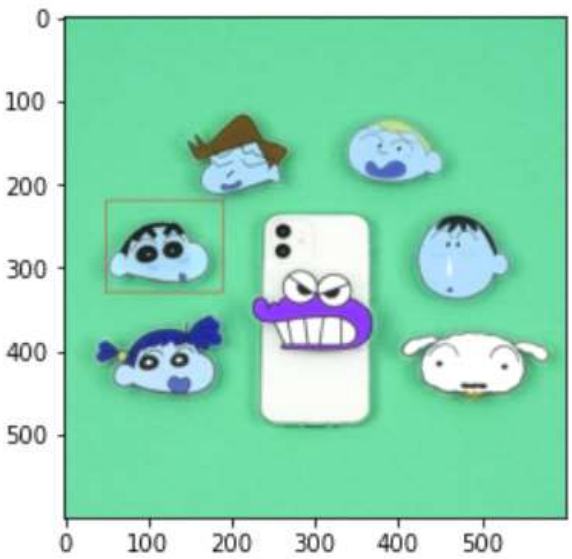


그림 1 . 짱구 이미지에서 짱구 bounding box

그러므로 single-object detection의 목표는 4개의 숫자를 이용한 **bounding box를 예측**하는 것이다. 정사각형 물체에 대하여, 우리는 width와 height를 고정할 수

있으므로 간단하게 2개의 숫자를 예측하는 문제가 될 것이다. 요 단원에서는 두 숫자를 사용하여 눈 이미지의 fovea를 locate하는 방법을 배울 것이다.

1. Explorary data analysis

Explorary data analysis(데이터 분석 탐색, 탐색적 자료 분석)은 데이터의 특징을 이해하기 위해 꼭 필요한 단계이다. 이 단계에서 우리는 우리의 데이터를 검사하고 통계적인 특징을 boxplot, histogram 등의 시각화 툴이나 샘플로 나타내야 한다. 예로 tabular data에서 우리는 컬럼과 행, 기록의 수, 표준 편차나 평균 같은 통계적인 측정 항목 등등을 살펴본다.
여기에서 사용할 데이터는 iChallenge-AMD이다.

- 데이터 불러오기

```
import os
import pandas as pd

path2data="./data/"
path2labels=os.path.join(path2data,"Training400","Fovea_location.xlsx")
```

./Training400/Foreva_loacation.xlsx 데이터는 다음과 같다.

	ID	imgName	Fovea_X	Fovea_Y
0	1	A0001.jpg	1182.264278	1022.018842
1	2	A0002.jpg	967.754046	1016.946655
2	3	A0003.jpg	1220.206714	989.944033
3	4	A0004.jpg	1141.140888	1000.594955
4	5	A0005.jpg	1127.371832	1071.109440
...	...	...	...	...
395	396	N0307.jpg	823.024991	690.210211
396	397	N0308.jpg	647.598978	795.653188
397	398	N0309.jpg	624.571803	755.694880
398	399	N0310.jpg	687.523044	830.449187
399	400	N0311.jpg	746.107631	759.623062

400 rows × 4 columns

그림 1. Foreva_location.xlsx 데이터

ID를 기준으로 DataFrame으로 불러오면 다음과 같다.

```
labels_df=pd.read_excel(path2labels,index_col="ID")
labels_df.head()
```

ID	imgName	Fovea_X	Fovea_Y
1	A0001.jpg	1182.264278	1022.018842
2	A0002.jpg	967.754046	1016.946655
3	A0003.jpg	1220.206714	989.944033
4	A0004.jpg	1141.140888	1000.594955
5	A0005.jpg	1127.371832	1071.109440

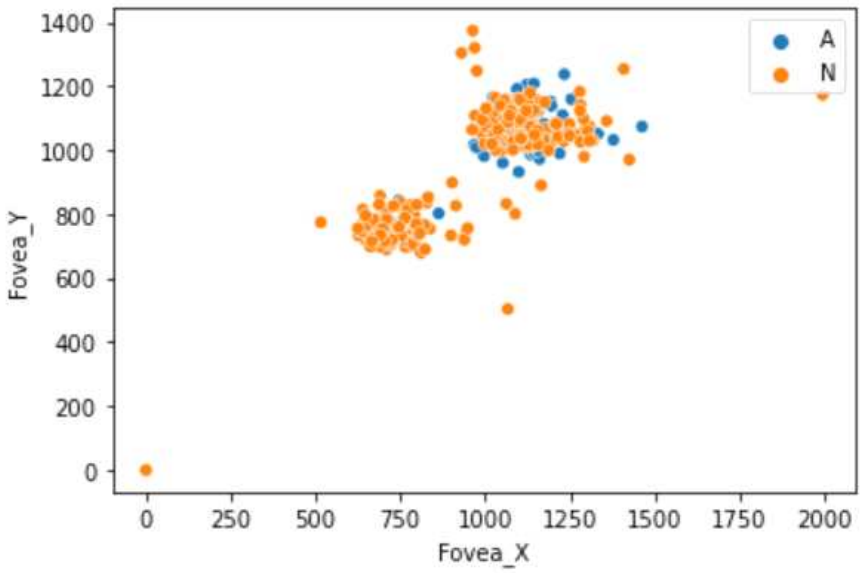
labels_df 데이터는 ImgName 첫 글자에 따라 A로 분류되는 이미지와 N으로 분류되는 데이터로 나뉜다. 먼저 A와 N label의 분포를 살펴보자.

```
import seaborn as sns
%matplotlib inline

AorN=[imn[0] for imn in labels_df['imgName']]
#print(AorN)
print(AorN.count('A'))
print(AorN.count('N'))
```

출력 결과 A : N = 89 : 311이다.
다음은 Fovea 위치를 화면에 출력시켜보자.

```
sns.scatterplot(x=labels_df['Fovea_X'], y=labels_df['Fovea_Y'],hue=AorN)
# 조건 = AorN
```

<AxesSubplot:xlabel='Fovea_X', ylabel='Fovea_Y'>



- img, label 시각화
- 시각화 기본 설정 및 랜덤 이미지 index 가져오기

```
import numpy as np
from PIL import Image, ImageDraw
import matplotlib.pylab as plt
np.random.seed(2019)

# size 설정, subplot 너비,높이 비율 조정
plt.rcParams['figure.figsize'] = (15, 9)
plt.subplots_adjust(wspace=0, hspace=0.3)
nrows,ncols=2,3

imgName=labels_df["imgName"]
ids=labels_df.index
print(ids) # 400

# 0 ~ (ids-1)에서 nrows*nrows(6) 만큼 난수 생성
rndIds=np.random.choice(ids,nrows*ncols)
print(rndIds) # [73 371 160 294 217 191]
```

- labels_df에서 img, label load 함수 정의

```
def load_img_label(labels_df,id_):
    imgName=labels_df["imgName"]
```

```
    if imgName[id_][0]=="A":
        prefix="AMD"
    else:
        prefix="Non-AMD"

    fullPath2img=os.path.join(path2data,"Training400",prefix,imgName[id_])
    img = Image.open(fullPath2img)

    # 중심
    x=labels_df["Fovea_X"][id_]
    y=labels_df["Fovea_Y"][id_]

    label=(x,y)
    return img,label
```

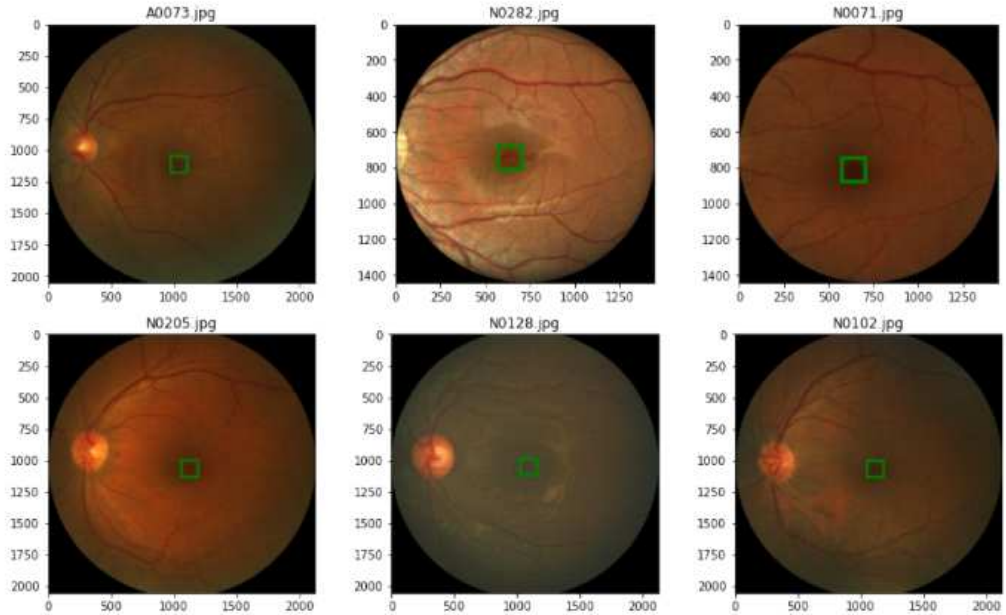
- bounding box 화면 출력 함수 정의

```
def show_img_label(img,label,w_h=(50,50),thickness=2):
    w,h=w_h
    cx,cy=label

    # img에 사각형(바운딩 박스) 그리기
    draw = ImageDraw.Draw(img)
    # (left, top), (right, bottom)
    draw.rectangle(((cx-w/2, cy-h/2), (cx+w/2, cy+h/2)), outline="green", width=
    thickness)

    plt.imshow(np.asarray(img))
```

- 시각화

```
# 아까 랜덤으로 고른 수에 대하여 시각화
for i,id_ in enumerate(rndIds):

    img,label=load_img_label(labels_df,id_)
    print(img.size,label)

    plt.subplot(nrows, ncols, i+1)
    show_img_label(img,label,w_h=(150,150),thickness=20)
    plt.title(imgName[id_])
```

```
(2124, 2056) (1037.89889229694, 1115.71768088143)
(1444, 1444) (635.148992978281, 744.648850248249)
(1444, 1444) (639.360312038611, 814.762764100936)
(2124, 2056) (1122.08407442503, 1067.58829793991)
(2124, 2056) (1092.93333646222, 1055.15333296773)
(2124, 2056) (1112.50135915347, 1070.7251775623)
```



- widths, heights 리스트 안에 값 넣기

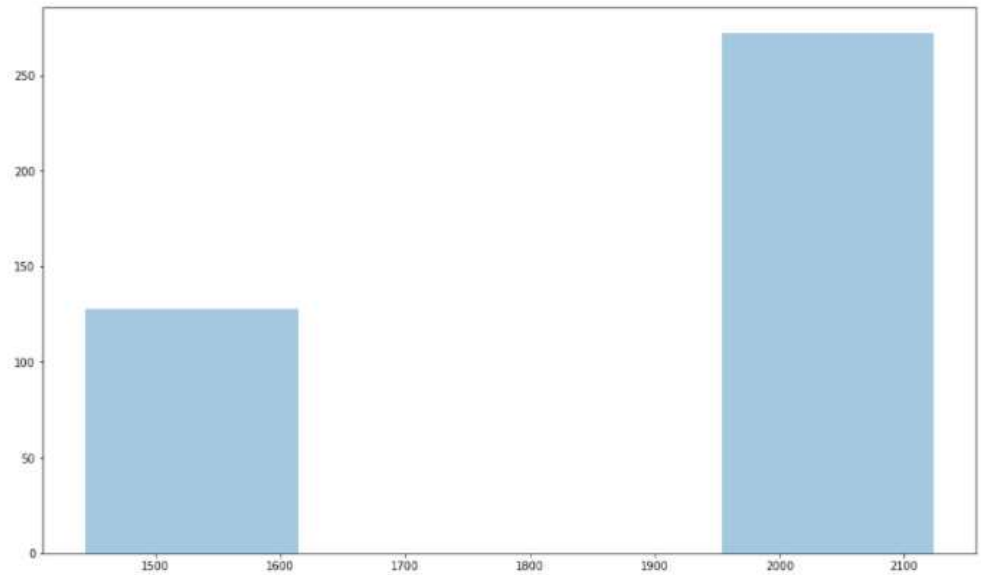
```
# widths, heights를 두 개의 리스트로 만든다.
h_list,w_list=[],[]
for id_ in ids:
    if imgName[id_][0]=="A":
        prefix="AMD"
    else:
        prefix="Non-AMD"

    fullPath2img=os.path.join(path2data,"Training400",prefix,imgName[id_])



    img = Image.open(fullPath2img)
    h,w=img.size
    h_list.append(h)
    w_list.append(w)
```

각각 h_list와 w_list의 분포를 살펴본다.

- h의 분포

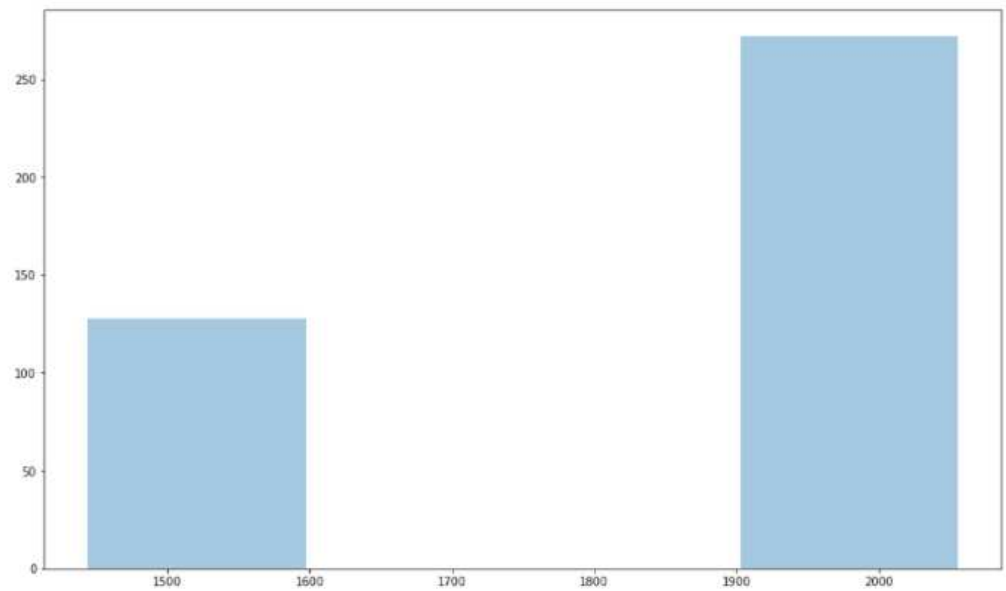
```
sns.distplot(a=h_list, kde=False) # kde : 파란색 선
```

<AxesSubplot:>



- w의 분포

<AxesSubplot:>



<Exploratory 요약>

- Fovea.xlxs 파일 컬럼

imgName	이미지 이름 의미
Fovea_X	Fovea 중심의 X 좌표

Fovea_Y	Fovea 중심의 Y 좌표

- Fovea 파일 라벨의 분포

letter 'A' AMD 파일	89
letter 'N' Non-AMD 파일	311

- 특징

* scatter plot을 보니 AMD와 Non-AMD 파일의 Fovea Location 사이에는 상관관계가 없다.
* 중심좌표의 위치가 (0,0)인 이미지도 있다. 하지만 데이터 개수가 400개 밖에 없어서 그냥 두기로 결정하였다.
* 사진의 크기인 widths와 heights의 많은 양이 1900 ~ 2100 사이에 있다.

2. Data Transformation for Object Detection

Data augmentation과 Data transformation은 딥러닝 알고리즘을 학습시킬 때 중요한 과정이다. 특히, 400개처럼 조그만 데이터를 다룰 때 필수적이다.
위 단계에서의 데이터 처리는 다음과 같다.
1) resize
2) augmentation skill use(horizontal flipping, vertical flipping, translation 등)

- resize 함수 정의

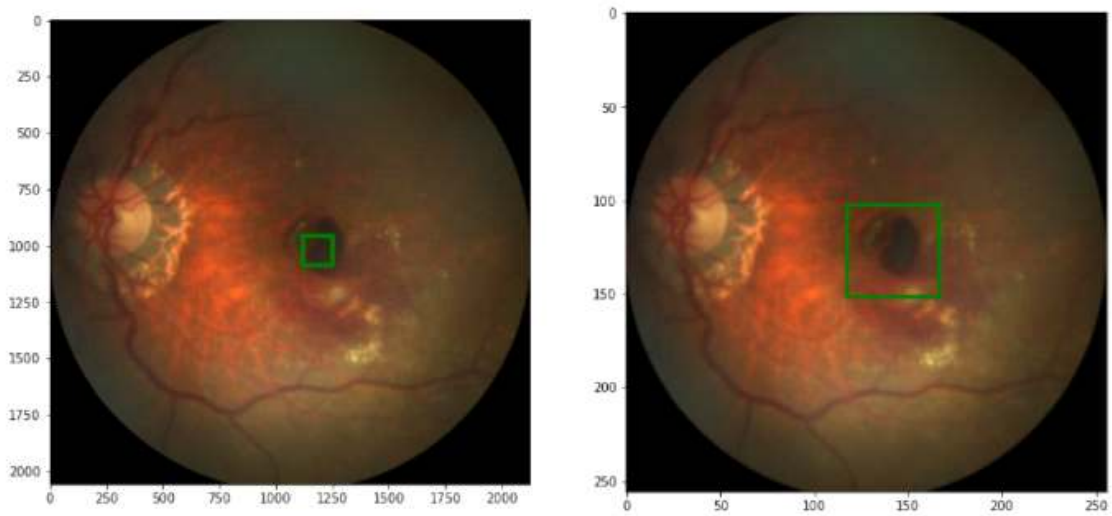
```
import torchvision.transforms.functional as TF

# resize 함수 정의
def resize_img_label(image,label=(0.,0.),target_size=(256,256)):
    w_orig,h_orig = image.size
    w_target,h_target = target_size
    cx, cy= label

    image_new = TF.resize(image,target_size)
    # 중심좌표/이미지크기*새 이미지 크기
    label_new= cx/w_orig*w_target, cy/h_orig*h_target

    return image_new,label_new
```

- 함수 적용

```
img, label=load_img_label(labels_df,1) #id=1인거
```

```
print(img.size,label)

img_r,label_r=resize_img_label(img,label)
print(img_r.size,label_r)

plt.subplot(1,2,1) # 행 열, index
show_img_label(img,label,w_h=(150,150),thickness=20)
plt.subplot(1,2,2)
show_img_label(img_r,label_r)
```

```
(2124, 2056) (1182.26427759023, 1022.01884158854)
(256, 256) (142.4951295024006, 127.25526432230848)
```

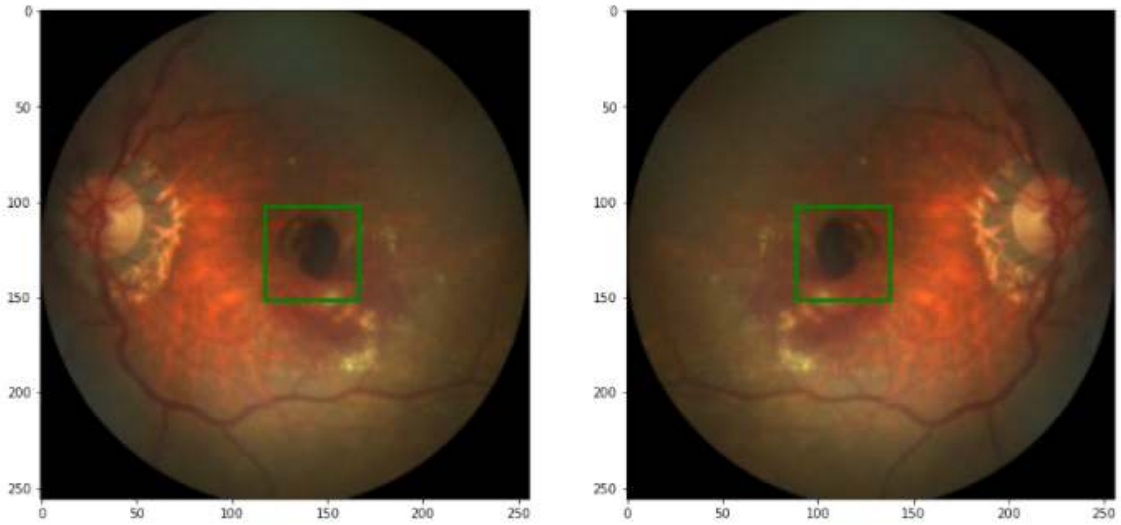


- 수평 뒤집기(horizontal flipping) 함수 정의

```
def random_hflip(image,label):
    w,h=image.size
    x,y=label

    image = TF.hflip(image)
    label = w-x, y
    return image,label
```

```
img, label=load_img_label(labels_df,1)
img_r,label_r=resize_img_label(img,label)
img_fh,label_fh=random_hflip(img_r,label_r)

plt.subplot(1,2,1)
```

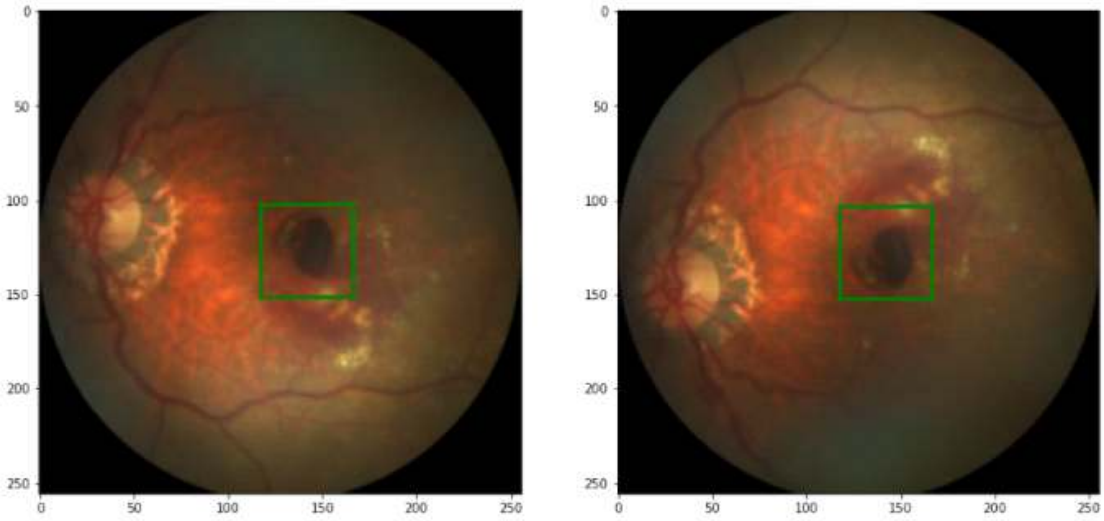
```
show_img_label(img_r,label_r)
plt.subplot(1,2,2)
show_img_label(img_fh,label_fh)
```



- 수평 뒤집기(vertical flipping) 함수 정의

```
# 수직 뒤집기 함수 정의
def random_vflip(image,label):
    w,h=image.size
    x,y=label

    image = TF.vflip(image)
    label = x, h-y
    return image, label
```

```
img, label=load_img_label(labels_df,1)
img_r,label_r=resize_img_label(img,label)
img_fv,label_fv=random_vflip(img_r,label_r)

plt.subplot(1,2,1)
show_img_label(img_r,label_r)
plt.subplot(1,2,2)
show_img_label(img_fv,label_fv)
```



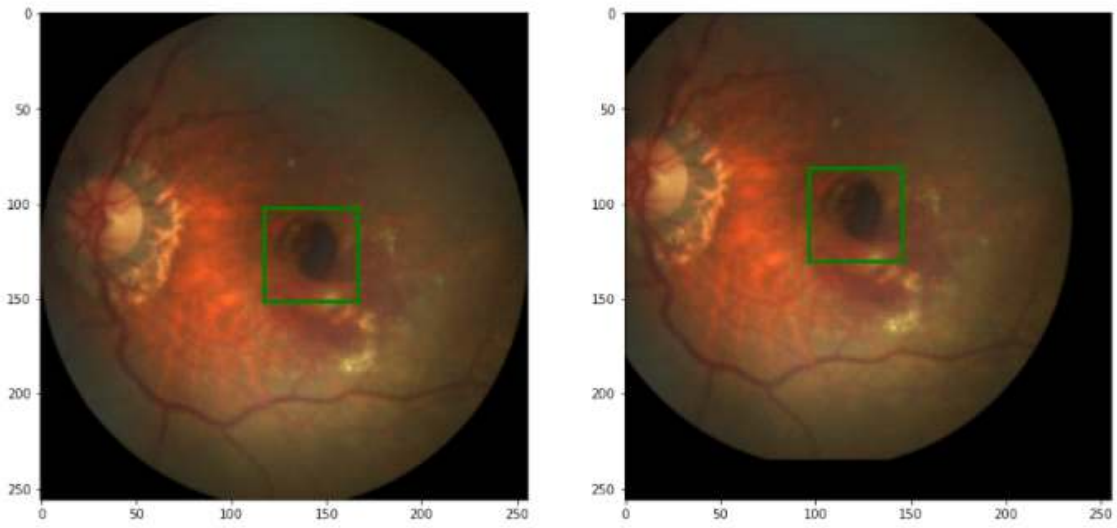
- Shift 또는 Translate 함수 정의

affine 변환이란 회전, 평행이동, 스케일 뿐 아니라 shearing(전단), reflection(대칭)까지 포함한 변환이다.

```
import numpy as np
np.random.seed(1)

# shift 또는 translate 함수 정의
def random_shift(image,label,max_translate=(0.2,0.2)):
    w,h=image.size
    max_t_w, max_t_h=max_translate
    cx, cy=label

    # rand() 값 범위 [0,1] -> translate 계수 범위 [-1,1]
    trans_coef=np.random.rand()*2-1
    w_t = int(trans_coef*max_t_w*w)
    h_t = int(trans_coef*max_t_h*h)

    image=TF.affine(image,translate=(w_t, h_t),shear=0,angle=0,scale=1)
    label = cx+w_t, cy+h_t

    return image,label
```

```
img, label=load_img_label(labels_df,1)
img_r,label_r=resize_img_label(img,label)
img_t,label_t=random_shift(img_r,label_r,max_translate=(.5,.5))

plt.subplot(1,2,1)
```

```
show_img_label(img_r,label_r)
plt.subplot(1,2,2)
show_img_label(img_t,label_t)
```



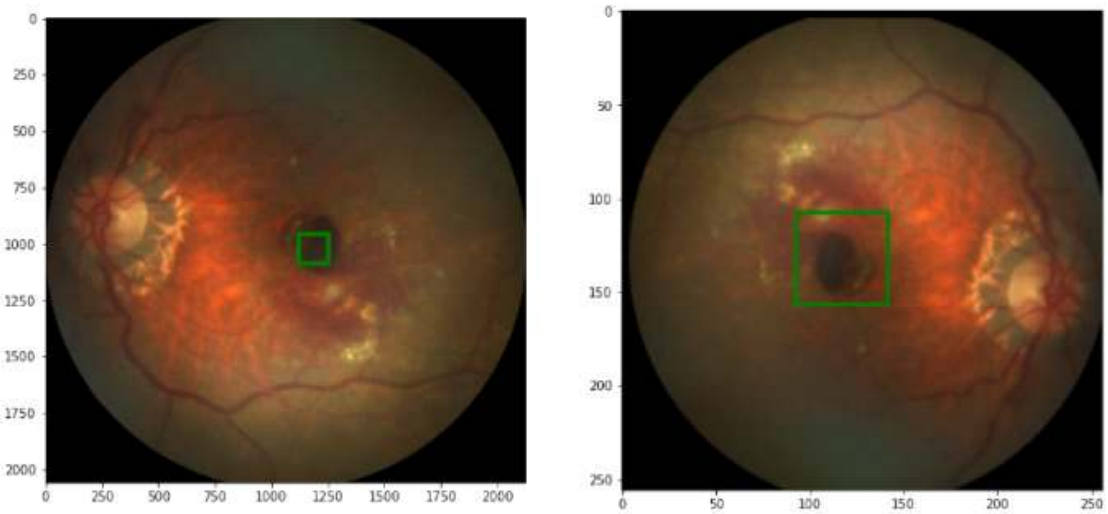
- 위 정의 함수를 하나의 함수로 나타내자.

```
# 정의 함수를 하나의 함수로 나타내자!
def transformer(image, label, params):
    image,label=resize_img_label(image,label,params["target_size"])

    # 만약에 변환을 안 원하면 params를 조정하라고
    # if문 넣어둠.
    # random.random()의 범위는 [0,1)
    if random.random() < params["p_hflip"]:
        image,label=random_hflip(image,label)

    if random.random() < params["p_vflip"]:
        image,label=random_vflip(image,label)

    if random.random() < params["p_shift"]:
        image,label=random_shift(image,label, params["max_translate"])

    image=TF.to_tensor(image)
    return image, label
```

```
import random
np.random.seed(0)
random.seed(0)
```

```
img, label=load_img_label(labels_df,1)

params={
    "target_size" : (256, 256),
    "p_hflip" : 1.0,
    "p_vflip" : 1.0,
    "p_shift" : 1.0,
    "max_translate": (0.2, 0.2),
}
img_t,label_t=transformer(img,label,params)
```

```
plt.subplot(1,2,1)
show_img_label(img,label,w_h=(150,150),thickness=20)
plt.subplot(1,2,2)
show_img_label(TF.to_pil_image(img_t),label_t) # tensor -> pil data type
```



<Data Transformation for Object Detection Summary>

- 이미지 크기가 달라서 resize (256,256)
- horizontal, vertically flipping
- left, right, up, down에 대해 shift or translate 변환 :

변환 값은 랜덤하게 지정하지만 변환할 수 있는 **최대 차원 (widths,heights)** = (0.2,0.2)이다. 이유는 예로 img (256,256) 에 대하여 각각 차원에 대하여 최대 변환 값은 51 pixel 이다.

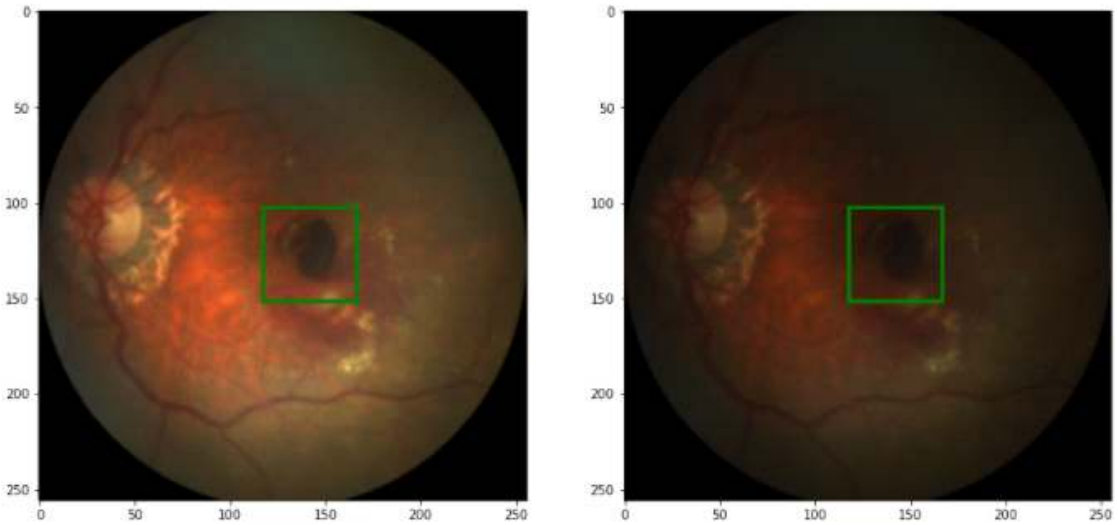
이렇게 차원을 변환 후 object가 화면 밖으로 나갈 수도 있다. **안전한 값은 [0.1, 0.2]** 사이이다. 하지만 확실하게 이 값을 조정해야 한다. 또한 랜덤하게 변환된 이미지에 대하여, 우리는 일반적으로 랜덤 변환 값 범위를 [-1,1]로 하고 가장 큰 변환 값에 대하여 곱한다.

- affine 함수를 사용하여 다양한 type의 변환을 수행할 수 있다. 예로 rotation, shearing, scaling이다. 여기서는 translation만 한다.
- 변환 방법을 하나의 함수로 나타내었다. 5가지 변환 방법이 순차적으로 적용이 되고 마지막으로 PIL image로 적용을 해주었다.

3. There's more

다른 변환 방법

```
img, label=load_img_label(labels_df,1)
img_r,label_r=resize_img_label(img,label)

# adjust brightness(명도)
# factor 1을 기준으로 내려가면 -
img_t=TF.adjust_brightness(img_r,brightness_factor=0.5)
label_t=label_r

plt.subplot(1,2,1)
show_img_label(img_r,label_r)
plt.subplot(1,2,2)
show_img_label(img_t,label_t)
```




```
# brightness contrast(대조)
img_t=TF.adjust_contrast(img_r,contrast_factor=0.4)

# gamma correction(보정)
img_t=TF.adjust_gamma(img_r,gamma=1.4)
```

- 스케일링

object detection 작업의 경우 더 나은 모형을 위해 레이블을 [0,1] 범위로 만드는

것이 중요하다.

```
# label scale [0,1]
def scale_label(a,b):
    div = [ai/bi for ai,bi in zip(a,b)]
    return div
```

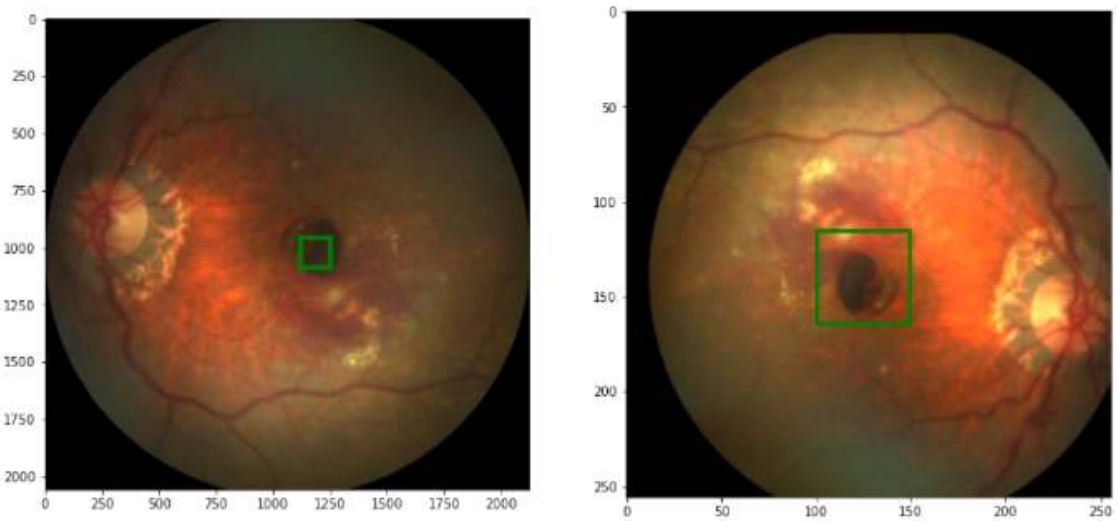
```
def transformer(image, label, params):
    image,label=resize_img_label(image,label,params["target_size"])

    if random.random() < params["p_hflip"]:
        image,label=random_hflip(image,label)

    if random.random() < params["p_vflip"]:
        image,label=random_vflip(image,label)

    if random.random() < params["p_shift"]:
        image,label=random_shift(image,label, params["max_translate"])

    if random.random() < params["p_brightness"]:
        brightness_factor=1+(np.random.rand()*2-1)*params["brightness_factor"]
        image=TF.adjust_brightness(image,brightness_factor)

    if random.random() < params["p_contrast"]:
        contrast_factor=1+(np.random.rand()*2-1)*params["contrast_factor"]
        image=TF.adjust_contrast(image,contrast_factor)

    if random.random() < params["p_gamma"]:
        gamma=1+(np.random.rand()*2-1)*params["gamma"]
        image=TF.adjust_gamma(image,gamma)

    if params["scale_label"]:
        label=scale_label(label,params["target_size"])

    image=TF.to_tensor(image)
    return image, label
```

```
np.random.seed(0)
random.seed(0)

img, label=load_img_label(labels_df,1)
```

```
params={
```

```
    "target_size" : (256, 256),
    "p_hflip" : 1.0,
    "p_vflip" : 1.0,
    "p_shift" : 1.0,
    "max_translate": (0.5, 0.5),
    "p_brightness": 1.0,
    "brightness_factor": 0.8,
    "p_contrast": 1.0,
    "contrast_factor": 0.8,
    "p_gamma": 1.0,
    "gamma": 0.4,
    "scale_label": False,
}
img_t,label_t=transformer(img,label,params)
```

```
plt.subplot(1,2,1)
show_img_label(img,label,w_h=(150,150),thickness=20)
plt.subplot(1,2,2)
show_img_label(TF.to_pil_image(img_t),label_t)
```



- 마지막으로 labels에 rescale해서 원래 이미지의 size로 돌려준다.

```
def rescale_label(a,b):
    div = [ai*bi for ai,bi in zip(a,b)]
    return div
```

4. Creating Custom Datasets

이번 단계에서는 dataset을 정의를한다. 이때 우리는 dataset을 sub-classing을 사용하고 init와 getitem을 재정의한다. __len__은 데이터의 길이를 반환하는데

파이썬의 len 함수로도 호출이 가능하다. __getitem__은 이미지의 지정된 index를 반환한다.
후에 DataLoader를 사용하여 만들어내고 그를 이용하여 데이터의 mini-batchs를 얻을 수 있다.

```
from torch.utils.data import Dataset
from PIL import Image
```

- AMD_dataset class 정의

```
class AMD_dataset(Dataset):
    def __init__(self, path2data, transform, trans_params):
        pass

    def __len__(self):

        return len(self.labels)

    def __getitem__(self, idx):
        pass
```

- init 새로 정의

```
def __init__(self, path2data, transform, trans_params):

    # full path of the labels file
    path2labels=os.path.join(path2data,"Training400_labels","Fovea_location.xlsx")

    # read and extract labels
    labels_df=pd.read_excel(path2labels,index_col="ID")
    self.labels = labels_df[["Fovea_X","Fovea_Y"]].values

    # extract ID and imgName columns
    self.imgName=labels_df["imgName"]
    self.ids=labels_df.index

    self.fullPath2img=[0]*len(self.ids) # ids 길이만큼 0
    for id_ in self.ids:
        if self.imgName[id_][0]=="A":
            prefix="AMD"
        else:
            prefix="Non-AMD"

self.fullPath2img[id_-1]=os.path.join(path2data,"Training400",prefix,self.imgName[id
_])

    self.transform = transform
    self.trans_params=trans_params
```

- getitem 새로 정의

```
def __getitem__(self, idx):

    image = Image.open(self.fullPath2img[idx])
    label= self.labels[idx]



    image,label = self.transform(image,label,self.trans_params)

    return image, label
```

- AMD_dataset init, getitem 재정의

```
AMD_dataset.__init__=__init__
AMD_dataset.__getitem__=__getitem__
```

```
trans_params_train={
    "target_size" : (256, 256),
    "p_hflip" : 0.5,
    "p_vflip" : 0.5,
    "p_shift" : 0.5,
    "max_translate": (0.2, 0.2),
    "p_brightness": 0.5,
    "brightness_factor": 0.2,
    "p_contrast": 0.5,
    "contrast_factor": 0.2,
    "p_gamma": 0.5,
    "gamma": 0.2,
    "scale_label": True,
}

trans_params_val={
    "target_size" : (256, 256),
    "p_hflip" : 0.0,
    "p_vflip" : 0.0,
    "p_shift" : 0.0,
    "p_brightness": 0.0,
    "p_contrast": 0.0,
    "p_gamma": 0.0,
    "gamma": 0.0,
    "scale_label": True,
}
amd_ds1=AMD_dataset(path2data,transformer,trans_params_train)
amd_ds2=AMD_dataset(path2data,transformer,trans_params_val)
```

```
from sklearn.model_selection import ShuffleSplit

# 분할 1회
sss = ShuffleSplit(n_splits=1, test_size=0.2, random_state=0)

indices=range(len(amd_ds1))

for train_index, val_index in sss.split(indices):
    print(len(train_index)) # 320
```

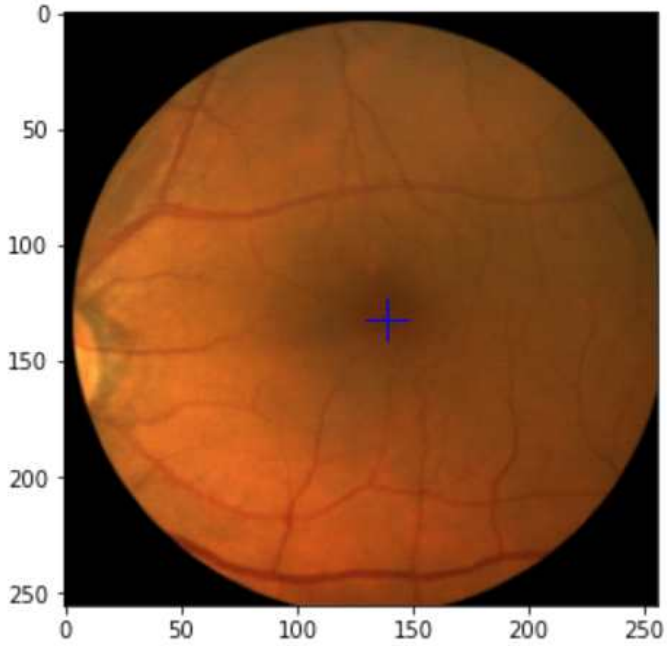
```
    print("-"*10)
    print(len(val_index)) # 80
```

```
from torch.utils.data import Subset

train_ds=Subset(amd_ds1,train_index)
print(len(train_ds)) # 320

val_ds=Subset(amd_ds2,val_index)
print(len(val_ds)) # 80
```

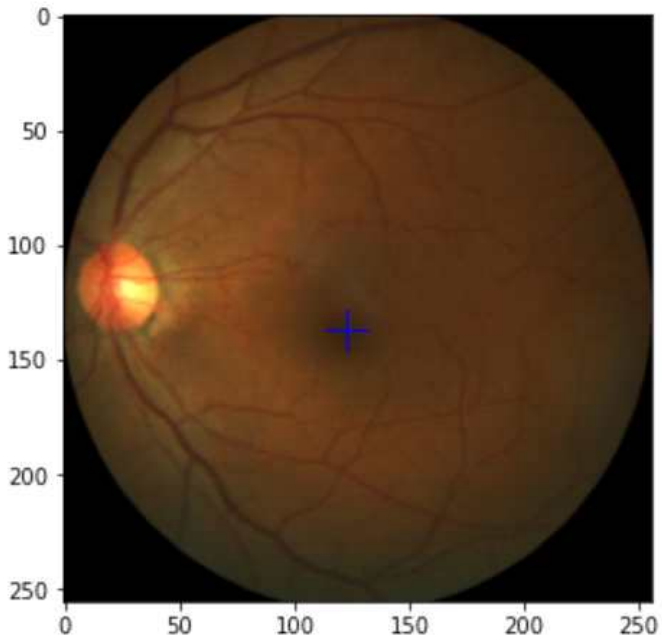
- train_ds, val_ds 시각화

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
np.random.seed(0)

def show(img,label=None):
    # 파이토치 영상  -> 넘파이 영상 출력(1,2,0)
    npimg = img.numpy().transpose((1,2,0)) # c,w,h -> w,h,c
    plt.imshow(npimg)
    if label is not None:
        label=rescale_label(label,img.shape[1:])
        x,y=label
        plt.plot(x,y,'b+',markersize=20)
```

```
plt.figure(figsize=(5,5))
for img,label in train_ds:
    show(img,label)
    break
```




```
plt.figure(figsize=(5,5))
for img,label in val_ds:
    show(img,label)
    break
```




```
from torch.utils.data import DataLoader
```

```
train_dl = DataLoader(train_ds, batch_size=8, shuffle=True)
val_dl = DataLoader(val_ds, batch_size=16, shuffle=False)

for img_b, label_b in train_dl:
    print(img_b.shape,img_b.dtype)
    print(label_b)
    break
```

```
torch.Size([8, 3, 256, 256]) torch.float32
[tensor([0.5187, 0.4810, 0.4935, 0.4977, 0.4592, 0.3723, 0.33
86, 0.6502],
       dtype=torch.float64), tensor([0.5125, 0.4921, 0.4809,
0.4856, 0.5343, 0.3251, 0.3576, 0.4998],
       dtype=torch.float64)]
```

요기에서 label_b 값을 살펴보면 **list 형태**이다.

우리는 출력값이 torch.float 형태로 나와야 하기 때문에 stack 함수를 이용하여 수정한다.

Q.위 label_b와 아래 label_b의 값이 다르다.

```
import torch

for img_b, label_b in train_dl:
    print(img_b.shape,img_b.dtype)
    # 지정한 차원으로 확장하여 tensor 바꿔주기
    label_b=torch.stack(label_b,1) [] stack -> [[]]
    print(label_b)
    label_b=label_b.type(torch.float32)
    print(label_b.shape,label_b.dtype)
    break
```

```
torch.Size([8, 3, 256, 256]) torch.float32
tensor([[0.5059, 0.5302],
        [0.4607, 0.5067],
        [0.5125, 0.5000],
        [0.4717, 0.4140],
        [0.4660, 0.4362],
        [0.5307, 0.5227],
        [0.3670, 0.4275],
        [0.5373, 0.4867]], dtype=torch.float64)
torch.Size([8, 2]) torch.float32
```

* cat vs stack 차이점 *

```
for img_b, label_b in val_dl:
    print(img_b.shape,img_b.dtype)

    # convert to tensor
    label_b=torch.stack(label_b,1)
    label_b=label_b.type(torch.float32)
    print(label_b.shape,label_b.dtype)
    break
```

```
torch.Size([16, 3, 256, 256]) torch.float32
torch.Size([16, 2]) torch.float32
```

<Creating Custom Datasets Summary>

- custom dataset 정의 -> 기존의 __init__, __getitem__을 새로 정의하여 정의된 custon dataset를 재정의한다.
- 두 개의 물체(A,N)에 대하여 train, val 그룹을 8:2로 나누어준다. 그리고 각각의 trans_params를 따로 지정해준다. 이때 train은 transformation의 확률을 다 0.5로 지정하여 모두가 적용되게 하였고 val은 0.0으로 지정하여 모두 적용되지 않게 하였다.
- 이미지를 train, val의 index에 따라 나눠질 수 있게 하였다.
- data loader를 정의하여 각 값을 살펴보았다.

5. Creating the model

이번 단계에서 single-object detection 문제에 대해 모형을 정의한다.
우리의 목표는 eye image에서 x,y 좌표를 예측하는 것이다.
여기에서는 ResNet을 이용하여 예측을 할 것이다. 마지막 단계에 x,y 좌표 예측값이 나오도록 설계하고 input에 RGB 이미지로 resize를 한다.

- Net class 정의

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self, params):
        super(Net, self).__init__()
```

```
    def forward(self, x):
        return x
```

- init 함수 재정의

```
def __init__(self, params):
    super(Net, self).__init__()

    C_in,H_in,W_in=params["input_shape"]
    init_f=params["initial_filters"]
    num_outputs=params["num_outputs"]

    self.conv1 = nn.Conv2d(C_in, init_f, kernel_size=3,stride=2,padding=1)
    self.conv2 = nn.Conv2d(init_f+C_in, 2*init_f, kernel_size=3,stride=1,padding=1)
    self.conv3 = nn.Conv2d(3*init_f+C_in, 4*init_f, kernel_size=3,padding=1)
    self.conv4 = nn.Conv2d(7*init_f+C_in, 8*init_f, kernel_size=3,padding=1)
    self.conv5 = nn.Conv2d(15*init_f+C_in, 16*init_f, kernel_size=3,padding=1)
    self.fc1 = nn.Linear(16*init_f, num_outputs)
```

- forward 함수 재정의

```
def forward(self, x):
    identity=F.avg_pool2d(x,4,4)
    x = F.relu(self.conv1(x))
    x = F.max_pool2d(x, 2, 2)
    x = torch.cat((x, identity), dim=1)

    identity=F.avg_pool2d(x,2,2)
    x = F.relu(self.conv2(x))
    x = F.max_pool2d(x, 2, 2)
    x = torch.cat((x, identity), dim=1)

    identity=F.avg_pool2d(x,2,2)
    x = F.relu(self.conv3(x))
    x = F.max_pool2d(x, 2, 2)
    x = torch.cat((x, identity), dim=1)

    identity=F.avg_pool2d(x,2,2)
    x = F.relu(self.conv4(x))
```

```
    x = F.max_pool2d(x, 2, 2)
    x = torch.cat((x, identity), dim=1)

    x = F.relu(self.conv5(x))

    x=F.adaptive_avg_pool2d(x,1)  resnet 특징 중 가장 중요
    x = x.reshape(x.size(0), -1)

    x = self.fc1(x)
    return x
```

- Net class에서 init, forward 함수 재정의

```
Net.__init__=__init__
Net.forward=forward
```

```
params_model={
          "input_shape": (3,256,256),
          "initial_filters": 16,
          "num_outputs": 2,
              }

model = Net(params_model)
```

```
if torch.cuda.is_available():
    device = torch.device("cuda")
    model=model.to(device)
print(model)
```

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2), p
adding=(1, 1))
  (conv2): Conv2d(19, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  (conv3): Conv2d(51, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  (conv4): Conv2d(115, 128, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1))
  (conv5): Conv2d(243, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1))
  (fc1): Linear(in_features=256, out_features=2, bias=True)
)
```

<Creating the model summary>

- 우리는 Net class를 정의해주었다. 첫 번째로 필수 패키지를 load 하였다. 그리고 init, forward를 정의하였다. init 안에는 blocks를 쌓았다. 요 블록 안에는 전부 padding = 1로 설정하여 input의 양쪽을 0으로 채워주었다. 그리고 output size를 2로 나누어 유지하였다. (input에 채웠으니 output으로 조절)
- channel은 skip connections 때문에 이전 layer와 skip layer로부터 output channels는 더해진다.

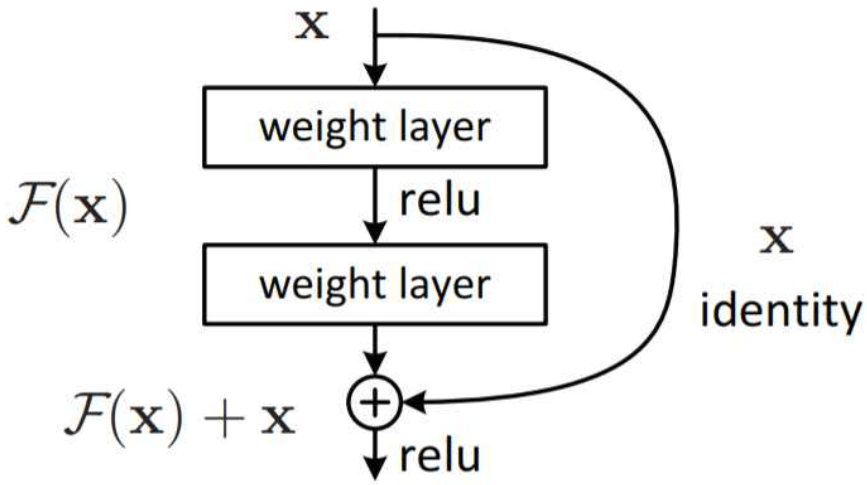


Figure 2. Residual learning: a building block.

- forward 함수를 정의하였다. 이때 두 개의 tensor를 연결하는 경우 연결 차원을 제외하고는 shape이 같아야 한다. 요기 tensor의 shape은 B*C*H*W이다.
- 마지막 layer에는 skip connection이 없다. 추출된 특징을 나타낸다. 1*1 의 output size를 얻기 위해 추출된 특징에 대해 adaptive_avg_pool2d를 이용하여 adaptive 평균 pooling을 수행한다.
- 마지막 layer는 활성화가 따로 필요 없다.

6. Defining Loss, Optimizer and IOU Metric

- Loss 정의

```
loss_func = nn.SmoothL1Loss(reduction="sum")
```

https://pytorch.org/docs/1.9.1/generated/torch.nn.SmoothL1Loss.html

default : beta = 1.0

$$l_n = \begin{cases} 0.5(x_n - y_n)^2/beta, & \text{if } |x_n - y_n| < beta \\ |x_n - y_n| - 0.5 * beta, & \text{otherwise} \end{cases}$$

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction} = \text{'mean'}; \\ \text{sum}(L), & \text{if reduction} = \text{'sum'}. \end{cases}$$

mse,mae의 장점을 혼합한 loss = smoothL1Loss

밑에는 그 예시 ~

알려진 y, target으로 loss 값을 구해보면 다음과 같다.

```
n,c=8,2
# 1. 0.5 multiple
y = 0.5 * torch.ones(n, c, requires_grad=True)
print(y.shape)

target = torch.zeros(n, c, requires_grad=False)
print(target.shape)

loss = loss_func(y, target)
print(loss.item())

# 2. 2 multiple
y = 2 * torch.ones(n, c, requires_grad=True)
target = torch.zeros(n, c, requires_grad=False)
loss = loss_func(y, target)
print(loss.item())
```

```
torch.Size([8, 2])
torch.Size([8, 2])
2.0
24.0
```

- Optimizer 정의

```
from torch import optim
opt = optim.Adam(model.parameters(), lr=3e-4)
```

- learning rate를 읽어오는 함수 정의

```
def get_lr(opt):
    for param_group in opt.param_groups:
        return param_group['lr']
```

```
current_lr=get_lr(opt)
print('current lr={}'.format(current_lr)) # current lr= 0.0003
```

- learning rate scheduler 정의

```
from torch.optim.lr_scheduler import ReduceLROnPlateau
lr_scheduler = ReduceLROnPlateau (opt, mode='min',factor=0.5, patience=20, verbose=1)
```

https://koreapy.tistory.com/774

ReduceLROnPlateat는 더 이상 학습이 진행되지 않을 때(성능 향상이 없을 때) learning rate를 감소시키는 schedular이다. schedular에 input으로 metric를 주면, 일정 epoch 동안 변화가 없을 때 learning rate를 감소시킨다. 위 정의에서는 opt를 input으로 주었다.

① mode : Input으로 주는 metric이 낮을수록 좋은지, 높을수록 좋은지 의미

② factor : learning rate를 감소시키는 비율 new_lr = lr * factor이다.

③ patience(인내) : metric이 얼마 동안 변화가 없을 때 learning rate를 감소시킬지 결정한다. 위는 metric이 20 epoch 동안 변화가 없으면 learning rate를 감소시킨다.

④ verbose 0 - 아무것도 출력하지 않음

1 - 진행도를 보여줌

- 실행

```
for i in range(100):
    lr_scheduler.step(1)
```

```
Epoch    22: reducing learning rate of group 0 to 1.5000e-04.
Epoch    43: reducing learning rate of group 0 to 7.5000e-05.
Epoch    64: reducing learning rate of group 0 to 3.7500e-05.
Epoch    85: reducing learning rate of group 0 to 1.8750e-05.
```

patience를 20으로 설정했기 때문에 20 epoch동안 변화가 없으면 학습률을 감소시킨다.

다음으로 우리는 데이터의 batch에 대한 IOU를 계산하는 함수를 정의할 것이다.

- cxcy2bbox 정의 : 좌표를 bounding box로 변환하는 함수

```
def cxcy2bbox(cxcy,w=50./256,h=50./256):
    # w,h에 대하여 두 개의 새로운 tensor를 정의
    w_tensor=torch.ones(cxcy.shape[0],1,device=cxcy.device)*w
    h_tensor=torch.ones(cxcy.shape[0],1,device=cxcy.device)*h
```

```
    # cx, cy에서 가져옴
    cx=cxcy[:,0].unsqueeze(1)
    cy=cxcy[:,1].unsqueeze(1)

    # cx,cy,w,h를 합침
    boxes=torch.cat((cx,cy, w_tensor, h_tensor), -1) # cx,cy,w,h

    return torch.cat((boxes[:, :2] - boxes[:, 2:]/2, # xmin, ymin
                      boxes[:, :2] + boxes[:, 2:]/2), 1) # xmax, ymax
```

- cxcy, bb 확인

```
torch.manual_seed(0)

cxcy=torch.rand(1,2)
print("center:", cxcy*256)

bb=cxcy2bbox(cxcy)
print("bounding box", bb*256)
```

```
center: tensor([[127.0417, 196.6648]])
bounding box tensor([[102.0417, 171.6648, 152.0417, 221.664
8]])
```

- metric 함수 정의 & 값 넣어서 확인

https://pytorch.org/docs/stable/generated/torch.diagonal.html

위 함수에서는 output과 target을 각각 bounding box로 변환한 후 iou 값 추출,

```
import torchvision
def metrics_batch(output, target):
    output=cxcy2bbox(output)
    target=cxcy2bbox(target)

    iou=torchvision.ops.box_iou(output, target)
    return torch.diagonal(iou, 0).sum().item()
```

```
n,c=8,2
target = torch.rand(n, c, device=device)
target=cxcy2bbox(target)
metrics_batch(target,target)
```

8.0

- loss batch 함수 정의
loss 값이랑 metric_b 값 출력

```
def loss_batch(loss_func, output, target, opt=None):
    # get loss
    loss = loss_func(output, target)

    # get performance metric
    metric_b = metrics_batch(output,target)

    if opt is not None:
        opt.zero_grad()
        loss.backward()
        opt.step()

    return loss.item(), metric_b
```

- 값을 넣어서 loss_batch 확인

```
for xb,label_b in train_dl:
    # convert to tensor
    label_b=torch.stack(label_b,1)
    label_b=label_b.type(torch.float32)
    label_b=label_b.to(device)

    l,m=loss_batch(loss_func,label_b,label_b)
    print(l,m)
    break
```

0.0 0.8

7. Traning and evaluation of the model

여기에서는 우리 모형을 학습시킨다.
코드의 가독성을 위해 함수를 정의하여 사용한다.

- loss epoch 함수 정의

```
def loss_epoch(model,loss_func,dataset_dl,sanity_check=False,opt=None):
    running_loss=0.0
    running_metric=0.0
    len_data=len(dataset_dl.dataset)

    for xb, yb in dataset_dl:
        # tensor로 변환
        yb=torch.stack(yb,1)
        yb=yb.type(torch.float32).to(device)

        # model에 넣어서 output
        output=model(xb.to(device))

        # batch마다 loss 값 구하기
        loss_b,metric_b=loss_batch(loss_func, output, yb, opt)

        # running loss 업데이트
        running_loss+=loss_b

        # running metric 업데이트
        if metric_b is not None:
            running_metric+=metric_b

    # loss value 평균 구하기
    loss=running_loss/float(len_data)

    # metric value 평균 구하기
    metric=running_metric/float(len_data)

    return loss, metric
```

- train val 함수 정의
위 함수는 parameters를 정의한다.

```
import copy
def train_val(model, params):
    # extract parameters
    num_epochs=params["num_epochs"]
    loss_func=params["loss_func"]
    opt=params["optimizer"]
    train_dl=params["train_dl"]
    val_dl=params["val_dl"]
    sanity_check=params["sanity_check"]
    lr_scheduler=params["lr_scheduler"]
    path2weights=params["path2weights"]

    # history of loss values in each epoch
    loss_history={
        "train": [],
        "val": [],
    }

    # histroy of metric values in each epoch
    metric_history={
        "train": [],
        "val": [],
    }



    # a deep copy of weights for the best performing model
    best_model_wts = copy.deepcopy(model.state_dict())

    # initialize best loss to a large value
    best_loss=float('inf')

    for epoch in range(num_epochs):
        # get current learning rate
        current_lr=get_lr(opt)
        print('Epoch {}/{}, current lr={}'.format(epoch, num_epochs - 1,
current_lr))

        # train the model
        model.train()
        train_loss,
```

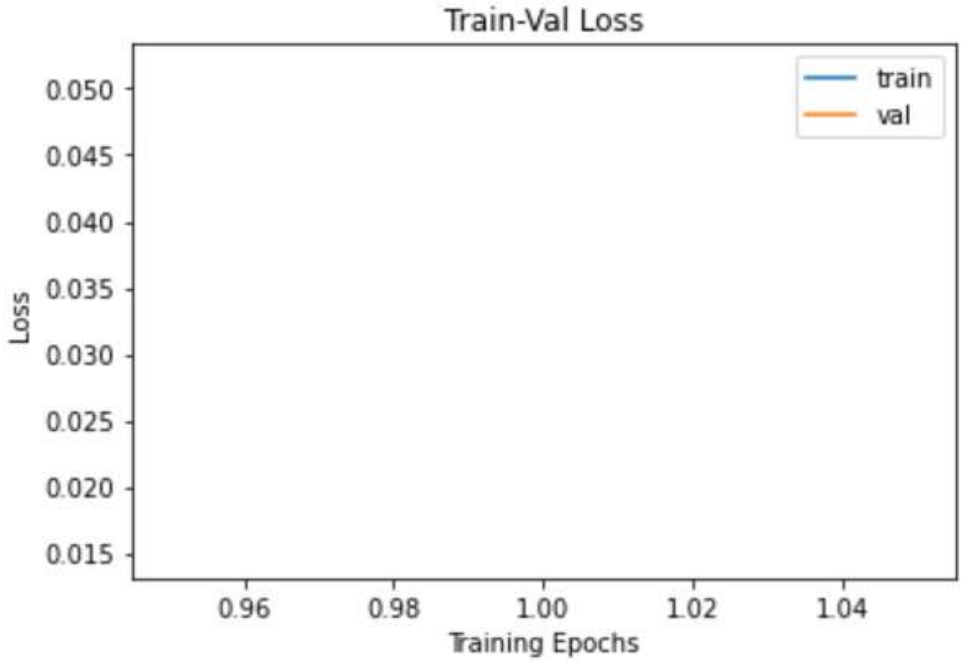
```
train_metric=loss_epoch(model,loss_func,train_dl,sanity_check,opt)

        # collect loss and metric for training dataset
        loss_history["train"].append(train_loss)
        metric_history["train"].append(train_metric)

        # evaluate the model
        model.eval()
        with torch.no_grad():
            val_loss, val_metric=loss_epoch(model,loss_func,val_dl,sanity_check)

        # collect loss and metric for validation dataset
        loss_history["val"].append(val_loss)
        metric_history["val"].append(val_metric)



        # best model 저장
        if val_loss < best_loss:
            best_loss = val_loss
            best_model_wts = copy.deepcopy(model.state_dict())

            # store weights into a local file
            torch.save(model.state_dict(), path2weights)
            print("Copied best model weights!")

        # learning rate schedule
        lr_scheduler.step(val_loss)
        if current_lr != get_lr(opt):
            print("Loading best model weights!")
            model.load_state_dict(best_model_wts)



        print("train loss: %.6f, accuracy: %.2f" %(train_loss,100*train_metric))
        print("val loss: %.6f, accuracy: %.2f" %(val_loss,100*val_metric))
        print("-"*10)



    # load best model weights
    model.load_state_dict(best_model_wts)
    return model, loss_history, metric_history
```

```
loss_func=nn.SmoothL1Loss(reduction="sum")
opt = optim.Adam(model.parameters(), lr=1e-4)
lr_scheduler = ReduceLROnPlateau(opt, mode='min',factor=0.5, patience=20,verbose=1)

path2models= "./models/"
if not os.path.exists(path2models):
        os.mkdir(path2models)

params_train={
    "num_epochs": 1,
    "optimizer": opt,
    "loss_func": loss_func,
    "train_dl": train_dl,
    "val_dl": val_dl,
    "sanity_check": False,
    "lr_scheduler": lr_scheduler,
    "path2weights": path2models+"weights_smoothl1.pt",
}

# train and validate the model
model,loss_hist,metric_hist=train_val(model,params_train)
```

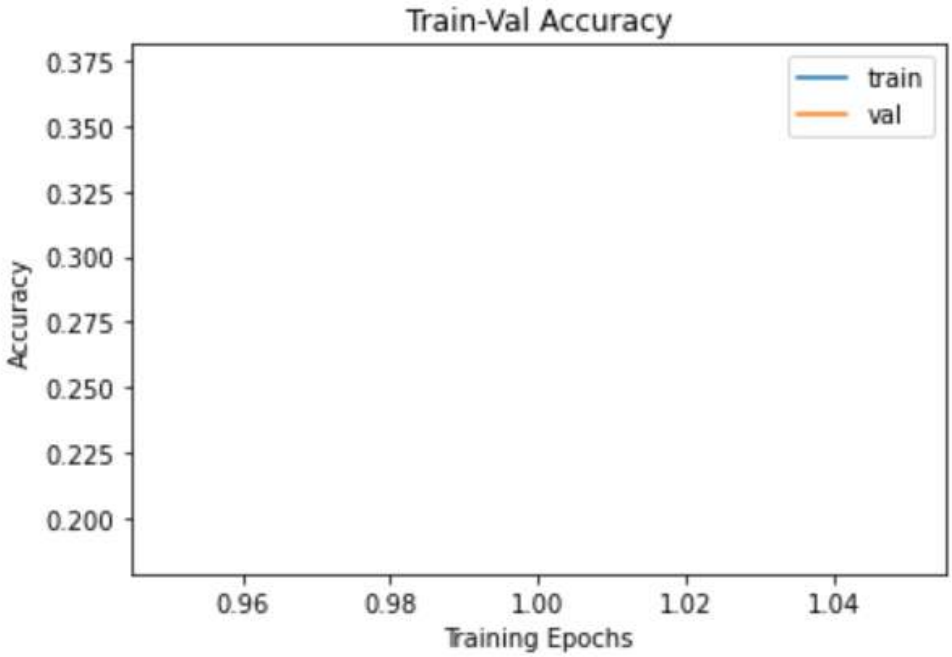
```
Epoch 0/0, current lr=0.0001
C:₩Users₩USER₩anaconda3₩lib₩site-packages₩torch₩nn₩functional.py:718: UserWarning: Named tensors and all their associated APIs are an experimental feature and subject to change. Please do not use them for anything important until they are released as stable. (Triggered internally at  ..₩c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
Copied best model weights!
train loss: 0.051518, accuracy: 18.78
val loss: 0.014951, accuracy: 37.21
----------
```

```
# Train-Validation Progress
num_epochs=params_train["num_epochs"]
```

```
# plot loss progress
plt.title("Train-Val Loss")
plt.plot(range(1,num_epochs+1),loss_hist["train"],label="train")
plt.plot(range(1,num_epochs+1),loss_hist["val"],label="val")
plt.ylabel("Loss")
plt.xlabel("Training Epochs")
plt.legend()
plt.show()
```




```
# plot accuracy progress
plt.title("Train-Val Accuracy")
plt.plot(range(1,num_epochs+1),metric_hist["train"],label="train")
plt.plot(range(1,num_epochs+1),metric_hist["val"],label="val")
plt.ylabel("Accuracy")
plt.xlabel("Training Epochs")
plt.legend()
plt.show()
```



<Traning and evaluation of the model Summary>

- epoch마다 loss와 IOU metric를 구하는 함수 정의

: 이 함수는 train 데이터와 validation 데이터에 모두 사용하는데, 이때 validation 데이터는 opt=None이 함수에 전달되면 최적화가 수행되지 않는다. data의 batches는 data loader 안의 내부로부터 얻을 수 있다. 이때 label 같은 경우 앞에서 언급한 것 처럼 stack 함수를 이용해 tensor로 변환하여 사용한다. 그리고 모형의 output은 loss_batch 함수를 이용해 mini-batch마다 loss 값과 IOU 값을 계산하여 얻을 수 있다.

- train_val 함수 정의

params로 표현하여 정의