**IVPG 과제2 보고서**

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### **전자정보통신공학과**

19010683

장성원

2025.03.24

## 

## **목차**

I. 서론

1. 문제 정의

a. one-stage Detector with fully connected layer

II. 본론

1. 문제 풀이

a. 풀이 과정

III. 결론

1. 손실 그래프

2. 전체 코드

**I. 서론**

1. 문제 정의

1. **one-stage Detector with fully connected layer**

1) 한 영상에 **하나의 사각형이 존재하는 영상**을 학습 영상으로 준비하세요.

- 사각형의 크기와 위치는 강의자료와 동일한 범위에서 무작위로 선정하세요.

- 사각형의 색상은 4가지 중에서 무작위로 설정하세요.

red(0,0,255), green(0,255,0), blue(255,0,0), yellow(0,255,255)

- Label 값은 red는 0, green은 1, blue는 2, yellow는 3으로 설정하세요.

2) 영상에 존재하는 사각형의 위치, 크기, 색상을 인식하는 검출기를 만드세요.

- VGG16을 backbone으로 사용하세요.

- Fully connected layer를 사용하여 위치, 크기, 색상을 알아내세요.

- Separate head와 custom loss function을 사용하세요.

- 위치와 크기에 대한 loss는 MSE를 사용하고, 색상 class에 대한 loss는 cross entropy를 사용하세요.

\* 검출기의 구조를 model.summary()를 통해 그려서 보고서에 넣으세요.

3) 위 검출기를 model.fit을 통해 학습하세요.

- Optimizer, learning\_rate, loss\_weight, epochs, batch\_size, validation\_split 등의 hyperparameter들은 적절하게 설정하세요.

4) 평가 영상을 생성하고 학습된 검출기에 적용하여 검출 결과를 확인하세요.

- 위치 추정 결과는 검은색 점으로 표시하세요.

cv.circle(test\_img[n],center=(x,y),radius=2,color=(0,0,0),thickness=2)

- 크기 추정 결과는 검은색 사각형 모양의 선으로 표시하세요.

cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)),color=(0,0,0),thickness=1)

- 색상 인식 결과는 영상 테두리의 색상을 인식된 색상으로 표시하세요.

cv.rectangle(test\_img[n],(0,0),(W-1,H-1),color=(0,0,255),thickness=1)

**II. 본론**

1. 문제 풀이
2. 풀이 과정

어떤 영상에 **red, green, blue, yellow의 4가지 색상 중 하나의 색을 가진 하나의 객체**가 들어올 때, 해당 객체의 위치, 크기, 어떤 색상인지를 모두 알아내는 문제를 해결해야 한다. 따라서 해당 객체의 위치에 대한 정보 (x,y)를 train\_label\_coord에, 크기에 대한 정보 (W,H)를 train\_label\_size에, 어떤 색상인지에 대한 정보 (r,g,b,y)를 train\_label\_class에 저장하도록 하였다. 영상 이미지는 4000장으로 설정 하였으며, 각 영상의 가로, 세로 길이는 각각 256, 256으로 설정하였다. 이에 대한 코드는 다음과 같다. 이때 train\_label\_class의 경우 [0,0,1,0] 과 같은 원 핫 인코딩 형식이 아닌, **각 색상마다 0, 1, 2, 3 중 하나의 값을 대입**해야 하므로 np.zeros([N,1], dtype=np.int32)의 N\*1 행렬을 선언해야 한다.

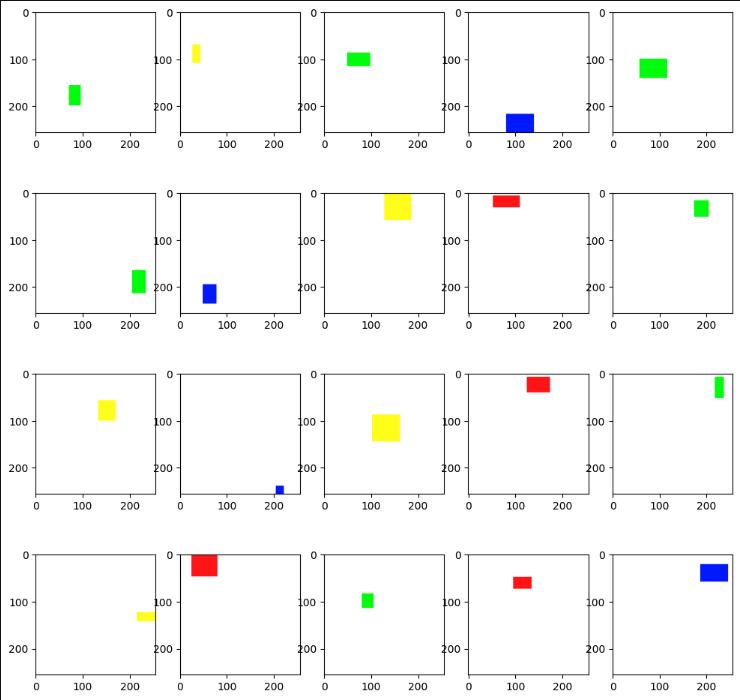
| # Generate training images and labels N=4000 H,W=256,256  train\_img=np.zeros([N,H,W,3],dtype=np.uint8) train\_img.fill(255)  train\_label\_coord=np.zeros([N,2],dtype=np.int32) train\_label\_size=np.zeros([N,2],dtype=np.int32) train\_label\_class=np.zeros([N,1],dtype=np.int32) |
| --- |

영상마다 4가지 색상 중 무작위로 하나를 선택해서 무작위의 크기를 가지는 객체를 생성하면서, 각 색상에 대해 0, 1, 2, 3의 라벨 값을 설정해야 한다. 이에 대한 코드는 다음과 같다.

| for n in range(N):  x,y=random.randint(0,W-1),random.randint(0,H-1)  bw,bh=random.randint(int(W/16),int(W/4)),random.randint(int(H/16),int(H/4))    if(x-bw/2<0): x=x-(x-bw/2)  elif(x+bw/2>W-1): x=x-(x+bw/2-(W-1))  if(y-bh/2<0): y=y-(y-bh/2)  elif(y+bh/2>H-1): y=y-(y+bh/2-(H-1))    x=int(x); y=int(y)    train\_label\_coord[n,0]=x; train\_label\_coord[n,1]=y  train\_label\_size[n,0]=bw; train\_label\_size[n,1]=bh  train\_label\_class[n]=random.randint(0,3) # 0: red, 1: green, 2: blue, 3: yellow    if train\_label\_class[n]==0: # red  cv.rectangle(train\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,0,255), thickness=-1)  elif train\_label\_class[n]==1: # green  cv.rectangle(train\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,255,0), thickness=-1)  elif train\_label\_class[n]==2: # blue  cv.rectangle(train\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(255,0,0), thickness=-1)  else: # yellow  cv.rectangle(train\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(255,255,0), thickness=-1) |
| --- |

아래의 코드를 사용하면 한 영상에 무작위의 색상, 크기를 가진 하나의 사각형이 잘 만들어진 것을 확인할 수 있다.

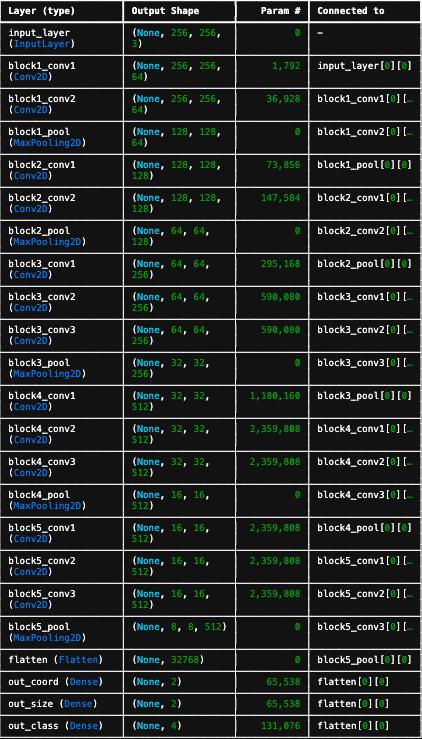
| # Display some images  fig = plt.figure(figsize=(12,12)) rows=4 columns=5  for idx in range(1, columns \* rows +1):  fig.add\_subplot(rows, columns, idx)  plt.imshow(train\_img[idx]) |
| --- |



다음으로 VGG16 넷을 backbone으로 사용하고, Fully Connected Layer을 사용하여 위치, 크기, 색상을 알아내야 한다. 이 문제는 위치에 대해서 (x, y), 크기에 대해서 (W, H), 색상에 대해서 (r,g,b,g)에 대한 **각각의 확률 값을 구해내는 문제**라고 볼 수 있다. 따라서 VGG16 넷의 특성 추출 레이어만 뽑아내고 Flatten한 다음, **Seperate Head 기법을 사용하여 각각 2, 2, 4개의 뉴런을 가진 3개의 Dense 레이어를 만들면 된다**. 위치, 크기 추출 레이어에 대해서는 2개의 변수에 대한 확률 값을 구해내는 문제이기 때문에 **활성화 함수로 sigmoid**를, 색상 추출 레이어에 대해서는 4개의 변수에 대한 확률 값을 구해내는 문제이기 때문에 **활성화 함수로 softmax** 함수를 사용한다. 이에 대한 코드는 다음과 같다.

| # Preprocess data train\_img = tf.keras.applications.vgg16.preprocess\_input(train\_img) train\_label\_coord=train\_label\_coord.astype(np.float32) train\_label\_coord[:,0]=train\_label\_coord[:,0]/W train\_label\_coord[:,1]=train\_label\_coord[:,1]/H train\_label\_size=train\_label\_size.astype(np.float32) train\_label\_size[:,0]=train\_label\_size[:,0]/(W/4) train\_label\_size[:,1]=train\_label\_size[:,1]/(H/4)  # Build model base\_model = tf.keras.applications.VGG16(input\_shape=[H,W,3], include\_top=False, weights='imagenet') x = base\_model.output x = tf.keras.layers.Flatten()(x)  out\_coord=tf.keras.layers.Dense(2,activation='sigmoid',name='out\_coord')(x) out\_size=tf.keras.layers.Dense(2,activation='sigmoid',name='out\_size')(x) out\_class=tf.keras.layers.Dense(4,activation='softmax',name='out\_class')(x) # 클래스가 4개  model=tf.keras.Model(inputs=base\_model.input,outputs=[out\_coord,out\_size,out\_class]) model.summary() |
| --- |

아래는 model.summary() 결과로 나온 표이다. **flatten 레이어에서 나온 출력이 각각 out\_coord, out\_size, out\_class 레이어로 잘 들어가는 것**을 확인할 수 있다.



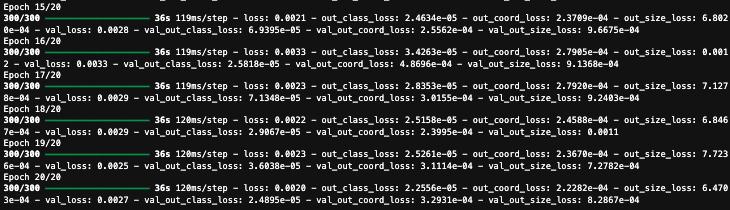
다음으로 custom loss function을 사용하여 out\_coord, out\_size, out\_class 레이어에 대해 손실 함수를 잘 지정해 주어야 한다. 세 레이어에 대한 손실 함수를 각각 coord\_loss\_func, size\_loss\_func, class\_loss\_func로 지정하였다. **위치와 크기에 대한 loss로는 MSE를 사용**했다. class\_loss\_func의 경우가 중요한데, 처음에 **train\_label\_class를 np.zeros([N,1],dtype=np.int32) 로 0~3 사이의 N\*1의 행렬로 지정한데에 반해 out\_class의 출력 값은 (batch\_size, 4)로 []의 확률 값을 가진 N\*4 행렬의 형태를 띄기 때문**에 서로 다른 성격을 가진다는 문제가 생긴다. 따라서 **train\_label\_class에 대해 원 핫 인코딩을 적용할 필요**가 있다. 4개의 클래스로 분류해야하는 문제이기 때문에 손실 함수로는 **categorical\_crossentropy를 사용하였다**. 이에 대한 코드는 다음과 같다.

| # Custom loss functions def coord\_loss\_func(y\_true,y\_pred):  loss\_coord=tf.keras.losses.MSE(y\_true,y\_pred)  return loss\_coord  def size\_loss\_func(y\_true,y\_pred):  loss\_size=tf.keras.losses.MSE(y\_true,y\_pred)  return loss\_size   def class\_loss\_func(y\_true,y\_pred):   y\_true = tf.cast(y\_true, tf.int32)  y\_true = tf.one\_hot(y\_true, 4)  y\_true = tf.reshape(y\_true, [-1,4])   # One Hot Encoding을 한 경우 e.g. [0,0,1,0] : categorical\_crossentropy, 그렇지 않은 경우 (e.g. [3,0,1] : spare\_categorical\_crossentropy   loss\_class=tf.keras.losses.categorical\_crossentropy(y\_true,y\_pred)  return loss\_class |
| --- |

학습 진행을 위한 파라미터로는 optimizer은 Adam, learning rate는 10^-4, loss\_weight는 out\_coord, out\_size, out\_class에 대해 각각 3, 2, 2로, epoch는 20, batch\_size는 10, validation\_split은 0.25로 설정하여 학습을 진행하였다. 이에 대한 코드는 다음과 같다.

| # Train and save model model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0001),  loss={'out\_coord':coord\_loss\_func,'out\_size':size\_loss\_func,'out\_class':class\_loss\_func},  loss\_weights={'out\_coord':3,'out\_size':2,'out\_class':2})  history = model.fit(x=train\_img,  y={'out\_coord':train\_label\_coord,'out\_size':train\_label\_size,'out\_class':train\_label\_class},  epochs=20,batch\_size=10,validation\_split=0.25) model.save('model.keras') |
| --- |

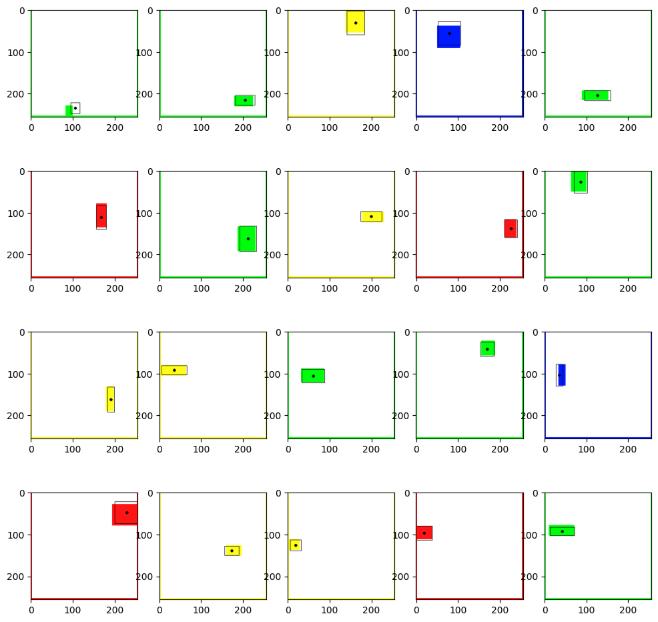
Epoch가 지남에 따라 training loss는 잘 떨어지고, validation loss는 잘 떨어지다가 어느정도 수렴하는 것을 확인할 수 있었다.



마지막으로 테스트 영상을 새롭게 생성하여 테스트 영상에 대해서 모델이 테스트 영상의 위치, 크기, 라벨 정보를 잘 추론하는지 확인해 보았다. 위치 추정 결과는 검은색 점으로, 크기 추정 결과는 검은색 사각형 모양의 선으로, 색상 인식 결과는 영상 테두리의 색상을 인식된 색상으로 표시해 넣었다. 이에 대한 코드는 다음과 같다.

| # Generate test images and labels N=20 H,W=256,256  test\_img=np.zeros([N,H,W,3],dtype=np.uint8) test\_img.fill(255) test\_label\_coord=np.zeros([N,2],dtype=np.int32) test\_label\_size=np.zeros([N,2],dtype=np.int32) test\_label\_class=np.zeros([N,1],dtype=np.int32) # 출력 라벨 클래스가 4개  for n in range(N):  x,y=random.randint(0,W-1),random.randint(0,H-1)  bw,bh=random.randint(int(W/16),int(W/4)),random.randint(int(H/16),int(H/4))    if(x-bw/2<0): x=x-(x-bw/2)  elif(x+bw/2>W-1): x=x-(x+bw/2-(W-1))  if(y-bh/2<0): y=y-(y-bh/2)  elif(y+bh/2>H-1): y=y-(y+bh/2-(H-1))    x=int(x); y=int(y)    test\_label\_coord[n,0]=x; test\_label\_coord[n,1]=y  test\_label\_size[n,0]=bw; test\_label\_size[n,1]=bh  test\_label\_class[n]=random.randint(0,3) # 0: red, 1: green, 2: blue, 3: yellow    if test\_label\_class[n]==0: # red  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,0,255), thickness=-1)  elif test\_label\_class[n]==1: # green  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,255,0), thickness=-1)  elif test\_label\_class[n]==2: # blue  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(255,0,0), thickness=-1)  else: # yellow  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(255,255,0), thickness=-1)  # Preprocess test images test\_img\_ = tf.keras.applications.vgg16.preprocess\_input(test\_img) # Predict object locations in test images model=tf.keras.models.load\_model('model.keras', custom\_objects={'coord\_loss\_func':coord\_loss\_func,  'size\_loss\_func':size\_loss\_func, 'class\_loss\_func':class\_loss\_func})  pred\_coord,pred\_size,pred\_class=model.predict(test\_img\_) pred\_coord[:,0]=pred\_coord[:,0]\*H pred\_coord[:,1]=pred\_coord[:,1]\*W pred\_size[:,0]=pred\_size[:,0]\*(H/4) pred\_size[:,1]=pred\_size[:,1]\*(W/4) pred\_class=np.argmax(pred\_class,axis=1)  # Display prediction results  fig = plt.figure(figsize=(12, 12)) rows = 4 columns=5  for n in range(N):  x=pred\_coord[n,0].astype('int')  y=pred\_coord[n,1].astype('int')  bw=pred\_size[n,0].astype('int')  bh=pred\_size[n,1].astype('int')    obj\_class=pred\_class[n]    if obj\_class==0: # red  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)),color=(0,0,0),thickness=1)  cv.rectangle(test\_img[n],(0,0),(W-1,H-1),color=(0,0,255),thickness=4)  elif obj\_class==1: # green  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,0,0),thickness=1)  cv.rectangle(test\_img[n],(0,0),(W-1,H-1),color=(0,255,0),thickness=4)  elif obj\_class==2: # blue  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)),color=(0,0,0),thickness=1)  cv.rectangle(test\_img[n],(0,0),(W-1,H-1),color=(255,0,0),thickness=4)  else : # yellow  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)),color=(0,0,0),thickness=1)  cv.rectangle(test\_img[n],(0,0),(W-1,H-1),color=(255,255,0),thickness=4)    cv.circle(test\_img[n],center=(x,y),radius=2,color=(0,0,0),thickness=2)  fig.add\_subplot(rows, columns, n+1)  plt.imshow(test\_img[n]) |
| --- |

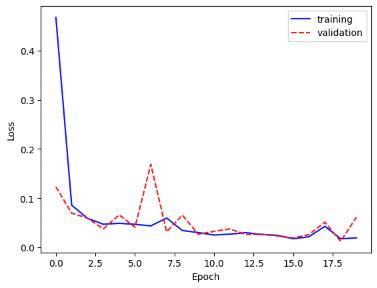
아래는 테스트 데이터 셋에 대해 학습된 모델을 적용한 결과이다. 크기와 위치, 색상을 잘 추론해 내는 것을 확인할 수 있었다.



**III. 결론**

1. 손실 그래프

아래 그림은 out\_coord, out\_size, out\_class의 출력에 각각 다른 가중치를 두어 계산된 train, validation dataset의 총 손실에 대한 epoch-loss 그래프이다. train dataset과 validation dataset에 대해 epoch가 지날수록 loss가 잘 떨어지는 것을 보아 어느정도 학습이 잘 진행되고 있음을 확인할 수 있다.



1. 전체 코드

| # Import libraries  import numpy as np import tensorflow as tf import cv2 as cv import random import matplotlib.pyplot as plt  # Generate training images and labels N=4000 H,W=256,256  train\_img=np.zeros([N,H,W,3],dtype=np.uint8) train\_img.fill(255)  train\_label\_coord=np.zeros([N,2],dtype=np.int32) train\_label\_size=np.zeros([N,2],dtype=np.int32) train\_label\_class=np.zeros([N,1],dtype=np.int32)  for n in range(N):  x,y=random.randint(0,W-1),random.randint(0,H-1)  bw,bh=random.randint(int(W/16),int(W/4)),random.randint(int(H/16),int(H/4))    if(x-bw/2<0): x=x-(x-bw/2)  elif(x+bw/2>W-1): x=x-(x+bw/2-(W-1))  if(y-bh/2<0): y=y-(y-bh/2)  elif(y+bh/2>H-1): y=y-(y+bh/2-(H-1))    x=int(x); y=int(y)    train\_label\_coord[n,0]=x; train\_label\_coord[n,1]=y  train\_label\_size[n,0]=bw; train\_label\_size[n,1]=bh  train\_label\_class[n]=random.randint(0,3) # 0: red, 1: green, 2: blue, 3: yellow    if train\_label\_class[n]==0: # red  cv.rectangle(train\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,0,255), thickness=-1)  elif train\_label\_class[n]==1: # green  cv.rectangle(train\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,255,0), thickness=-1)  elif train\_label\_class[n]==2: # blue  cv.rectangle(train\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(255,0,0), thickness=-1)  else: # yellow  cv.rectangle(train\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(255,255,0), thickness=-1)  # Display some images  fig = plt.figure(figsize=(12,12)) rows=4 columns=5  for idx in range(1, columns \* rows +1):  fig.add\_subplot(rows, columns, idx)  plt.imshow(train\_img[idx])  # Preprocess data train\_img = tf.keras.applications.vgg16.preprocess\_input(train\_img) train\_label\_coord=train\_label\_coord.astype(np.float32) train\_label\_coord[:,0]=train\_label\_coord[:,0]/W train\_label\_coord[:,1]=train\_label\_coord[:,1]/H train\_label\_size=train\_label\_size.astype(np.float32) train\_label\_size[:,0]=train\_label\_size[:,0]/(W/4) train\_label\_size[:,1]=train\_label\_size[:,1]/(H/4)  # Build model base\_model = tf.keras.applications.VGG16(input\_shape=[H,W,3], include\_top=False, weights='imagenet') x = base\_model.output x = tf.keras.layers.Flatten()(x)  out\_coord=tf.keras.layers.Dense(2,activation='sigmoid',name='out\_coord')(x) out\_size=tf.keras.layers.Dense(2, activation='sigmoid',name='out\_size')(x) out\_class=tf.keras.layers.Dense(4,activation='softmax',name='out\_class')(x) # 클래스가 4개  model=tf.keras.Model(inputs=base\_model.input,outputs=[out\_coord,out\_size,out\_class]) model.summary()  # Custom loss functions def coord\_loss\_func(y\_true,y\_pred):  loss\_coord=tf.keras.losses.MSE(y\_true,y\_pred)  return loss\_coord  def size\_loss\_func(y\_true,y\_pred):  loss\_size=tf.keras.losses.MSE(y\_true,y\_pred)  return loss\_size   def class\_loss\_func(y\_true,y\_pred):   y\_true = tf.cast(y\_true, tf.int32)  y\_true = tf.one\_hot(y\_true, 4)  y\_true = tf.reshape(y\_true, [-1,4])   # One Hot Encoding을 한 경우 e.g. [0,0,1,0] : categorical\_crossentropy, 그렇지 않은 경우 (e.g. [3,0,1] : spare\_categorical\_crossentropy   loss\_class=tf.keras.losses.categorical\_crossentropy(y\_true,y\_pred)  return loss\_class  # Train and save model model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0001),  loss={'out\_coord':coord\_loss\_func,'out\_size':size\_loss\_func,'out\_class':class\_loss\_func},  loss\_weights={'out\_coord':3,'out\_size':2,'out\_class':2})  history = model.fit(x=train\_img,  y={'out\_coord':train\_label\_coord,'out\_size':train\_label\_size,'out\_class':train\_label\_class},  epochs=20,batch\_size=10,validation\_split=0.25) model.save('model.keras')  # Show training history plt.figure() plt.plot(history.history['loss'], 'b-', label='training') plt.plot(history.history['val\_loss'], 'r--', label='validation') plt.xlabel('Epoch'), plt.ylabel('Loss'), plt.legend() plt.show()  # Generate test images and labels N=20 H,W=256,256  test\_img=np.zeros([N,H,W,3],dtype=np.uint8) test\_img.fill(255) test\_label\_coord=np.zeros([N,2],dtype=np.int32) test\_label\_size=np.zeros([N,2],dtype=np.int32) test\_label\_class=np.zeros([N,1],dtype=np.int32) # 출력 라벨 클래스가 4개  for n in range(N):  x,y=random.randint(0,W-1),random.randint(0,H-1)  bw,bh=random.randint(int(W/16),int(W/4)),random.randint(int(H/16),int(H/4))    if(x-bw/2<0): x=x-(x-bw/2)  elif(x+bw/2>W-1): x=x-(x+bw/2-(W-1))  if(y-bh/2<0): y=y-(y-bh/2)  elif(y+bh/2>H-1): y=y-(y+bh/2-(H-1))    x=int(x); y=int(y)    test\_label\_coord[n,0]=x; test\_label\_coord[n,1]=y  test\_label\_size[n,0]=bw; test\_label\_size[n,1]=bh  test\_label\_class[n]=random.randint(0,3) # 0: red, 1: green, 2: blue, 3: yellow    if test\_label\_class[n]==0: # red  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,0,255), thickness=-1)  elif test\_label\_class[n]==1: # green  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,255,0), thickness=-1)  elif test\_label\_class[n]==2: # blue  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(255,0,0), thickness=-1)  else: # yellow  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(255,255,0), thickness=-1)  # Preprocess test images test\_img\_ = tf.keras.applications.vgg16.preprocess\_input(test\_img) # Predict object locations in test images model=tf.keras.models.load\_model('model.keras', custom\_objects={'coord\_loss\_func':coord\_loss\_func,  'size\_loss\_func':size\_loss\_func, 'class\_loss\_func':class\_loss\_func})  pred\_coord,pred\_size,pred\_class=model.predict(test\_img\_) pred\_coord[:,0]=pred\_coord[:,0]\*H pred\_coord[:,1]=pred\_coord[:,1]\*W pred\_size[:,0]=pred\_size[:,0]\*(H/4) pred\_size[:,1]=pred\_size[:,1]\*(W/4) pred\_class=np.argmax(pred\_class,axis=1)  # Display prediction results  fig = plt.figure(figsize=(12, 12)) rows = 4 columns=5  for n in range(N):  x=pred\_coord[n,0].astype('int')  y=pred\_coord[n,1].astype('int')  bw=pred\_size[n,0].astype('int')  bh=pred\_size[n,1].astype('int')    obj\_class=pred\_class[n]    if obj\_class==0: # red  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)),color=(0,0,0),thickness=1)  cv.rectangle(test\_img[n],(0,0),(W-1,H-1),color=(0,0,255),thickness=4)  elif obj\_class==1: # green  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)), color=(0,0,0),thickness=1)  cv.rectangle(test\_img[n],(0,0),(W-1,H-1),color=(0,255,0),thickness=4)  elif obj\_class==2: # blue  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)),color=(0,0,0),thickness=1)  cv.rectangle(test\_img[n],(0,0),(W-1,H-1),color=(255,0,0),thickness=4)  else : # yellow  cv.rectangle(test\_img[n],(x-int(bw/2),y-int(bh/2)),(x+int(bw/2),y+int(bh/2)),color=(0,0,0),thickness=1)  cv.rectangle(test\_img[n],(0,0),(W-1,H-1),color=(255,255,0),thickness=4)    cv.circle(test\_img[n],center=(x,y),radius=2,color=(0,0,0),thickness=2)  fig.add\_subplot(rows, columns, n+1)  plt.imshow(test\_img[n]) |
| --- |