**REPORT**

|  |  |
| --- | --- |
| **과목명**  **교수님**  **제출기한**  **팀원** | **인공지능**  **김동근 교수님**  **2020 – 12 - 06**  **201701786정수빈 201701794최진영 201703009조재훈 201801759김채원 201801781여성일** |

**인공지능 기말 팀 프로젝트**

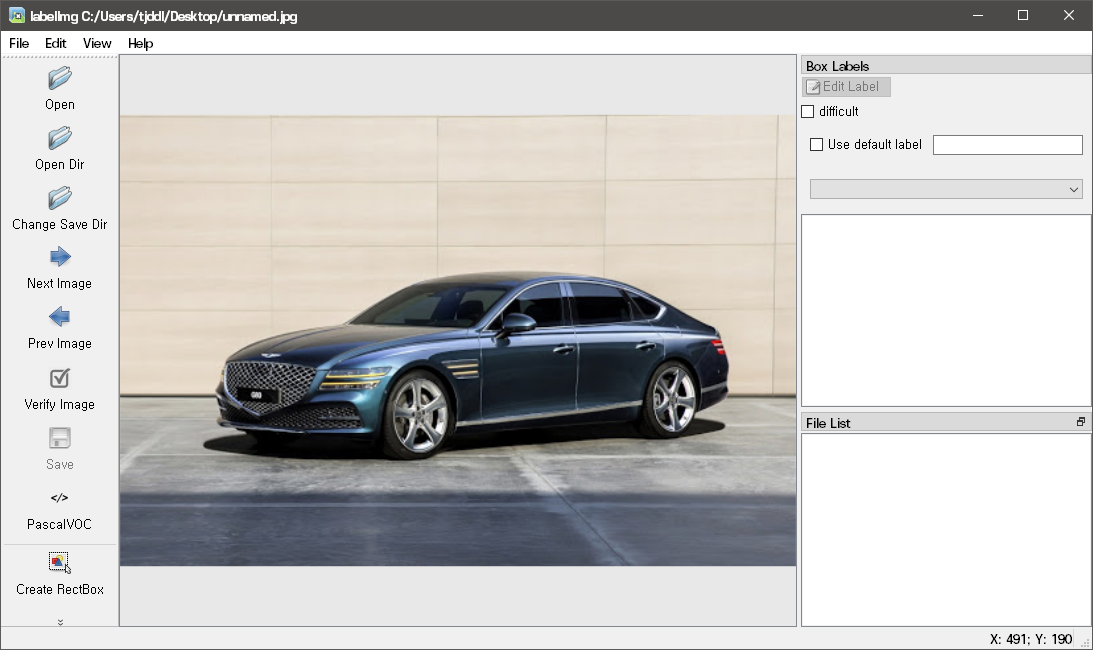
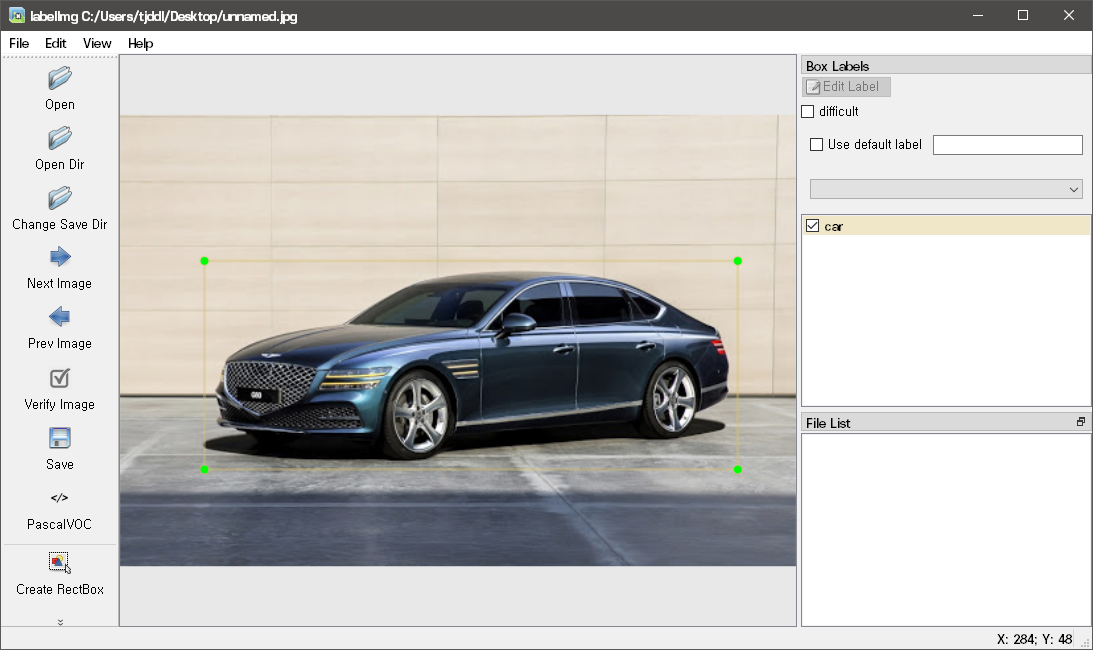


1. **서론**
   1. 문제 설명

저희 6조는 ‘승용차 검출(Car Detection)’을 주제로 텐서플로우(Tensorflow)를 이용한 딥러닝(DeepLearning)으로 학습을 하고 구현했습니다. 학습을 위한 데이터를 만들기 위해 라벨링 툴(labelImg)을 이용하였고, 승용차에 바운딩 박스를 만들어 승용차(car)로 라벨링 작업을 하였습니다. 라벨링 작업을 마친 후 파이썬 코드로 학습 후에, 테스트 데이터를 사용하여 학습이 잘 되었는지 결과를 확인하고, 그래프를 통해 오차(loss)와 IOU수치, custom\_acc(클래스 분류 정확도)를 통한 결과분석을 진행했습니다.

학습용 데이터는 총 900장 수집하였습니다. 정확한 학습을 위해 여러 각도와 여러 방향의 데이터를 수집하였습니다. 데이터 수집 방법은 팀원들이 각자 구글이나 네이버 등과 같은 포털사이트에서 직접 사진을 모으거나, 웹 크롤링 방법으로 사진을 추출하는 방법을 사용했습니다. 테스트 데이터는 학습용 데이터와 마찬가지로 정확한 테스트를 위해 장거리 사진, 근접 사진, 네 방향(앞, 뒤, 양 옆) 사진, 다른 사물과 동시에 나온 사진 등을 팀원들이 10장 내외로 직접 촬영하여 사용하였습니다.

데이터 라벨링은 입력할 학습 데이터가 많기 때문에 팀원들끼리 나눠서 각자 라벨링 툴(labelImg)으로 라벨링을 진행하였습니다. labelImg는 이미지 프로세싱 기본 학습 데이터를 만들 수 있는 툴이고, 데이터 라벨링에 필요한 파일(labelImg, python, pyqt5, lxml, Qt리소스 컴파일)을 설치한 후에 수집한 학습 데이터 파일을 라벨링하여 xml파일로 검출했습니다.

데이터 학습을 위한 파이썬 코드는 수업교재의 58\_02.py예제 파일을 참고하였습니다. 입력 데이터와 출력 부분, 학습 데이터를 학습할 때의 하이퍼 파라미터(hyper parameter)를 변경하여 최적으로 학습하고 영상을 검출하는 코드를 파이썬으로 작성했습니다.

1. **본 론**
   1. 코드 분석

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing import image

import os

import xml.etree.ElementTree as ET

import cv2

import tensorflow as tf

from tensorflow.keras.layers   import Input, Dense, Flatten

from tensorflow.keras.layers   import Conv2D, BatchNormalization,MaxPool2D, Dropout, MaxPooling2D

from tensorflow.keras.layers   import LeakyReLU

from tensorflow.keras.optimizers import Adam, RMSprop, SGD

from tensorflow.keras.applications.vgg16 import preprocess\_input, VGG16

CNN을 구동시키는데 필요한 모듈들입니다. numpy, matplotlib, os, xml.etree.ElementTree, cv2를 import 해주고 keras 라이브러리에서 import 해줍니다.

def getBB(file\_path):

  try:

    tree = ET.parse(file\_path)

  except FileNotFoundError:

    return None

  root = tree.getroot()

  ob = root.find('object')

  bndbox = ob.find('bndbox')

  xmin = bndbox.find('xmin').text

  xmax = bndbox.find('xmax').text

  ymin = bndbox.find('ymin').text

  ymax = bndbox.find('ymax').text

  return [int(xmin), int(ymin), int(xmax), int(ymax)]

getBB 함수는 file path 경로를 통해 XML 파일들을 가져와 우리가 labelImg를 이용하여 train data로 쓸 사진들에 해준 바운딩 박스를 찾아 박스에 대한 정보들을 int(xmin), int(ymin), int(xmax), int(ymax)로 반환해주는 함수입니다. ET.parse를 통해서 파일들을 파싱해주고 XML트리 구조에서 root의 자식구조인 object를 찾아줍니다. 또 object에서 bndbox를 찾아 그 밑에 있는 xmin, xmax, ymin, ymax를 가져온 후 모드 int형으로 반환해줍니다.

def load\_car(target\_size= (224, 224)):

    input\_file = "/content/gdrive/MyDrive/annotations/list.txt"

    file = open(input\_file)

    list\_txt = file.readlines()

    file.close()

    list\_txt = list\_txt[1:]

    np.random.shuffle(list\_txt)

    train\_dataset= {"name": [], "label": [], "image": [ ], "box": [] }

    test\_dataset = {"name": [], "label": [], "image": [ ]}

    for line in list\_txt:

        image\_name, class\_id, species, breed\_id = line.split()

       image\_file= "/content/gdrive/MyDrive/images/"+ image\_name + ".jpg"

        box\_file  = "/content/gdrive/MyDrive/annotations/xmls/"+ image\_name + ".xml"

        if not os.path.exists(image\_file):

            continue

        img = image.load\_img(image\_file)

        sx = target\_size[0]/img.width

        sy = target\_size[1]/img.height

        img = img.resize(size=target\_size)

        img = image.img\_to\_array(img)

        if  os.path.exists(box\_file):

            box = getBB(box\_file)

            box[0] = round(box[0]\*sx)

            box[1] = round(box[1]\*sy)

            box[2] = round(box[2]\*sx)

            box[3] = round(box[3]\*sy)

            train\_dataset["box"].append(box)

            train\_dataset["name"].append(image\_name)

            train\_dataset["label"].append(int(species)-1)

            train\_dataset["image"].append(img)

        else:

            test\_dataset["name"].append(image\_name)

            test\_dataset["label"].append(int(species)-1)

            test\_dataset["image"].append(img)

    train\_dataset["image"] = np.array(train\_dataset["image"])

    train\_dataset["box"]  = np.array(train\_dataset["box"])

    train\_dataset["label"] = np.array(train\_dataset["label"]) - 1

    train\_dataset["name"]  = np.array(train\_dataset["name"])

    test\_dataset["image"] = np.array(test\_dataset["image"])

    test\_dataset["label"] = np.array(test\_dataset["label"]) - 1

    test\_dataset["name"]  = np.array(test\_dataset["name"])

    return train\_dataset, test\_dataset

load\_car함수는 "/content/gdrive/MyDrive/annotations/list.txt"의 경로에서 list.txt 파일 안에 있는 데이터들을 각 행마다 읽습니다. list\_txt[1:]은 가장 윗줄을 건너 뛰고 데이터들 부분부터 읽게 해줍니다. 데이터들은  np.random.shuffle(list\_txt)로 랜덤하게 읽어옵니다.

train\_data는 name, label, image, box를 가지고 test\_data는 name, label, image를 갖습니다. train\_data의 box는 target\_size= (224, 224)로 스케일링 했습니다.

그다음 반복문을 통해 공백을 기준으로 image\_name, class\_id, species로 나누어줍니다.

Image\_file은 "/content/gdrive/MyDrive/images/"+ image\_name + ".jpg"

로 생성해주고 box\_file은 “/content/gdrive/MyDrive/annotations/xmls/”+image\_name+”.xml”로 찾아서 생성해줍니다. image\_file에서 PIL 컬러 이미지를 원본 크기로 img에 읽어줍니다. 그리고 설정해뒀던 target\_size 크기로 가로와 세로 확대 비율인 sx와 sy를 계산해줍니다. 이미지를 확대시켜주고 image.img\_to\_array(img)로 img를 넘파이 배열로 변경해줍니다.

만약 os.path.exists(box\_file)이 true 즉 xml파일이 존재한다면 getBB(box\_file)로 box\_file에서 바운딩 박스를 읽고 box, name, label, image를 train\_dataset에 추가해줍니다. xml파일이 존재하지 않는다면 name, label, image를 test\_dataset에 추가해줍니다. 그리고 모든 train\_dataset과 test\_dataset을 np.array() 넘파이 배열로 변경하여 반환해줍니다.

train\_dataset, test\_dataset = load\_car()

print("train\_dataset['image'].shape=", train\_dataset['image'].shape)

print("test\_dataset['image'].shape=", test\_dataset['image'].shape)

x\_train = train\_dataset["image"]/255.0

x\_test  = test\_dataset["image"]/255.0

x\_train과 x\_test를 정규화 시켜줍니다.

def make\_labels\_car(dataset, train\_data=True):

영상(image), 바운딩 박스(box), 고양이/개 레이블(label)을 이용하여 새로운 학습 데이터 레이블을 생성하는 함수입니다.

label = dataset[”label”]

N = dataset[’image’].shape[0]

    label = dataset["label"]

    N = dataset['image'].shape[0]

    if train\_data:

        size = dataset['image'].shape[1]

        box   = dataset["box"]

        x = (box[:,0] + box[:,2])/(2.\*size)

        y = (box[:,1] + box[:,3])/(2.\*size)

        w = (box[:,2] - box[:,0])/size

        h = (box[:,3] - box[:,1])/size

바운딩 박스에 대하여 [0, 1]로 정규화하며 중심점(x, y), 크기(w, h)로 변경합니다.

    else:

        x = np.full(shape = (N,), fill\_value=0.5, dtype="float32")

        y = np.full(shape = (N,), fill\_value=0.5, dtype="float32")

        w = np.ones(shape = (N,), dtype="float32")

        h = np.ones(shape = (N,), dtype="float32")

Oxford-IIIT Pet으로부터 생성한 테스트 데이터에 대하여 바운딩 박스를 생성해주는 부분입니다. 바운딩 박스가 없기 때문입니다. 따라서 영상 전체를 의미하는 중심점(0.5, 0.5), 크기(1, 1)로 바운딩 박스를 생성합니다.

 pc = np.ones(shape = (N,), dtype="float32")

pc는 물체 존재의 신뢰도(confidence)이며 훈련 데이터와 테스트 데이터에서 물체(고양이 또는 개)를 모두 가지고 있으므로 1로 설정해야 합니다. (갖지 않으면 pc=0으로 해야 합니다.)

  C = np.zeros((N, 2))

  C[np.arange(N), label]= 1

C는 분류정보로 원-핫 인코딩으로 생성합니다.

  label\_y = np.zeros(shape=(N, 7), dtype="float32")

    label\_y[:, 0] = pc

    label\_y[:, 1] = x

    label\_y[:, 2] = y

    label\_y[:, 3] = w

    label\_y[:, 4] = h

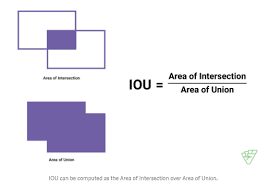
    label\_y[:, 5:]= C

    return label\_y

y\_train = make\_labels\_car(train\_dataset)

y\_test  = make\_labels\_car(test\_dataset, train\_data=False)

새로운 레이블 y\_train과 y\_test를 생성합니다. y\_train.shape=(3671, 7), y\_test.shape=(3678, 7)입니다.



IOU(Intersection Over Union)함수는 2개의 바운딩박스(훈련데이터의 박스, 모델의 예측 출력박스)의 교집합(intersect\_area)을 합집합(Union\_area)영역으로 나누어 계산합니다. 바운딩박스가 완전히 겹치면 1, 분리되어있으면 0입니다.

#4:

def IOU(y\_true, y\_pred):

    b1\_pc, b1\_x, b1\_y, b1\_w, b1\_h, b1\_c1, b1\_c2 = tf.unstack(y\_true, 7, axis=-1)

    b2\_pc, b2\_x, b2\_y, b2\_w, b2\_h, b2\_c1, b2\_c2 = tf.unstack(y\_pred, 7, axis=-1)

#4의 IOU()함수는 y\_true, y\_pred에서 바운딩 박스의 IOU함수를 계산합니다. y\_true와 y\_pred의 7개항목(pc, x, y, w, h, ,c1, c2)에서 (x, y, w, h)를 사용하여 IOU를 계산합니다.

(x, y)는 중심 점이고 (w, h)는 크기입니다.

  zero = tf.convert\_to\_tensor(0.0, y\_true.dtype)

    b1\_width  = tf.maximum(zero, b1\_w)

    b1\_height = tf.maximum(zero, b1\_h)

    b2\_width  = tf.maximum(zero, b2\_w)

    b2\_height = tf.maximum(zero, b2\_h)

zero변수는 0.0로 초기화, data type은 y\_true와 같은 float32입니다.

값이 음수가 될 수 있으므로 zero와 b1\_w(크기의 x좌표값)중에서 큰 값을 b1\_width에 저장합니다. B1\_height, b2\_width, b2\_height도 같은 방법으로 tf.maximum로 저장합니다.

  b1\_w2 = b1\_width/2

    b1\_h2 = b1\_height/2

    b1\_xmin = b1\_x - b1\_w2

    b1\_ymin = b1\_y - b1\_h2

    b1\_xmax = b1\_x + b1\_w2

    b1\_ymax = b1\_y + b1\_h2

b1\_xmin = b1\_x – b1\_width/2를 하면 b1의 바운딩 박스에서 가장 작은 x좌표를 b1\_xmin저장합니다. 같은 방법으로 b1\_ymin은 b1박스의 y좌표 최소값, b1\_xamx은 b1박스의 x좌표 최댓값, b1\_ymax는 b1박스의 y좌표의 최댓값을 저장합니다.

    b2\_w2 = b2\_width/2

    b2\_h2 = b2\_height/2

    b2\_xmin = b2\_x - b2\_w2

    b2\_ymin = b2\_y - b2\_h2

    b2\_xmax = b2\_x + b2\_w2

    b2\_ymax = b2\_y + b2\_h2

같은 방법으로 b2 바운딩박스도 x, y좌표의 최대, 최소값을 변수에 저장합니다.

    b1\_width  = tf.maximum(zero, b1\_xmax - b1\_xmin)

    b1\_height = tf.maximum(zero, b1\_ymax - b1\_ymin)

    b2\_width  = tf.maximum(zero, b2\_xmax - b2\_xmin)

    b2\_height = tf.maximum(zero, b2\_ymax - b2\_ymin)

    b1\_area = b1\_width \* b1\_height

    b2\_area = b2\_width \* b2\_height

b1\_width = tf.maximum(zero, b1\_max – b1.xmin)에는 b1의 폭을 저장합니다. X좌표의 최대값에서 최소값을 뺀 값, 또는 0 중에서 큰값을 저장합니다. 0은 바운딩 박스가 겹치지 않았을 때, 음수 값이 뜨는 경우에 겹치지 않음을 처리하기 위함입니다. 같은 방법으로 b1의 높이(height)와, b2의 폭과 높이를 구한 후에, b1\_area, b2\_area를 width와 height를 곱한 값을 저장합니다.

    intersect\_ymin = tf.maximum(b1\_ymin, b2\_ymin)

    intersect\_xmin = tf.maximum(b1\_xmin, b2\_xmin)

    intersect\_ymax = tf.minimum(b1\_ymax, b2\_ymax)

    intersect\_xmax = tf.minimum(b1\_xmax, b2\_xmax)

intersect\_~ 변수에는 교집합의 범위를 저장합니다. 더 작은 값을 골랐던 xmin과 ymin에서는 더 큰 값(tf.maxmium)을 구하여 저장하고, 더 큰 값을 골랐던 xmax와 ymax에서는 더 작은 값(tf.minimun)을 선택하여 교집합(intersect\_area)의 x좌표와 y좌표의 범위를 정해줍니다.

    intersect\_width = tf.maximum(zero, intersect\_xmax - intersect\_xmin)

    intersect\_height = tf.maximum(zero,intersect\_ymax - intersect\_ymin)

    intersect\_area = intersect\_width \* intersect\_height

교집합(intersecti\_area)의 넓이를 구해줍니다. 최대값에서 최소값을 빼서 width와 height를 구하고 곱하여 initersect\_area를 구합니다.

  union\_area = b1\_area + b2\_area - intersect\_area

    iou = intersect\_area/union\_area

    return iou

합집합(union area)의 넓이는 b1과 b2의 넓에서 중복된 교집합의 넓이를 빼줍니다.

Iou에 교집합/합집합을 저장하고 return iou합니다.

def custom\_loss(y\_true, y\_pred):

    y\_true\_conf, b1\_x, b1\_y, b1\_w, b1\_h, b1\_c1, b1\_c2 = tf.unstack(y\_true, 7, axis=-1)

    y\_pred\_conf, b2\_x, b2\_y, b2\_w, b2\_h, b2\_c1, b2\_c2 = tf.unstack(y\_pred, 7, axis=-1)

    iou = IOU(y\_true, y\_pred)

    loss\_conf = tf.square(y\_true\_conf\*iou- y\_pred\_conf)

    loss\_xy = tf.square(b1\_x - b2\_x) + tf.square(b1\_y - b2\_y)

#5의 custom\_loss()함수는 y\_true, y\_pred의 손실을 YOLO손실함수와 같이 계산합니다.

Loss\_conf는 물체 존재 신뢰도 손실, loss\_xy는 바운딩 박스의 중심점(x, y)의 손실, loss\_wh는 바운딩 박스의 크기(w, h)손실 loss\_classes는 클래스(C1, C2)의 분류 손실입니다.

    b1\_w = tf.sqrt(b1\_w)

    b1\_h = tf.sqrt(b1\_h)

    b2\_w = tf.sqrt(b2\_w)

    b2\_h = tf.sqrt(b2\_h)

    loss\_wh = tf.square(b1\_w -b2\_w) + tf.square(b1\_h - b2\_h)

    epsilon=1e-12

    b2\_c1 = tf.keras.backend.clip(b2\_c1, epsilon, 1.0-epsilon)

    b2\_c2 = tf.keras.backend.clip(b2\_c2,epsilon,1.0-epsilon)

    loss\_class = -(b1\_c1\*tf.math.log(b2\_c1) + b1\_c2\*tf.math.log(b2\_c2))

loss\_xy와 loss\_wh는 오차의 제곱으로 계산하고, loss\_classes는 크로스 엔트로피로 계산합니다.

    loss = loss\_conf + (loss\_xy+ loss\_wh + loss\_class)\*y\_true\_conf

전체 손실 loss는 영상에 물체가 없으면(y\_true\_conf=0) loss=loss\_conf이고, 물체가 있으면 loss=loss\_conf + loss\_xt + loss\_wh + loss\_class입니다.

    loss = tf.reduce\_mean(loss, axis=-1)

    return loss

행방향으로 loss의 평균을 반환(return)합니다.

def custom\_acc(y\_true, y\_pred):

    y\_true\_class = y\_true[...,-2:]

    y\_pred\_class = y\_pred[...,-2:]

    return tf.keras.metrics.categorical\_accuracy(y\_true\_class, y\_pred\_class)

#6의 custom\_acc()함수는 y\_true와 y\_pred에서 y\_true[…,-2:]와 y\_pred[,,,.-2:]를 이용하여 클래스 분류 정확도를 계산합니다.

def custom\_activations(x):

    x\_0 = tf.keras.activations.sigmoid(x[...,:5])

  x\_1 = tf.keras.activations.softmax(x[...,5:])

    new\_x = tf.keras.layers.concatenate([x\_0, x\_1], axis = 1)

    return new\_x

custom\_activations 함수를 정의합니다.

변수 x\_0, 변수 x\_1에 활성함수 sigmoid를 선언한 후에, 변수 new\_x에 x\_0과 x\_1을 axis=-1을 기준으로 concat()합니다. concat()과 stack()의 차이점은 concat()은 데이터의 괄호 rank가 추가되지 않는 반면, stack()은 값을 괄호로 덮어 버립니다. 마지막으로, new\_x값을 리턴 합니다.

def create\_VGG(input\_shape=(224, 224, 3), num\_outs = 7):

    inputs = Input(shape = input\_shape)

    vgg\_model = VGG16(weights = 'imagenet', include\_top = False, input\_tensor = inputs)

    vgg\_model.trainable = False

create\_VGG 함수를 정의합니다.

변수 inputs에 shape를 include\_top 값이 False 이므로 input\_shape로 선언합니다.

변수 vgg\_model에 VGG16모델을 weights 값을 ‘imagenet’(ImageNet에 대한 선행 학습)으로, Input\_tensor(모델의 이미지 인풋으로 사용할 수 있는 선택적 tf.keras) 값을 위에서 정의한 inputs로 선언 합니다.

vgg\_model.trainable값을 False로 선언하여 레이어를 고정시킵니다.

    x = vgg\_model.output

    x = Flatten()(x)

    x = Dense(64)(x)

    x = LeakyReLU()(x)

    x = BatchNormalization()(x)

변수 x에 모델의 출력을 지정합니다. Flatten()을 이용하여 모델의 출력 값을 일차원으로 바꿔주고,Dense()를 이용하여 64개의 Unit을 가진 연결층으로 바꿔줍니다.

위의 연결층을 LeakyReLU층으로 바꿔주고 배치 정규화(BatchNormalization) 해줍니다.

    x = Dense(32)(x)

    x = LeakyReLU()(x)

Dense()를 이용하여 32개의 Unit을 가진 연결층으로 바꿔주고, 연결층을 LeakyReLU층으로 바꿔줍니다.

    outs  = Dense(num\_outs, activation =custom\_activations)(x)

    model = tf.keras.Model(inputs = inputs, outputs = outs)

    return model

outs 변수에 num\_outs개의 Unit을 가진 custom\_activatios층 모델에 선언합니다.

model 변수에 inputs데이터와 outs데이터를 가진 모델을 선언한 후에 return 합니다.

model= create\_VGG()

model 변수에 위에서 정의한 VGG모델을 만든 후, model.summary()로 모델 layer층을 확인 합니다.

opt = RMSprop(learning\_rate = 0.001)

model.compile(optimizer = opt, loss = custom\_loss, metrics =  [IOU, custom\_acc])

opt 변수에 옵티마이저 RMSprop을 학습률 0.001로 지정하여 선언합니다.

위에서 구성한 모델을 compile 메소드를 호출하여 모델 학습 과정을 설정합니다.

* optimizer : 옵티마이저, 훈련 과정을 설정 하는 것 (Adam, RMSprop, SGD) 등이 있다.
* loss : 최적화 과정에서 최소화 될 손실함수를 설정하는 것, (MSE, binary\_crossentropy) 등이 있다.
* metrics : 훈련을 모니터링 하기 위해 사용한다.

ret = model.fit(x\_train, y\_train, epochs =400, batch\_size =8, verbose = 2)

위에서 구성한 모델을 fit 메소드를 호출하여 주어진 epoch수 만큼 모델을 학습합니다.

model.evaluate(x\_train, y\_train, verbose = 2)

model.evaluate(x\_test,  y\_test, verbose = 2)

위에서 구성한 모델을 evaluate 메소드를 호출하여 모델의 성능을 평가합니다. 이 코드에서는 트레이닝 데이터, 테스트 데이터의 성능을 평가합니다.

fig, ax = plt.subplots(1, 3, figsize = (10, 6))

fig과 ax변수에 1x3행렬로 그래프를 합칠 수 있게 지정해줍니다.

ax[0].plot(ret.history['loss'], "r-")

ax[0].set\_title('train loss')

ax[0].set\_xlabel('epochs')

ax[0].set\_ylabel('loss')

첫번째 그래프에 Title은 ‘train loss’, x축은 ‘epochs’ y축은 ‘loss’로 loss 기록을 빨간 선으로 기록합니다

ax[1].plot(ret.history['IOU'], "g-")

ax[1].set\_title('IOU')

ax[1].set\_xlabel('epochs')

ax[1].set\_ylabel('IOU')

두번째 그래프에 Title은 ‘IOU’, x축은 ‘epochs’, y축은 ‘IOU’로 IOU 기록을 초록 선으로 기록합니다.

ax[2].plot(ret.history['custom\_acc'], "b-")

ax[2].set\_title('custom\_acc')

ax[2].set\_xlabel('epochs')

ax[2].set\_ylabel('custom\_acc')

fig.tight\_layout()

plt.show()

세번째 그래프에 Title은 ‘custom\_acc’, x축은 ‘epochs’, y축은 ‘custom\_acc’로 custom\_acc 기록을 파란 선으로 기록합니다.

간격을 자동으로 띄우고, 위에서 그린 그래프를 보여줍니다(plt.show()).

k = 16

train\_pred = model.predict(x\_train[:k])

test\_pred  = model.predict(x\_test[:k])

학습한 모델을 predict(예측)하기 위해 각 변수에 모델의 x\_train값과 x\_test값을 인덱스 16까지 선언 합니다.

true\_label = train\_dataset["label"][:k]

pred\_label = np.argmax(train\_pred[:, -2:], axis=-1)

train\_matches = np.sum(true\_label== pred\_label)

print("train\_matches ={}/{}".format(train\_matches, k))

학습 데이터 셋을 true\_label에 선언합니다.

pred\_label 변수에 np.argmax메소드를 이용하여 train\_pred[:, -2:], axis=-1에서의 최대값의 인덱스를 가져옵니다. train\_matches 변수에 true\_label과 pred\_label 값이 같다면 합을 구합니다. 계산한train\_matches 값을 출력합니다.

true\_label = test\_dataset["label"][:k]

pred\_label = np.argmax(test\_pred[:, -2:], axis=-1)

test\_matches = np.sum(true\_label== pred\_label)

print("test\_matches ={}/{}".format(test\_matches, k))

같은 방법으로 테스트 데이터 셋을 true\_label에 선언합니다.

pred\_label 변수에 np.argmax메소드를 이용하여 test\_pred[:, -2:], axis=-1에서의 최대값의 인덱스를 가져옵니다. test\_matches 변수에 true\_label과 pred\_label 값이 같다면 합을 구합니다. 계산한test\_matches 값을 출력합니다.

k = 16

train\_pred = model.predict(x\_train[:k])

test\_pred = model.predict(x\_test[:k])

true\_label = train\_dataset[“label”][:k]

pred\_label = np.argmax(trin\_pred[:, -2:], axis = -1)

train\_matches = np.sum(true\_label == pred\_label)

true\_label = train\_dataset[“label”][:k]

pred\_label = np.argmax(train\_pred[:, -2:], axis = -1)

train\_matches = np.sum(true\_label == pred\_label)

#10 은 model.predict()로 학습된 모델에서 k=16의 훈련데이터(x\_train)와 테스트 데이터(x\_test)를 입력하여 차의 바운딩 박스(train\_box, test\_box) 위치와 분류를 수행하고, 표시합니다.

def create\_label(pred):

p = pred[:, -2:]

p = tf.argmax(p, axis = -1)

return p.numpy()

pred배열의 전체 행, 열은 뒤에서 두번째 열까지 슬라이싱한 값을 p에 저장하고, tf.argmax를 이용해서 행방향으로 최대값 위치를 다시p에 저장합니다. 그리고 p.numpy()를 리턴해줍니다.

def display\_images(img,pred, true\_box = None, k = 16, size = 224):

box = pred[:, 1:5] \* size

label = ceate\_label(pred)

fig = plt.figure(figsize = (8, k//2))

for I in range(img.shape[0]):

plt.subplot(k // 4, 4, i + 1)

plt.title('car')

a\_img = (img[i] \* 255).astype(‘uint8’)

x, y, w, h = box[i]

w2 = w / 2

h2 = h / 2

xmin = round(x – w2)

xmax = round(x + w2)

ymin = round(y – w2)

ymax = round(y + w2)

cv2.rectangle(a\_img, (xmin, ymin), (xmax, ymax), (0, 0, 255), 3)

if true\_box is not None:

xmin, ymin, xmax, ymax = true\_box[i]

cv2.rectangle(a\_img, (xmin, ymin), (xmax, ymax), (255, 0, 0), 3)

plt.imshow(a\_img)

plt.axis(“off”)

fig.tight\_layout()

plt.show()

display\_images(x\_train[:k], train\_pred, train\_dataset[“box”][:k])

display\_images(x\_test[:k]. test\_pred)

display\_images() 함수는 k개의 영상(img), 예측한 바운딩 박스(pred\_box), 분류 레이블을 표시하는 함수입니다. box에는 예측값들의 배열에 있는 1열~4열까지 즉, x, y, w, h의 값에 224를 곱해서 사이즈를 맞춘 값들을 배열로 저장합니다. 그리고 위에 있는 함수인 create\_label을 호출해서 변수label에 저장합니다. 변수 fig에는 최초 창의 크기를 가로 8, 세로 k // 2(=8)로 설정하고, img.shape[0]까지 for문을 이용해서 k // 4(=4),4인 크기의 subplot을 i+1의 위치에 label을 출력합니다.(i+1인 이유는 인덱스가 0부터 시작하기 때문) 그리고 이미지들을 0~255까지 표현하는 uint8로 타입을 변환해서 a\_img에 저장합니다.

중심점에서 크기에 대한 값(w와 h)의 1/2을 곱한 값을 빼거나 더한 값들. 즉, xmin, xmax, ymin, ymax의 값을 반올림 해서 인식한 곳에 사각형을 그리는데, 훈련 데이터의 바운딩 박스가 없으면 파란색, 있다면 빨간색으로 a\_img에 그립니다.

데이터 공유 드라이버 링크

anniotations:

<https://drive.google.com/drive/folders/1woYdOnY4xC5e6lNv9GWt5ru-zzxvC1i7?usp=sharing>

images:

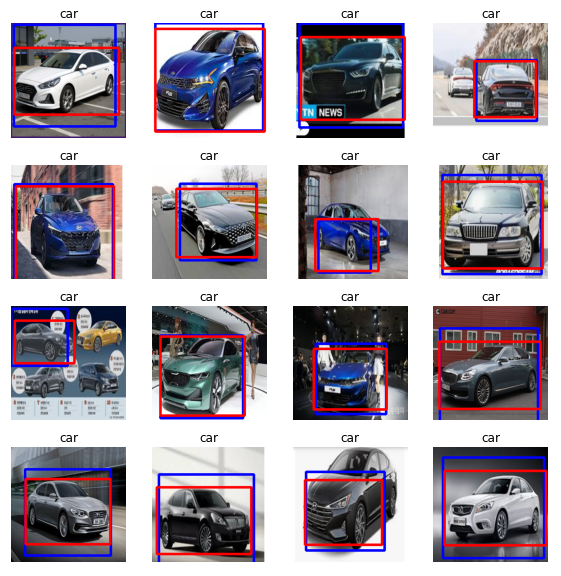
https://drive.google.com/drive/folders/1drND9ELa1\_EW3t6Dp-ovT\_Eb5p0M0uND?usp=sharing

* 1. 추진 일정

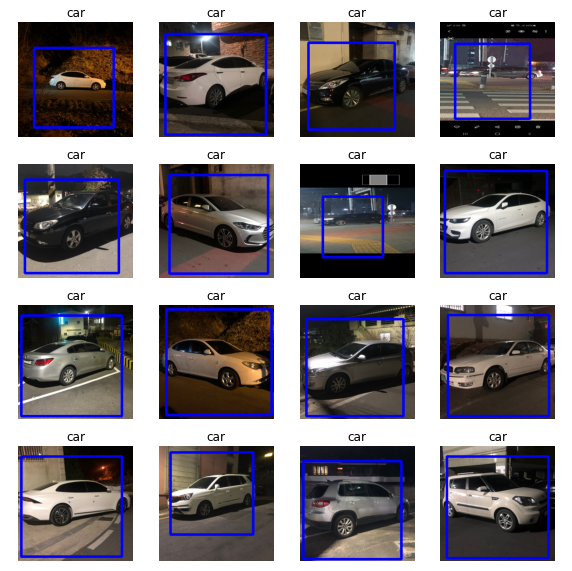
|  |  |
| --- | --- |
| 날짜 | 내용 |
| 11월 9일 | 첫 모임 |
| 11월 18일 | 역할 분배 |
| 11월 18일~11월 22일 | 데이터 수집 (train data + test data) |
| 11월 25일 | 온라인 회의(데이터 라벨링, 검토) |
| 11월 26일~11월 27일 | 프로그램 구동 및 결과 분석 |
| 11월 27일~12월 3일 | 코드 분석 |
| 12월 4일 | 보고서 작성(합치기) |

**3. 결론**

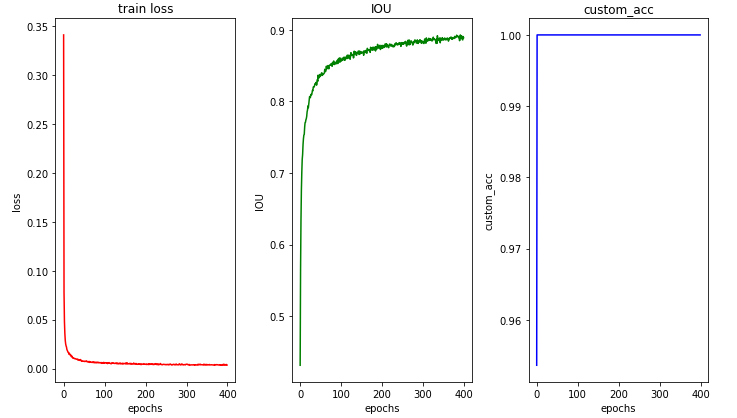
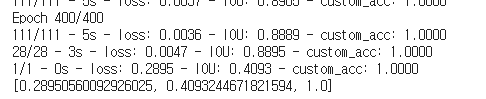
**A. 결과 분석**



위의 16개의 이미지들은 학습용 데이터들입니다. 빨간색 박스는 저희가 labelImg를 통해서 바운딩 박스를 생성했던 부분입니다. 즉, 데이터 학습을 위한 train data의 바운딩 박스를 표시한 부분이 빨간색 박스입니다. 그리고 파란색 박스는 인공지능이 학습한 내용으로 승용차라고 판단되는 부분을 보여줍니다. 16개의 결과만 임의로 추출하여 출력했을 때, train data의 바운딩 박스와 비슷하게 파란색 사각형 표시를 하고 있음을 알 수 있습니다.



Test data를 직접 촬영하여 프로그램을 실행한 결과 입니다. 16장의 사진은 공지하신 바와 같이직접 촬영하였습니다. 라벨링하지 않은 데이터들을 넣어 train data가 아닌 사진에서도 인공지능이 승용차를 찾아낼 수 있는지 확인해봤습니다. 위와 같이 파란색 박스로 보여주는데 바운딩 박스를 표시합니다. 승용차가 가깝게 찍힌 사진들은 정확도가 높게 승용차를 찾아냈고 조금 흐릿하거나 멀리에서 찍힌 승용차들도 선명한 자동차에 비해서는 정확도가 떨어지지만 전체적인 형태는 찾아 출력했습니다.



Learning\_rate는 0.001로 epochs는 400, batch\_size는 8로 hyper parameter를 설정한 후 학습했습니다.

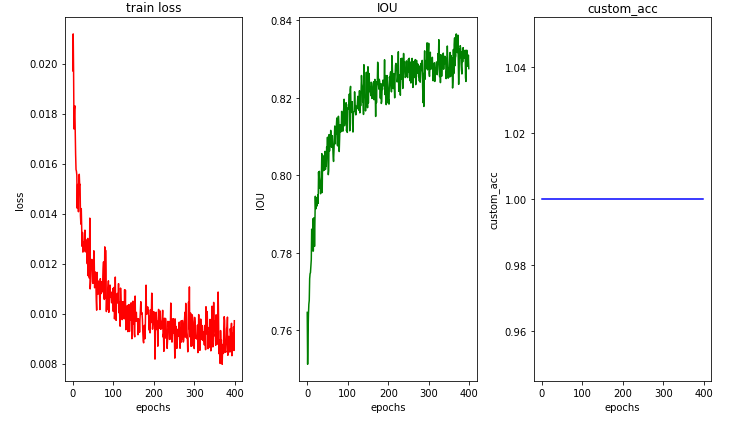
마지막 epochs=400에서 Train data의 evaluate의 결과는 Loss는 0.0047까지 떨어졌고, IOU는 약 0.8895, custom\_acc(클래스별 분류 정확도)는 1.0이 나왔습니다. Test data는 Loss는 0.2895, IOU는 0.4093, 정확도는 1.0으로 출력되었습니다.

epochs를 200으로 돌렸을 때 데이터양이 그리 많지가 않아서 인지 학습을 다 시켰음에도 불구하고 테스트 데이터 안에 승용차를 검출하지 못하거나 엉뚱한 것을 검출하는 등 정확도가 많이 떨어진 모습을 보였었습니다. 그래서 횟수를 늘려 충분한 학습이 필요하다고 느꼈습니다. Epochs를 조금씩 늘려가면서 테스트 데이터들을 확인했고 프로그램 실행시간 등을 고려했을 때, 400정도 돌렸을 때가 가장 적당하다고 판단되었습니다.

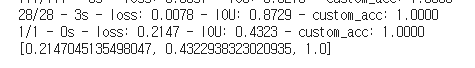
**B. 성능 향상 방안**

프로그램의 외적인 방면에서 학습하는데 필요한 하드웨어 조건들도 어느정도 맞춰야한다고 생각합니다. 처음에 train data가 부족하여 1000장 이상의 사진을 모았었는데, 프로그램을 돌리는 과정에서 900장이상은 수용하기 힘들어했습니다. Train data를 900장 밖에 사용하지 못하였고, 한 번 학습하는데에도 40분 정도 시간이 걸렸기 때문에, RAM같은 하드웨어 조건이 받쳐준다면 더 많은 학습데이터로 빠르게 학습하여 성능을 향상시킬 수 있었다고 생각합니다.

또한, 수업시간에 배운 바와 같이 learning\_rate와 epochs를 적절하게 맞춰주어야 할 것 같습니다. Epochs 역시 컴퓨터의 문제로 큰 수까지 하지 못했지만 epochs를 증가시키면 학습 횟수가 더 증가하기 때문에, 더 높은 정확도와 더 낮은 오차를 출력했을 것입니다.



Learning rate를 0.01일 때의 그래프입니다. 0.001일때와 비교했을 때, 그래프가 완만하지 않고 불안정적인 결과를 볼 수 있습니다.



Train data와 loss도 이전의 결과보다 덜 떨어진 결과를 확인할 수 있습니다. 따라서, learning rate가 낮을수록 정확한 결과와 최소한의 오차로 학습시킬 수 있다는 사실을 알 수 있습니다.

* 1. 구성원 역할

|  |  |
| --- | --- |
| 정수빈 | 데이터수집, 데이터 라벨링, 코드분석, 최종보고서 작성 |
| 최진영 | 데이터수집, 데이터 라벨링, 프로그램 빌드, 코드분석 |
| 조재훈 | 데이터수집, 데이터 라벨링, 코드분석 |
| 김채원 | 데이터수집, 데이터 라벨링, 코드분석 |
| 여성일 | 데이터수집, 데이터 라벨링, 코드분석, 발표자료 제작 |

1. **부록**
   1. 참고 문헌

**[참고 예제] 텐서플로 딥러닝 프로그래밍,** [**김동근**](http://www.yes24.com/SearchCorner/Result?domain=ALL&author_yn=Y&query=&auth_no=264176)**저 |**[**가메출판사**](javascript:void(0);)

**[Labelimg] Labelimg설치, 사용법**

[출처] Cameleong - https://blog.naver.com/a1n4s8/221571630755

[출처] Cameleong - https://blog.naver.com/a1n4s8/221571657071

**[머신러닝/딥 러닝] 합성곱 신경망(Convolutional Neural Network, CNN)과 학습 알고리즘**

[출처] CHML - https://untitledtblog.tistory.com/150

**[파이썬] 파이썬 슬라이싱 기본과 예제**

[출처] TWpower’s TechBlog - https://twpower.github.io/119-python-list-slicing-examples

**[파이썬] 파이썬 Matplotlib 기본 사용**

[출처] WikiDocs - https://wikidocs.net/92071

**[텐서플로우] TensorFlow 2.0 Keras API Overview**

[출처] 텐서 플로우 블로그 - https://tensorflow.blog/2019/03/06/tensorflow-2-0-keras-api-overview/

**[텐서플로우] TensorFlow를 이용하여 나만의 사진으로 CNN 학습시켜보기**

[출처] maru – https://emaru.tistory.com/25

[출처] maru - https://emaru.tistory.com/26

**[텐서플로우] 첫 번째 신경망 훈련하기: 기초적인 분류 문제**

[출처] TensorFlow.org - <https://www.tensorflow.org/tutorials/keras/classification?hl=ko>