

STAT431 통계학특강 기말레포트

2015100059, 통계학과, 윤창원

- 위성사진에서 이전학습과 잔차연결을 이용한 객체 분할 -

목 차

I. 주제 선정 배경	1
II. 데이터 소개	2
III. EDA 및 전처리	2
IV. 모델 학습	4
V. 한계점 및 발전방안	6
별첨. 최종 모델 Source Codes	7

요약

최근 위성사진은 딥러닝 이미지 처리 기술의 발전으로 다양한 분야에서 활용되고 있습니다. 이에, 본 과제에서는 바다를 찍은 위성사진에서 선박을 구분하는 딥러닝 모델을 만들었습니다. 이를 위해 전통적으로 이미지 처리에 사용된 CNN (Convolutional Neural Network) 모델을 사용하였으며, 객체 분할 (Instance Segmentation) 문제에 큰 효과를 보인 이전학습 (Transfer Learning)과 잔차연결 (Residual Connection)을 적용하였습니다. 이전학습은 VGG16과 Inception V3 모델, 잔차연결은 Unet 구조와 Resnet 구조를 적용하였으며, 이들을 조합하여 총 4개의 모델을 학습시키고 결과를 비교하였습니다. 비교 결과 Inception V3 모델로 이전학습을 진행하고 Resnet 구조의 잔차연결을 적용한 모델이 val_loss: 0.0078로 가장 좋은 성능을 보여 채택하였으며, 처음 보는 Test set에서도 선박을 구분해 내는 모습을 보였습니다. 이에 앞으로 해운 분야에서도 위성사진을 활용한 딥러닝 모델이 적극 사용될 수 있음을 확인하였습니다.

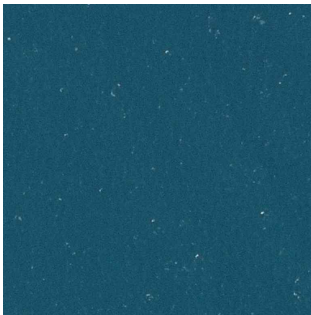
I. 주제 선정 배경

최근 우주 기술의 발달로 위성사진에 대한 접근성이 증대됨에 따라 위성사진은 전통적인 군사·안보 분야 외에도 환경, 교통·물류 등 여러 분야에서 활용되고 있습니다. 또한, 딥러닝을 이용한 이미지 처리 기술을 위성사진 분석에 접목시켜 분석의 효율성을 증대시키고 더 나은 분석 결과를 얻기 위한 노력이 계속되고 있습니다. 예를 들어, 농경지의 위성사진을 분석해 농업에 도움을 주는 기술도 연구되고 있습니다.¹⁾ 이에 위성사진을 이용한 딥러닝 모델에 흥미를 느꼈고, 동시에 수업에서 배운 다

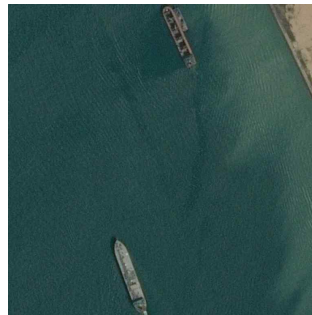
양한 이미지 처리 기법들을 적용하기에 적합한 데이터라고 판단하여 위성사진을 이용한 딥러닝 모델을 프로젝트 주제로 선정하게 되었습니다.

II. 데이터 소개

본 과제에서 사용한 데이터는 과거 Kaggle의 Competition이었던 'Airbus Ship Detection Challenge'²⁾에서 다운로드한 데이터입니다. 이는 바다를 찍은 위성사진으로, 각 이미지에는 아래와 같이 여러 척의 선박이 떠 있을 수 있습니다.



<선박이 없는 경우>

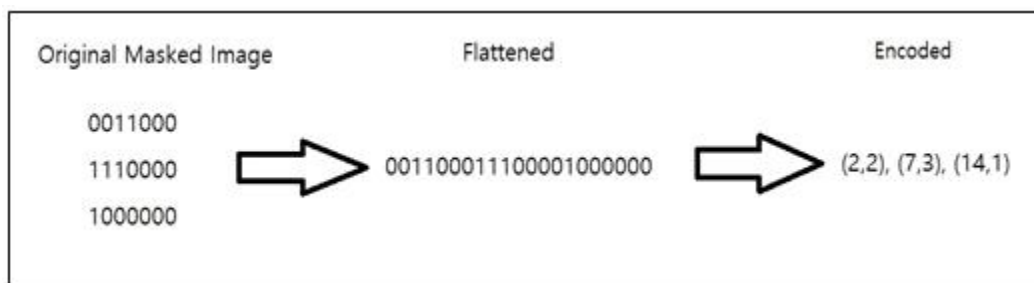


<2척의 선박이 있는 경우>

본 과제에서는 수업 7장에서 배운 객체 분할(Instance Segmentation)을 통해, 위와 같은 위성사진에서 배의 위치를 Masking 하는 이미지 데이터 처리를 수행하고자 하였습니다. 이를 통해, 해상 물류량 예측, 해상 안전 등에 도움을 줄 수 있는 모델을 만들 수 있을 것으로 기대됩니다.

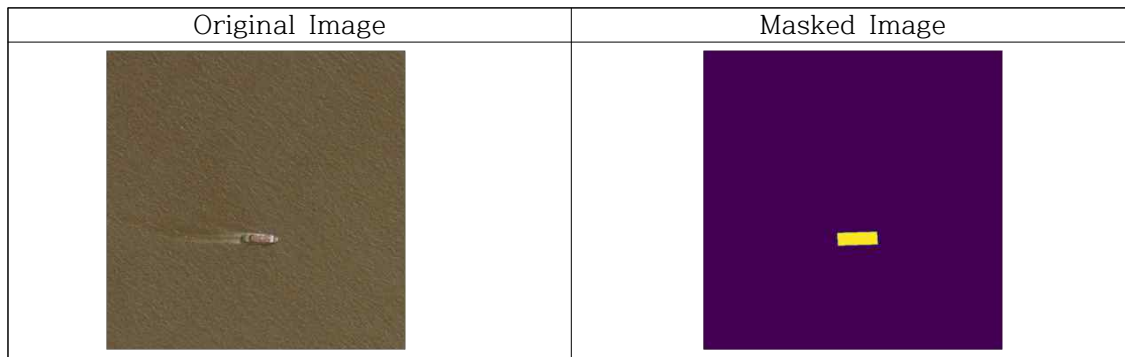
III. EDA 및 전처리

먼저 Train data의 Shape은 (768,768,3)으로 상당히 고해상도 이미지였으며, 사진의 개수가 192,556장으로 상당히 많은 수의 관측치가 주어졌습니다. 또한, Train data에 Masking 된 이미지 데이터가 따로 존재하는 것이 아니라 Run-Length-Encoding (RLE) 방법으로 Encoding 되어있어, 이를 다시 이미지 형태로 Decode 하는 과정이 필요했습니다. RLE은 기본적으로 Mask에 해당하는 부분만 1이고 나머지 부분은 0으로 처리된 Masking 된 이미지를 Flatten 시켜 0과 1만을 갖는 1차원 array로 반환한 뒤, 1이 시작되는 위치와 길이만을 저장하는 Encoding 방법입니다. 따라서, 이를 다시 이미지 형태로 반환하기 위해 해당 과정을 역으로 수행하는 RLE_decode function을 사용하였습니다.



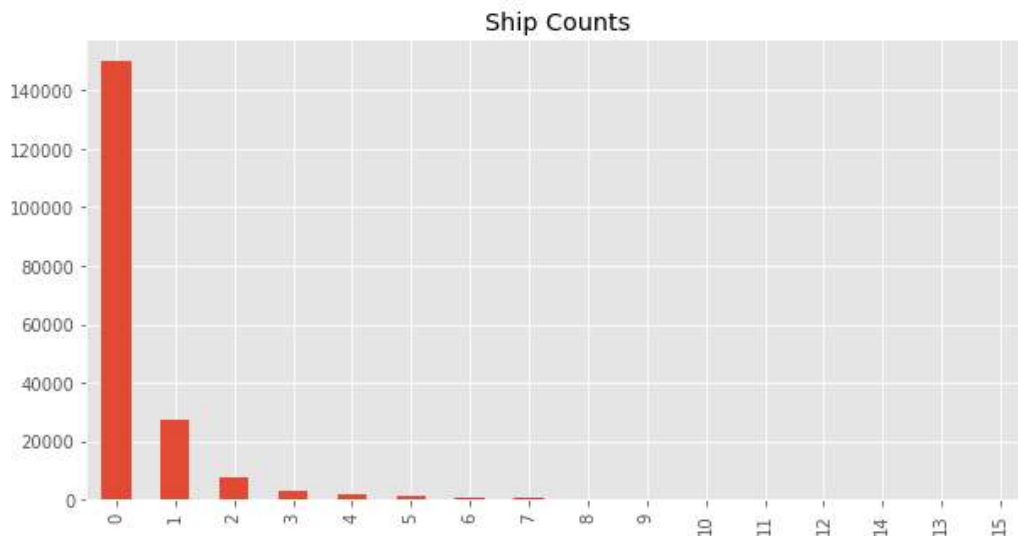
<Run-Length-Encoding 예시>

1) <https://www.rti.org/impact/using-satellite-images-and-artificial-intelligence-improve-agricultural-resilience>
2) <https://www.kaggle.com/c/airbus-ship-detection>



<Original Image와 Masked Image 예시>


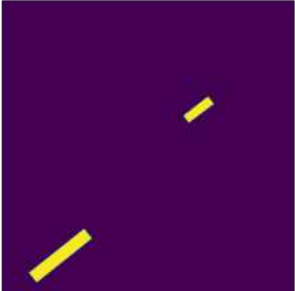
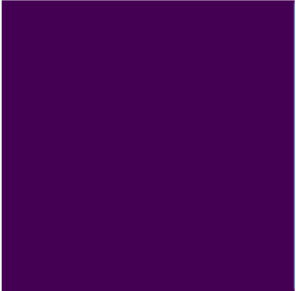
또한, 이렇게 RLE 방식으로 Encoding 된 Masking은 같은 사진에서도 선박별로 구분되어 있었기 때문에 Train data에서 이미지별 선박의 개수도 구할 수 있었습니다.



<이미지별 선박 개수의 분포>

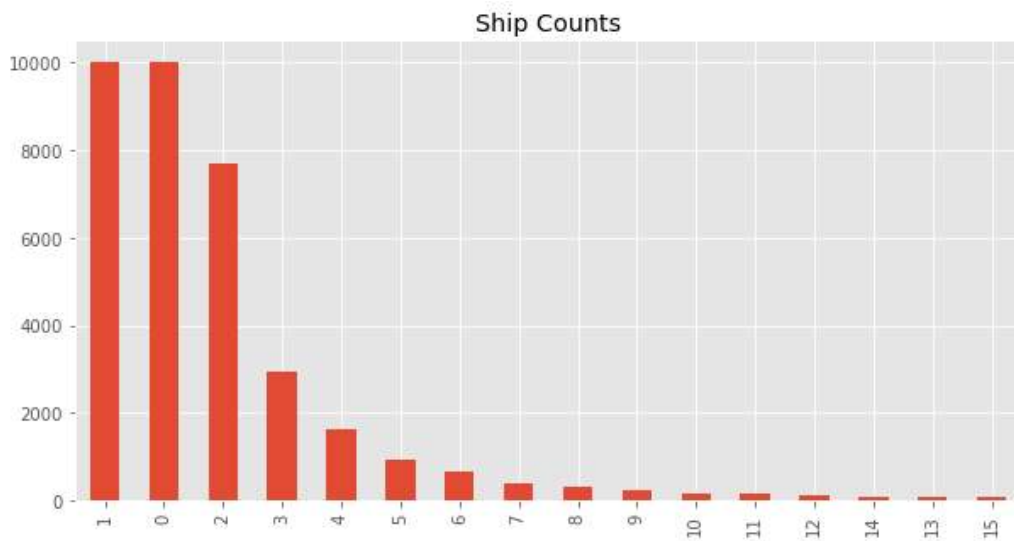
이미지별 선박 분포를 살펴보면, 선박이 없는 사진이 150,000개로, 전체 데이터의 약 80%를 차지한다는 것을 확인 할 수 있었습니다. 또한, Bar Plot에는 그 수가 너무 적어 드러나지 않지만, 한 이미지 안에 8척 이상의 선박이 존재하는 이미지도 있었으며, 최대값은 15척이었습니다. 이는 Train data가 심각하게 불균형한 데이터라는 것을 의미합니다.

실제로, 이러한 데이터의 불균형은 고해상도 이미지라는 점과 함께 모델 학습에 매우 부정적인 영향을 주었습니다. Train data를 수정하지 않은 상태에서 이전학습과 잔차연결을 이용하지 않은 간단한 모델을 학습시킨 결과, 수치상 train_acc와 val_acc가 모두 이미 99%가 넘는 것을 확인할 수 있었습니다. 하지만 실제로 Masking 된 이미지를 살펴보니 그 어느 곳도 Masking 되어있지 않았습다. 동일한 데이터를 이용해 이전학습과 잔차연결을 사용한 복잡한 모델을 학습시켰을 때도 같은 현상을 확인하였습니다. 즉, 768 x 768의 고해상도 이미지에서 선박이 차지하는 비중은 굉장히 적을뿐 아니라 선박이 존재하지 않는 이미지의 비중이 약 80%나 되었기 때문에, 모든 이미지에 선박이 없다고 학습해도 Accuracy가 99%가 넘을 수 있었던 것입니다. 일례로, 위의 Original Image와 Masked Image 예시에서 선박이 차지하는 비중은 0.5%로 Masking 결과를 모두 0으로 주어도 99.5%의 accuracy를 낼 수 있습니다.

Original Image	Original Masked Image	Predicted Masked Image
		

<고해상도와 분포 불균형으로 인한 학습 왜곡, Model acc: 99.6%>

이러한 문제점을 해결하기 위해 먼저 Train data에서 선박 개수의 분포를 대폭 수정하였습니다. 선박이 존재하지 않는 이미지와 선박의 개수가 하나인 이미지에서 각각 10,000개의 표본을 랜덤 추출하여 Train data를 재구성하였습니다. 이를 통해 기존에 비해 훨씬 균형 잡힌 데이터 분포를 만들어낼 수 있었습니다.



<재구성된 Train data의 분포>

또한, 768 x 768의 고해상도 이미지를 256 x 256으로 Resize 시켰습니다. 이는 이후에 주어진 하드웨어 Resource 안에서 모델에 학습시킬 수 있는 이미지의 개수를 늘리는 데에도 긍정적인 영향을 주었습니다.

마지막으로, Resize 한 Train data를 255로 나눠 0에서 1 사이의 값을 가지도록 표준화하는 것으로 데이터 전처리를 마무리하였습니다.

IV. 모델 학습

객체 분할을 수행하기 위해 이전학습과 잔차연결을 적용한 모델을 고려하였습니다. 이전학습은 VGG16 그리고 Inception V3 모델을 각각 적용하였으며 잔차연결은 Unet 구조와 Resnet 구조를 각각 적용하였습니다. 이에 따라 이전학습 모델과 잔차연결 구조의 조합으로 총 4개의 모델을 만들고 각

모델에 동일한 데이터를 학습시켜 성능을 비교하였습니다. 하드웨어 Resource의 한계로 인해, 재구성한 Train data 약 3만 장의 사진 중 1000장을 랜덤 추출하여 동일한 데이터로 학습을 진행했습니다. 각 모델의 세부사항은 다음과 같습니다.

i) VGG16 + Unet

7장에서 제공된 프로그램과 동일한 조합으로 VGG16 이전학습 모델에 Unet 구조의 잔차연결을 접목한 모델입니다.

ii) VGG16 + Resnet

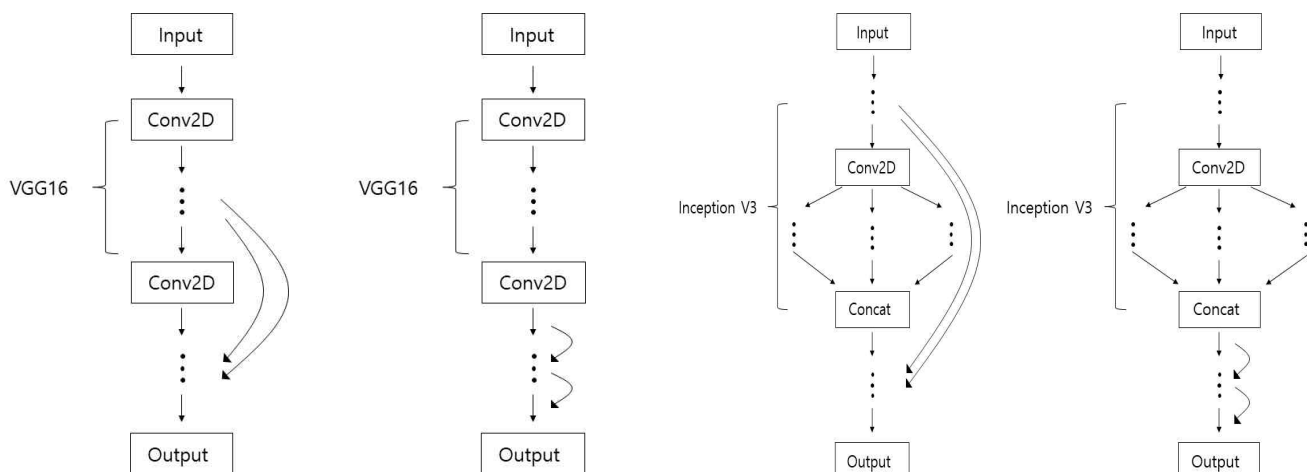
VGG16 이전학습 모델에 Resnet 구조의 잔차연결을 접목한 모델입니다. VGG16 모델의 Layer에 Resnet 구조의 잔차연결을 쓰게 되면 VGG16 모델의 구조를 변경해야 하기 때문에, 자체 Layer부터 잔차연결을 적용하였습니다.

iii) Inception V3 + Unet

Inception V3 이전학습 모델에 Unet 구조의 잔차연결을 접목한 모델입니다. Inception V3의 경우 기본적으로 병렬형 아키텍처를 가지고 있는 모델이면서 VGG16에 비해 은닉층이 훨씬 깊다는 특징을 가지고 있습니다. (include top = False에서 총 Layer 수 311개) 이에 Inception V3 모델의 특징을 살리면서 은닉층 깊이를 조절하기 위해, 첫 번째 병렬형 아키텍처가 끝나는 시점까지의 Layer만을 이용했습니다. 또한, 병렬형 아키텍처에 포함된 Layer은 잔차연결을 적용하기 힘들기 때문에 그 이전의 Layer를 Unet 구조로 연결하였습니다.


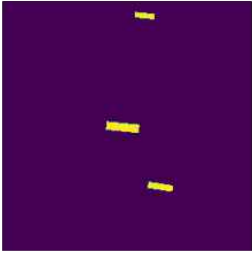
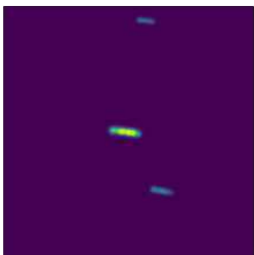
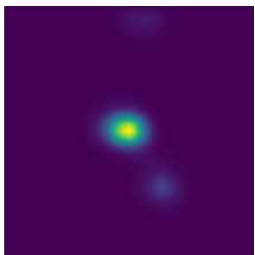
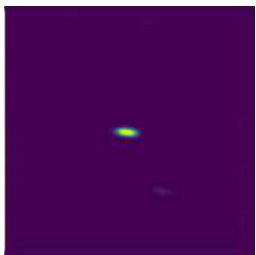
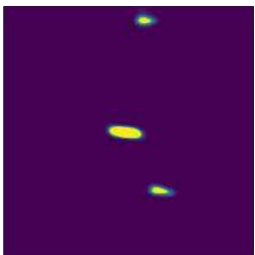
iv) InceptionV3 + Resnet

Inception_V3 이전학습 모델에 Resnet 구조의 잔차연결을 접목한 모델입니다. InceptionV3 + Unet 모델처럼 첫 번째 병렬형 아키텍처가 끝난 시점부터 자체 Layer를 연결하였으며, VGG16 + Resnet 모델과 동일하게 자체 Layer부터 잔차연결을 적용하였습니다.



<모형의 간단한 도식화 (본문 소개 순서와 동일)>

다음은 각 모델의 성능 비교표입니다. 수치상 비교는 val_acc가 아닌 val_loss를 기준으로 하였습니다. val_acc의 경우 모두 99%가 넘어 자세한 비교가 힘들기 때문입니다. 또한, 전처리 과정에서 언급했듯, 수치만으로는 모델 성능을 정확하게 파악하기 어렵기 때문에 별도의 Test set을 만들어 모델이 예측한 Masking을 시각화하여 비교하였습니다.

Original Image		Original Masked Image	
			
VGG16 + Unet (best val_loss: 0.0112)	VGG16 + Resnet (best val_loss: 0.0123)	InceptionV3 + Unet (best val_loss: 0.0184)	InceptionV3 + Resnet (best val_loss: 0.0078)
			


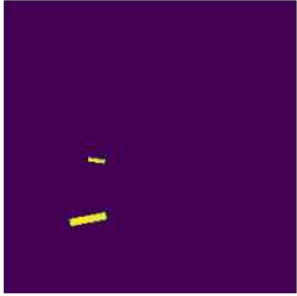
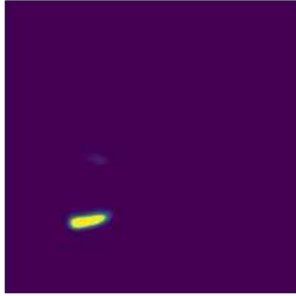
<모델별 성능 비교표>

모델 비교 결과, 수치상으로는 InceptionV3 + Resnet 모델이 val_loss: 0.0078으로 가장 좋은 성능을 보여주었습니다. 실제 예측한 Masking 결과를 비교하면, VGG16 + Resnet 모델과 InceptionV3 + Unet 모델은 수치상으로는 크게 차이를 보이지 않음에도 나머지 두 모델에 비해 선박을 제대로 Masking 하지 못하거나, Masking 범위를 너무 광범위하게 표시하는 등 실망스러운 모습을 보여주었습니다. 흥미로운 점은 좋은 성능을 보여준 VGG16 + Unet 모델과 InceptionV3 + Resnet 모델이 이전학습 모델과 잔차연결 방식에서 서로 완전히 다른 조합을 선택했다는 것입니다. 이는, 한가지 우월한 이전학습 모델과 잔차연결 구조가 있는 것이 아니라 조합에 따라 다른 시너지를 낼 수 있음을 보여줍니다. 최종모델로는 수치상으로도, Prediction Image 상으로도 가장 좋은 모습을 보여준 InceptionV3 + Resnet 모델을 선정하였습니다.

V. 한계점 및 발전방안

본 과제를 통해 위성사진에서 선박을 구별할 수 있는 딥러닝 모델을 만들면서 다시 한번 딥러닝이 이미지 처리에서 얼마나 유용하고 강력한 도구가 될 수 있는지 느꼈습니다. 다만, 본 과제를 수행함에 다음과 같은 아쉬운 점도 존재하였습니다.

첫 번째는 하드웨어 상의 한계와 시간적 한계로 인해 모델에 1000장의 사진밖에 학습시키지 못한 점입니다. 이에 가장 성능이 좋았던 최종 모델에서도 몇몇 사진에서 선박을 제대로 찾아내지 못하는 모습을 보여주었습니다. 이후 더 좋은 하드웨어 혹은 충분한 시간을 거쳐 주어진 Train data 전체에 모델을 반복 학습시킬 수 있다면 더 좋은 성능을 보일 것으로 기대됩니다.

Original Image	Original Masked Image	Predicted Masked Image
		

<모델의 선박 식별 실패 사례>

두 번째는 선박의 개수까지 세는 다중 출력 구조의 딥러닝 아키텍처를 구현하지 못한 점입니다. 초기에는 단순히 선박을 구분해내는데 그치지 않고 구분한 선박의 개수까지 반환하는 다중 출력 구조를 구상하였습니다. 이를 위해 이전학습 Layer 이후 다중 출력으로 갈라지는 구조, Input에서부터 갈라지는 구조, 개수를 세는 것이 아니라 선박의 개수를 그룹화하여 {없음, 적음, 많음}으로 Classify 하는 구조, 혹은 Masking이 완료된 이미지를 바탕으로 개수를 세는 구조 등 다양한 경우를 조합하고 실험했으나 안타깝게도 모든 경우 유의미한 결과를 보이지 못했습니다. 이후 해당 부분까지 완성된다면 더 발전되고 유용한 모델이 될 것으로 기대됩니다.

별첨. 최종 모델 Source Codes

```
import tensorflow as tf
physical_devices = tf.config.experimental.list_physical_devices('GPU')
assert len(physical_devices) > 0, "Not enough GPU hardware devices available"
tf.config.experimental.set_memory_growth(physical_devices[0], True)

## Data Preprocessing

# import necessary libraries
import numpy as np
import glob, os, cv2
import matplotlib.pyplot as plt
import pyprind
import pandas as pd
import random
import pickle

img_dir = 'C:/Users/Changwon Yoon/Desktop/Deep Learning/Project/airbus-ship-detection/train_v2/'
img_paths = glob.glob(os.path.join(img_dir, '*.jpg'))
img_paths[:5]

seg = pd.read_csv("train_ship_segmentations_v2.csv") # import encoded masks
seg = seg.fillna(0)
```



```

img = np.zeros(shape[0]*shape[1], dtype =np.uint8)
for start, end in zip(starts, ends):
    img[start:end] =1
return img.reshape(shape).T #Modification of Direction

```

```

x = []
y = []

```

```

pbar=pyprind.ProgBar(1000)
for i in range(1000):
    img = cv2.imread(img_dir + 'WW'+train_df['ImageId'][i])
    img = cv2.resize(img,(256,256))
    img_masks = seg.loc[seg['ImageId'] == train_df['ImageId'][i], 'EncodedPixels'].tolist()
    new_masks = np.zeros((768, 768))
    for mask in img_masks:
        new_masks += rle_decode(mask)
    new_masks = cv2.resize(new_masks,(256,256))
    x.append(img)
    y.append(new_masks)
    pbar.update()

```

```

x = np.array(x)
y = np.array(y)
x = x /255
y = y.reshape(y.shape[0],y.shape[1],y.shape[2],1)

```

```

## Building InceptionV3 + Resnet Model

```

```

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,UpSampling2D, Dropout, concatenate,
BatchNormalization, Flatten, Dense, add,Conv2DTranspose
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
import tensorflow.keras.utils
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

```

```

callback_list = [EarlyStopping(monitor ='val_loss',patience =3), ModelCheckpoint('Ship_detect.h5',monitor
='val_loss', save_best_only ='True')]

```

```

base_inception = InceptionV3(input_shape = (256,256,3), include_top = False, weights ='imagenet')
base_inception.trainable = False
base_inception.summary()

```

```

concatenate0 = Model(inputs =base_inception.input,outputs =base_inception.layers[40].output).output
drop = Dropout(0.5)(concatenate0)

up1 = Conv2DTranspose(512, 5, activation = 'relu')(UpSampling2D(size =(2,2))(drop))
shortcut1 = up1
conv1 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(up1)
conv1 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv1)
conv1 = BatchNormalization()(conv1)
merge1 = add([shortcut1,conv1])

up2 = Conv2DTranspose(256, 5, activation = 'relu')(UpSampling2D(size =(2,2))(merge1))
shortcut2 = up2
conv2 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(up2)
conv2 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv2)
conv2 = BatchNormalization()(conv2)
merge2 = add([shortcut2,conv2])

up3 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(UpSampling2D(size =(2,2))(merge2))
shortcut3 = up3
conv3 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(up3)
conv3 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv3)
conv3 = BatchNormalization()(conv3)
merge3 = add([shortcut3,conv3])

conv4 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge3)

img_output = Conv2D(1, 1, activation = 'sigmoid')(conv4)

ship_detect = Model(inputs = base_inception.input, outputs = img_output)
ship_detect.summary()

from tensorflow.keras.utils import plot_model
plot_model(ship_detect, show_shapes=True)

for layer in ship_detect.layers[:41]:
    layer.trainable = False

ship_detect.compile(optimizer='Adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

history = ship_detect.fit(x,y,epochs =20,batch_size =1,validation_split =0.1,callbacks = callback_list)

```