In [ ]:

```
# Global imports
from IPython.display import Image  # for displaying images
import os
import random
import shutil
from sklearn.model_selection import train_test_split
import xml.etree.ElementTree as ET
from PIL import Image, ImageDraw
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
# Loading requirements for YOLOv5 into a separate environment from https://raw.githubus
```

Data Preparation

In [ ]:

```
import csv

# convert adhoc box format to objects we can use anywhere
def extract_from_bbox_dict(path, class_id):
    def convert_to_obj(b):
        return {
            'class_id': class_id,
            'ymin': int(b[0]),
            'xmin': int(b[1]),
            'ymax': int(b[2]),
            'xmax': int(b[3])
        }
    with open(path) as file:
        content = file.readlines()
        rows = content[1:]
        bbox_dict = dict()

        for row in rows:
            row = row.split(',[(')
            imgname = row[0]
            objects_list = row[1][:-3].split('), (')

            bbox_dict[imgname] = [convert_to_obj(b) for b in [bbox.split(', ') for bbox

    return bbox_dict

# Yolo conversion from pascal
# Each row is class x_center y_center width height format.
# Box coordinates must be normalized by the dimensions of the image (i.e. have values b
# Class numbers are zero-indexed (start from 0).
def write_yolo_files(bbox_obj_dict, img_width, img_height):
    def convert_to_yolo(obj):
        return {
            'class_id': obj['class_id'],
            'center_x': ((obj['xmin'] + obj['xmax']) / 2) / img_width,
            'center_y': ((obj['ymin'] + obj['ymax']) / 2) / img_height,
            'width': (obj['xmax'] - obj['xmin']) / img_width,
            'height': (obj['ymax'] - obj['ymin']) / img_height
        }

    for img in bbox_obj_dict.keys():
```

```
        # TODO: get rid of all these hard codings
        with open('data/labels/yolov5_encoded/' + img[:-4] + '.txt', 'w') as yolo_file:
            for o in [convert_to_yolo(obj) for obj in bbox_obj_dict[img]]:
                yolo_file.write(f"{o['class_id']} {o['center_x']} {o['center_y']} {o['w

# By creating a sorted list we can both avoid empty images for training, and restrict t
def create_annotations_list(bbox_obj_dict):
    annotations = []
    for img in bbox_obj_dict.keys():
        annotations.append(img[:-4])

    annotations.sort()
    # NOTE: annotations list will have all images with objects
    with open('annotations.txt', 'w') as anf:
        for img in annotations:
            anf.write(f"{img}\n")
    return annotations

bbox_obj_dict = extract_from_bbox_dict('bbox_dictionary.csv', '0')
# Creates labels
write_yolo_files(bbox_obj_dict, 768, 768)

annotations = create_annotations_list(bbox_obj_dict)

annotations_subset = annotations[:300]
```

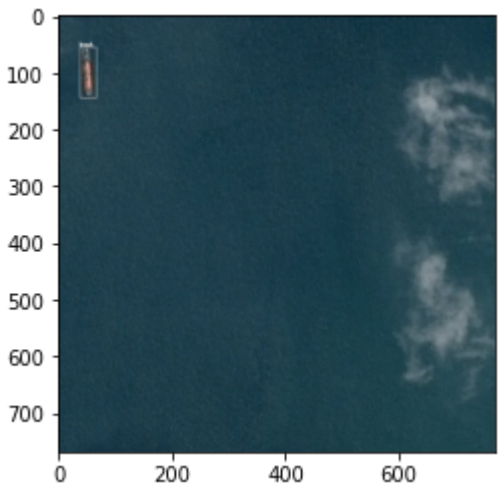
Verify that our conversions were successful

Running the cell below repeatedly will test a different image each time, drawing the bounding box from the YOLOv5 encodings we just created, and mapping the class back to a human legible label. Note: the yolov5 framework automatically converts these classes for you. You don't need to count from 80 if using COCO for example Per the original test data: boats which are occluded by the edge of the image aren't captured

In [ ]:

```
#  test box conversions
#  Adapted from https://blog.paperspace.com/train-yolov5-custom-data/

# TODO: move to constants or config
class_id_dict = {0:'boat', 'boat':0}
class_id_to_name_mapping = class_id_dict

def plot_bounding_box(image, annotation_list):
    annotations = np.array(annotation_list)
    w, h = image.size

    plotted_image = ImageDraw.Draw(image)

    transformed_annotations = np.copy(annotations)

    transformed_annotations[:,[1,3]] = annotations[:,[1,3]] * w
    transformed_annotations[:,[2,4]] = annotations[:,[2,4]] * h

    transformed_annotations[:,1] = transformed_annotations[:,1] - (transformed_annotati
    transformed_annotations[:,2] = transformed_annotations[:,2] - (transformed_annotati
    transformed_annotations[:,3] = transformed_annotations[:,1] + transformed_annotatio
    transformed_annotations[:,4] = transformed_annotations[:,2] + transformed_annotatio
```

```
    for ann in transformed_annotations:
        obj_cls, x0, y0, x1, y1 = ann
        plotted_image.rectangle(((x0,y0), (x1,y1)))

        plotted_image.text((x0, y0 - 10), class_id_to_name_mapping[(int(obj_cls))])

    plt.imshow(np.array(image))
    plt.show()

# Get any random annotation file
annotation_file = random.choice(annotations)

with open('data/labels/yolov5_encoded/' + annotation_file + '.txt', "r") as file:
    annotation_list = file.read().split("\n")[:-1]
    annotation_list = [x.split(" ") for x in annotation_list]
    annotation_list = [[float(y) for y in x ] for x in annotation_list]

#Get the corresponding image file
image_file = 'data/original/train_v2/' + annotation_file + '.jpg'
assert os.path.exists(image_file)

#Load the image
image = Image.open(image_file)

#Plot the Bounding Box
plot_bounding_box(image, annotation_list)
```



The conversion from an adhoc mask, to an adhoc bounding box, to Yolov5 appears to have been successful!

Next we split into train, val, and test images and labels: on paper only.

In [ ]:

```
images = [f"data/original/train_v2/{x}.jpg" for x in annotations_subset]
labels = [f"data/labels/yolov5_encoded/{x}.txt" for x in annotations_subset]

# Split the dataset into train-valid-test splits
# train_images, val_images, train_annotations, val_annotations = train_test_split(annot
train_images, val_images, train_labels, val_labels = train_test_split(images, labels, t
val_images, test_images, val_labels, test_labels = train_test_split(val_images, val_lab
```

Transfer files into appropriate folders for yolov5

In [ ]:
```
def copy_files(file_list, destination):
    for f in file_list:
        try:
            shutil.copy(f, destination)
        except:
            print(f)
            assert False

copy_files(train_images, 'images/train_original')
copy_files(test_images, 'images/test')
copy_files(val_images, 'images/val')
copy_files(train_labels, 'labels/train')
copy_files(test_labels, 'labels/test')
copy_files(val_labels, 'labels/val')
```

Transfer Learning

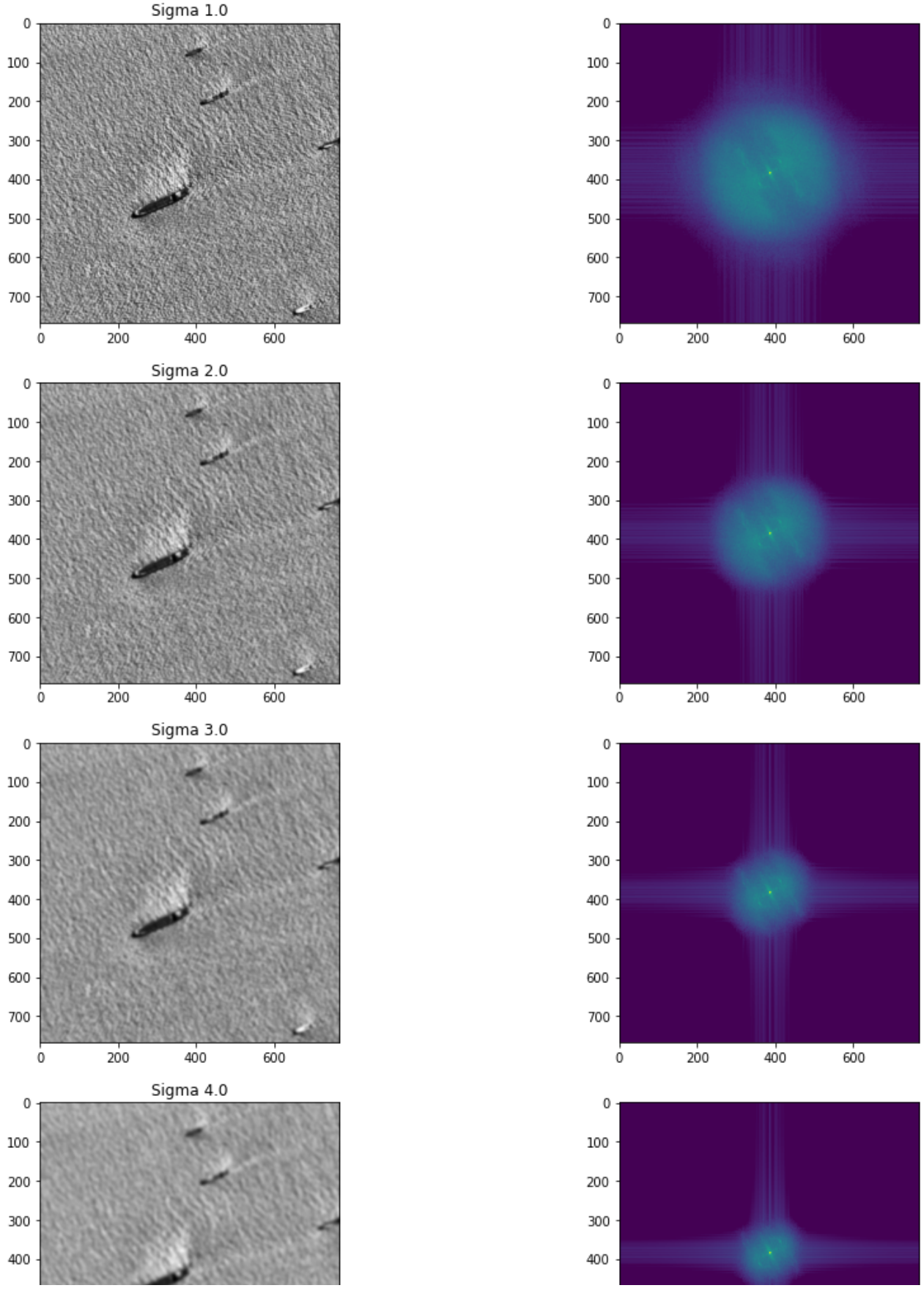
This is an example yolov5 framework call. All calls collated in the wandb model logs, as the runtime logging is far superior in console

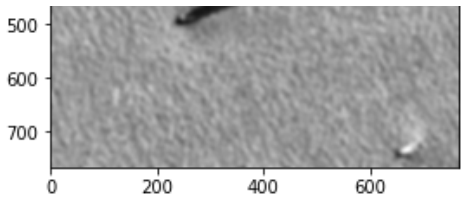
In [ ]:
```
# freeze all but output layer
!python yolov5/train.py --freeze 24 --img 384 --cfg yolov5s.yaml --hyp hyp.scratch-low.
```

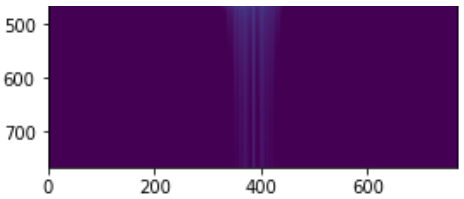
Testing gaussian filtering levels

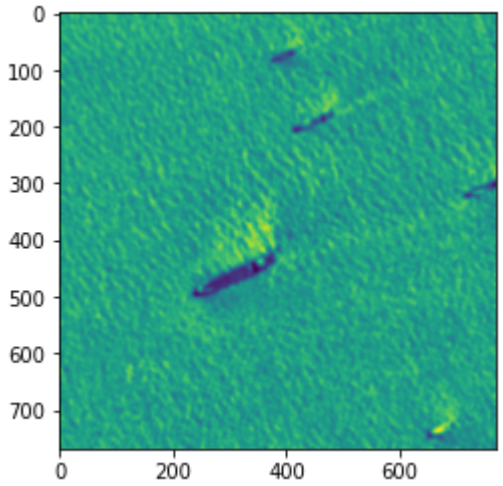
In [ ]:
```
from matplotlib.colors import LogNorm
from skimage.color import rgb2gray
from scipy import fft
from skimage import exposure

image_power = lambda x: np.square(np.abs(fft.fftshift(fft.fft2(x))))
gs_boat = exposure.equalize_adapthist(rgb2gray(image))

# fast fourier transform used to gaussian normalize
def plot_gaussian_and_fft(img, sigmas=[1.0, 2.0, 3.0, 4.0], titles=[]):
    fig, ax = plt.subplots(len(sigmas),2, figsize=(15, 20))
    for i in range(len(sigmas)):
        if len(titles) == len(sigmas):
            ax[i, 0].set_title(f"Sigma {titles[i]}")
        else:
            ax[i, 0].set_title(f"Sigma {sigmas[i]}")
        gaussian_img = filters.gaussian(img, sigma=sigmas[i])
        ax[i, 0].imshow(gaussian_img, cmap = plt.get_cmap('gray'))
        ax[i, 1].imshow(image_power(gaussian_img), norm=LogNorm(vmin=5))

    plt.show()

plot_gaussian_and_fft(gs_boat)

gaussian_2_img = filters.gaussian(gs_boat, 3.0)
_=plt.imshow(gaussian_2_img)
```

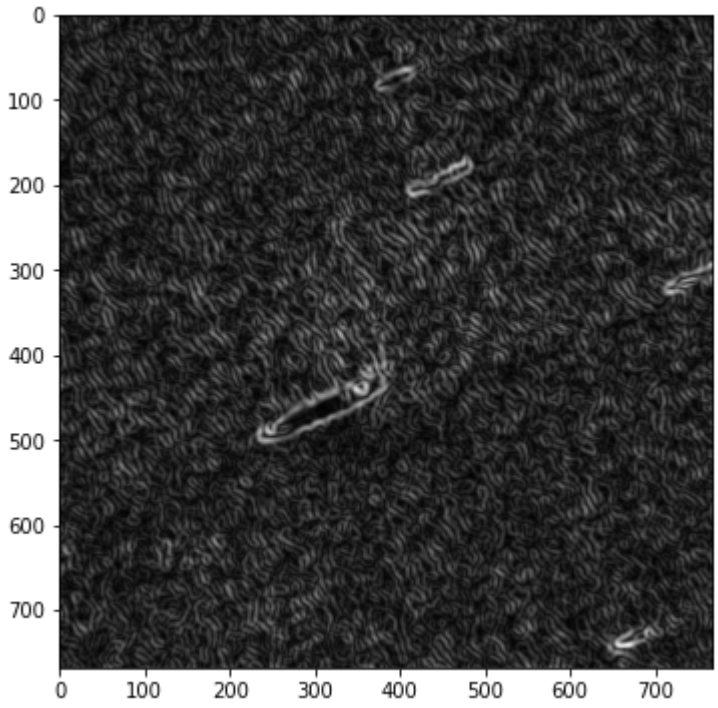
```
equalized_sobel = filters.sobel(gaussian_2_img)

fig, ax = plt.subplots( figsize=(6, 6))
_=ax.imshow(equalized_sobel, cmap=plt.get_cmap('gray'))
plt.show()
```











In [ ]:

```
# Apply gaussian normalization to reduce high frequency noise,
# and a sobel filter to enhance edge detection
from matplotlib.colors import LogNorm
from skimage.color import rgb2gray
from scipy import fft
from skimage import exposure
from skimage import io
import skimage.filters as filters

def pre_process_images(file_list, destination):
    for file in file_list:
        try:
            gs = exposure.equalize_adapthist(rgb2gray(io.imread(file)))
            img = filters.gaussian(gs, 4.0)
```

```
            es = filters.sobel(img)
            es = exposure.equalize_adapthist(es)
            threshold_value = filters.threshold_otsu(es)
            eo = es >= threshold_value

            io.imsave(f"{destination}/{file.split('/')[3][:-4] + '.png'}", eo)
        except:
            print(file)
            assert False

pre_process_images(train_images, 'images/train')
```

In [ ]:

```
# Apply only a gaussian normalization to reduce high frequency noise,

from skimage import img_as_ubyte
from matplotlib.colors import LogNorm
from skimage.color import rgb2gray
from scipy import fft
from skimage import exposure
from skimage import io
import skimage.filters as filters

def pre_process_images(file_list, destination):
    for file in file_list:
        try:
            gs = exposure.equalize_adapthist(rgb2gray(io.imread(file)))
            img = filters.gaussian(gs, 4.0)

            _=plt.imshow(img)
            plt.show()

            io.imsave(f"{destination}/{file.split('/')[3][:-4] + '.png'}", img)
        except:
            print(file)
            assert False

pre_process_images(train_images, 'images/train')
```

In [ ]:

```
# perform full tranformation on half of images
def pre_process_half_of_images(file_list, destination):
    i = 1
    for file in file_list:

        try:
            if i % 2 > 0:
                gs = exposure.equalize_adapthist(rgb2gray(io.imread(file)))
                img = filters.gaussian(gs, 4.0)

                es = filters.sobel(img)
                es = exposure.equalize_adapthist(es)
                threshold_value = filters.threshold_otsu(es)
                eo = es >= threshold_value

                io.imsave(f"{destination}/{file.split('/')[3][:-4] + '.png'}", eo)
                # _=plt.imshow(es)
                # plt.show()
```

```
            else:
                shutil.copy(file, destination)
            i = i + 1

        except:
            print(file)
            assert False

pre_process_half_of_images(train_images, 'images/train')
```