Whale Inference Module

Welcome to our Whale Inference Module!

Given a directory of aerial whale images, this module will detect each whale in each image and predict its lenght axis, width measurements, and polygon mask.

NOTE: If any of the steps fail or the Google Colab Notebook times out, please delete the session on Google Colab by going to `Runtime -> Disconnect and delete runtime` and start again.

Preliminary Setup

In this section you will install PyTorch, mount/link your Google Drive, specify a base Google Drive directory location, and load the depedencies to run the model.

Step 1 : Mount your Google Drive

Do only ONE of the options below

Option 1:

> On the left side bar click on the folder icon to open the *Files* dialog.
> Next, click on the *Mount Drive* icon on the top of this dialog (i.e. the 3rd icon that looks like a folder with the Google Drive logo)
> You will see a popup in the lower left corner saying "Mounting Google Drive...". Once your Google Drive is mounted you should see all of the contents of your drive under this *Files* dialog

Option 2:

> Or you can run the code cell below, by clicking on the *Play* button on the cell.
> This will create and link that you have to navigate to, in which you will have to copy an authentication token and paste in the input field that appears under the code cell.
> If successful it will print out a succesful mounting of your drive message.

```
#@title
import os, sys
from google.colab import drive
drive.mount('/content/mnt')
nb_path = '/content/notebooks'
```

```
Mounted at /content/mnt
```

Step 2 : Specify Directory of Technical Components

In the form cell below, type in or paste in the path to the specific Google Drive folder you uploaded our provided depedency package and model code.

Initially, you should see a placeholder name for the variable `base_drive_dir`, which is meant to visualize what your input should look like. Pay close attention to end with "/".

The underlying code for the form cell will automatically run if you insert something new in the field(s). If you do not change out the value in the field you should click the *Play* button on the cell.

Base Google Drive Directory

```
#@title Base Google Drive Directory { run: "auto" }
#@markdown Please specify Google Drive directory that contains
import os
base_drive_dir = "/content/mnt/MyDrive/base_drive_whales" #@pa
```

Please specify Google Drive directory that contains the Detectron2 models provided for the field `base_drive_dir`

base_drive_dir: " /content/mnt/MyDrive/base_drive_wh "

```
if not os.path.isdir(base_drive_dir):
    print("Base folder does not exist. Please re-enter path.")
```

Step 3 : Loading Dependencies

In this step, we load the dependencies by simply clicking the *Play* button on the code cell below. This step may take some time.

```
#@title
os.chdir(base_drive_dir)

if not os.path.exists(os.path.join(base_drive_dir, 'detectron2')):
  print("Installing Detectron2:")
  !git clone https://github.com/facebookresearch/detectron2.git
  !python -m pip install -e detectron2
  print("Detectron2 Installed")
else:
  print("Detectron2 is already present. Installing Detectron2:")
  !python -m pip install -e detectron2
  print("Detectron2 Installed")

os.chdir(os.path.join(base_drive_dir,'detectron2'))

# Common imports (already available in Colab)
import json
import random
import numpy as np
import pandas as pd
import cv2
import glob
import math
from collections import defaultdict
import datetime
from shapely.geometry import LineString
from math import hypot
import warnings
from tqdm.notebook import tqdm
import copy
import random
import matplotlib.pyplot as plt
random.seed(42)

# Detectron2 imports (not available in Colab)
import detectron2
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog
from detectron2.engine import DefaultTrainer
```

```
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.10/dist-packages (from tensorboard->detectron2==
Requirement already satisfied: grpcio>=1.48.2 in /usr/local/lib/python3.10/dist-packages (from tensorboard->detectron2
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard->detectror
Requirement already satisfied: protobuf!=4.24.0,>=3.19.6 in /usr/local/lib/python3.10/dist-packages (from tensorboard-
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard->detect
Requirement already satisfied: six>1.9 in /usr/local/lib/python3.10/dist-packages (from tensorboard->detectron2==0.6)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard->detectror
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->ter
Downloading hydra_core-1.3.2-py3-none-any.whl (154 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.5/154.5 kB 10.2 MB/s eta 0:00:00
Downloading iopath-0.1.9-py3-none-any.whl (27 kB)
Downloading omegaconf-2.3.0-py3-none-any.whl (79 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.5/79.5 kB 6.6 MB/s eta 0:00:00
Downloading yacs-0.1.8-py3-none-any.whl (14 kB)
Downloading black-24.10.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 35.1 MB/s eta 0:00:00
Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Downloading pathspec-0.12.1-py3-none-any.whl (31 kB)
Downloading portalocker-3.0.0-py3-none-any.whl (19 kB)
Building wheels for collected packages: fvcore, antlr4-python3-runtime
  Building wheel for fvcore (setup.py) ... done
  Created wheel for fvcore: filename=fvcore-0.1.5.post20221221-py3-none-any.whl size=61396 sha256=dd29e9b6c470c4b3e903
  Stored in directory: /root/.cache/pip/wheels/01/c0/af/77c1cf53a1be9e42a52b48e5af2169d40ec2e89f7362489dd0
  Building wheel for antlr4-python3-runtime (setup.py) ... done
  Created wheel for antlr4-python3-runtime: filename=antlr4_python3_runtime-4.9.3-py3-none-any.whl size=144555 sha256=
  Stored in directory: /root/.cache/pip/wheels/12/93/dd/1f6a127edc45659556564c5730f6d4e300888f4bca2d4c5a88
Successfully built fvcore antlr4-python3-runtime
Installing collected packages: antlr4-python3-runtime, yacs, portalocker, pathspec, omegaconf, mypy-extensions, iopath
  Running setup.py develop for detectron2
Successfully installed antlr4-python3-runtime-4.9.3 black-24.10.0 detectron2-0.6 fvcore-0.1.5.post20221221 hydra-core-
Detectron2 Installed
```

Running Inference

In this section you will first specify the input image directory you want to perform inference on and the output directory of where you want to see your results on. And then you will run the model to get an output.

Step 1 : Specify Image Directory & Output Directory

In the form cell below, type in or paste in the path to the specific Google Drive folder for your input folder and the output folder.

The `input_dir` field is where you should insert the directory that contains your whale images. The `output_dir` field is where you should insert the directory that you want your predictions for your images to be written in.

Initially, you should see a placeholder name for both the variables `input_dir` and `output_dir`, which is meant to visualize what your input should look like. Pay close attention to end with "/".

The underlying code for the form cell will automatically run if you insert something new in the field(s). If you do not change out the value in the field you should click the *Play* button on the cell.

Input & Output Directories

Please insert the image directory path for `input_dir` and the directory you want your outputs in for `output_dir`

`input_dir:` " /content/mnt/MyDrive/base_drive_whales/input "

`output_dir:` " /content/mnt/MyDrive/base_drive_whales/output "

Show code

Step 2 : Configure Model

In this step, we make initial configurations to run inference by simply clicking the *Play* buttons next to the code cells below. Make sure to click each button in order and before clicking the next one make sure that the code cell has stopped running (i.e. the loading wheel on the *Play* will have stopped).

Show code

```
* Configuring model...
/usr/local/lib/python3.10/dist-packages/fvcore/common/checkpoint.py:252: FutureWarning: You are using `torch.load` with `
  return torch.load(f, map_location=torch.device("cpu"))
* Done configuring Predictor 1
WARNING:fvcore.common.checkpoint:Some model parameters or buffers are not found in the checkpoint:
roi_heads.keypoint_head.conv_fcn1.{bias, weight}
roi_heads.keypoint_head.conv_fcn2.{bias, weight}
roi_heads.keypoint_head.conv_fcn3.{bias, weight}
roi_heads.keypoint_head.conv_fcn4.{bias, weight}
roi_heads.keypoint_head.conv_fcn5.{bias, weight}
roi_heads.keypoint_head.conv_fcn6.{bias, weight}
roi_heads.keypoint_head.conv_fcn7.{bias, weight}
roi_heads.keypoint_head.conv_fcn8.{bias, weight}
roi_heads.keypoint_head.score_lowres.{bias, weight}
* Done configuring Predictor 2
```

Step 3 : Obtain your Outputs

In this step, we run inference and obtain results by simply clicking the *Play* button next to the code cell below.

As the code cell runs, you will see feedback printed out under the cell as the model runs on each subsequent image, and once complete it will give a confirmation that describes where you can view your outputs at.

If you want to view the outputs in the Colab notebook, set `viz = True` in the code.

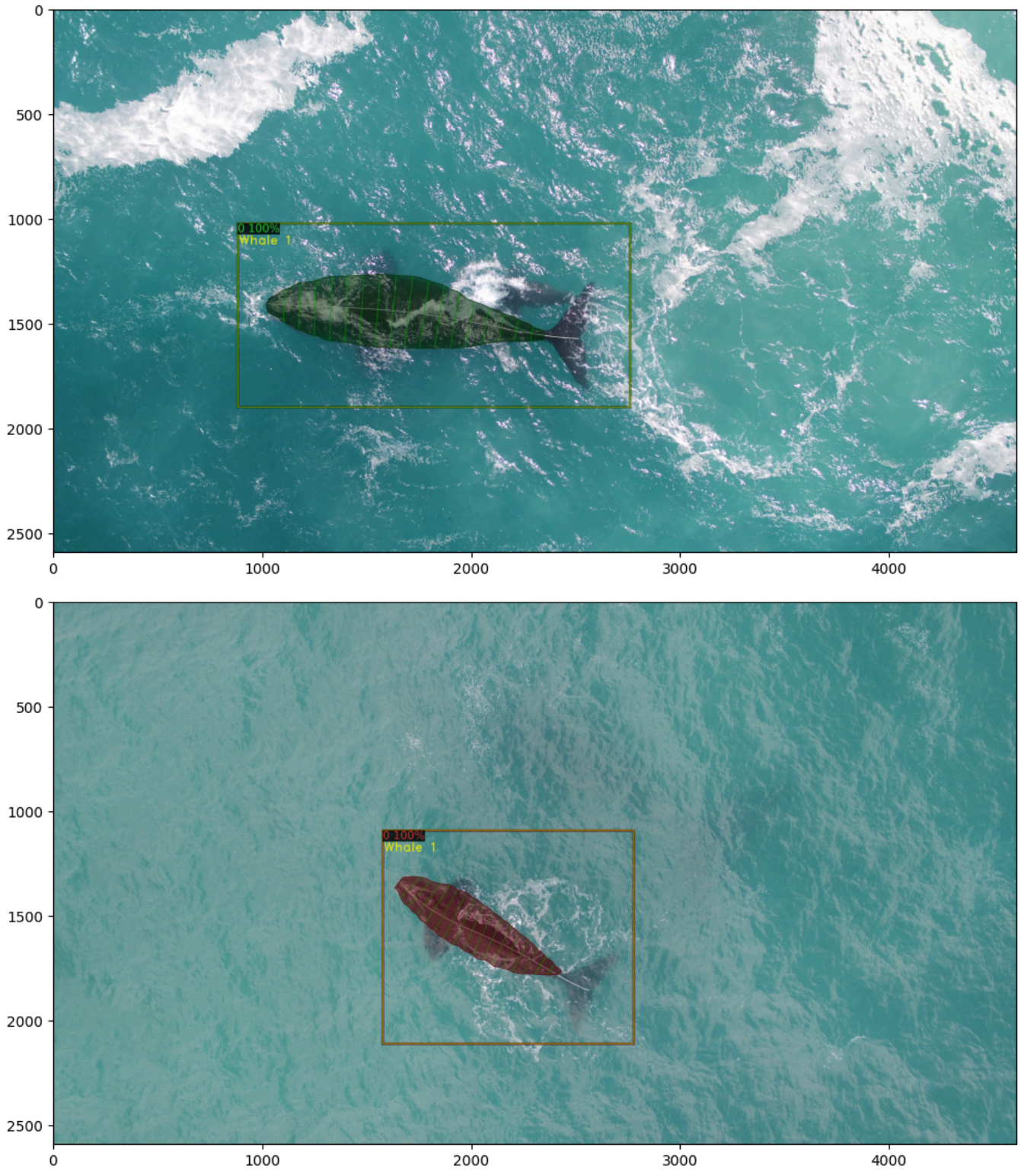
```
#@title

viz = True

import traceback
import signal
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")

def handler(signum, frame):
  # print("Forever is over!")
  raise Exception("Timed out")

## Create required output folders
ct = datetime.datetime.now()
date_str = ct.strftime("%Y_%d_%m_%H%M%S")
output_dir = os.path.join(output_dir, 'output_'+ date_str)
os.makedirs(output_dir)
os.makedirs(os.path.join(output_dir, 'predicted_images'))
os.makedirs(os.path.join(output_dir, 'masks'))
print("* Begun prediction...")

final_intersection=[]
number_of_files=len([name for name in os.listdir(input_dir) if os.path.isfile(os.path.join(input_dir, name))])
count=1
nseg = 20

# if input_dir[-1] == '/':
#     input_dir_select_all = input_dir+'*'
# else:
#     input_dir_select_all=input_dir+'/*'

input_dir_select_all = os.path.join(input_dir, '*')

all_files = glob.glob(input_dir_select_all)

final_intersection=[]
number_of_files=len([name for name in os.listdir(input_dir) if os.path.isfile(os.path.join(input_dir, name))])
count=1
debug = True
# if input_dir[-1]=='/':
#     input_dir_select_all=input_dir+'*'
# else:
#     input_dir_select_all=input_dir+'/*'

for filename in tqdm(glob.glob(os.path.join(input_dir, '*'))):
```

```
    for filename in tqdm(glob.glob(os.path.join(input_dir, '*'))):
        # print ("Processing image ",count,'/',number_of_files)
        count+=1
        # if count==10:
        #   break
        # if count<637:
        #   continue
        # print(filename)
        try:
          im = cv2.imread(filename)

          outputs1 = predictor1(im)
          outputs2 = predictor2(im)
          im_temp=copy.deepcopy(im)

          mapping_segmentation_to_keypoint = find_mappings(outputs1, outputs2)
          v = False
        except Exception:
            if debug:
              print(filename)
              traceback.print_exc()
          # print(len(mapping_segmentation_to_keypoint))
        for i in range(len(mapping_segmentation_to_keypoint)):
          im = copy.deepcopy(im_temp)
          signal.signal(signal.SIGALRM, handler)
          signal.alarm(60)
          try:
              instance_no = mapping_segmentation_to_keypoint[i][0] #Keypoint instance
              index = mapping_segmentation_to_keypoint[i][1] #Segmentation instance
              diff_arr=[]
              tens=outputs1["instances"].pred_keypoints[instance_no]
              kp_df=tens.cpu().numpy()

              midpoint = kp_df[:,:2].astype(int)

              x,y=zip(*midpoint)
              lspace=np.linspace(min(x),max(x),100)
              z=np.polyfit(x,y,5)
              draw_x=lspace
              draw_y=np.polyval(z,draw_x)
              draw_points = (np.asarray([draw_x, draw_y]).T).astype(np.int32)
              im=cv2.polylines(im, [draw_points], False, (255,255,255),thickness=2)
              intersection_points = divide_curve_into_n_segments(nseg, z, x, y)
              slopes,slope_calc_x1,slope_calc_y1,slope_calc_x2,slope_calc_y2,xvalues,yvalues=calculateSlopes(im,intersection_point

              intersection=findIntersections(outputs2["instances"].pred_masks[index],slope_calc_x1,slope_calc_y1,slope_calc_x2,slo
              area_in_sq_px=sum(sum(outputs2["instances"].pred_masks[index])).cpu().numpy()
              cv2.imwrite(os.path.join(output_dir,'masks/'+".".join((filename.split("/")[-1]).split('.')[:-1])+'.Whale.'+str(insta

              im = markWhaleID(im,instance_no,index,outputs2["instances"])
              whaleLength = calculateWhaleLength(int(tens[0][0]),int(tens[0][1]),int(tens[4][0]),int(tens[4][1]))
              intersection_val = [".".join((filename.split("/")[-1]).split('.')[:-1]),instance_no+1,int(tens[0][0]),int(tens[0][1]
              for key,val in intersection.items():
                  if len(val)==2:
                      im = cv2.line(im, tuple(val[0]), tuple(val[1]), thickness=2, color=(0, 255, 0))
                      if (np.polyval(z,val[0][0]) - val[0][1])>0: ## So that all (X1,Y1) are on one side and (X2,Y2) are on the ot
                        intersection_val.append(val[0][0])
                        intersection_val.append(val[0][1])
                        intersection_val.append(val[1][0])
                        intersection_val.append(val[1][1])
                      else:
                        intersection_val.append(val[1][0])
                        intersection_val.append(val[1][1])
                        intersection_val.append(val[0][0])
                        intersection_val.append(val[0][1])
                  else:
                      intersection_val.append(-1)
                      intersection_val.append(-1)
                      intersection_val.append(-1)
                      intersection_val.append(-1)
              final_intersection.append(intersection_val)

              v = Visualizer(im[:, :, ::-1],
                      # scale=0.8
                      scale=1
                      )
```

```
            try:
              if v:

                # print(output_dir+'predicted_images/'+".".join((filename.split("/")[-1]).split('.')[:-1])+'.WidthMarked.jpg')
                pred_mask = outputs2["instances"].pred_masks[index].cpu().numpy().astype('float32')
                img_with_overlayed_mask = overlay_mask_on_image(im, pred_mask, alpha=0.2)
                cv2.imwrite(os.path.join(output_dir,'predicted_images/'+".".join((filename.split("/")[-1]).split('.')[:-1])+f'Wh
                if viz:
                  out1 = v.draw_instance_predictions(outputs2["instances"].to("cpu"))
                  plt.figure(figsize=(12,8))
                  plt.imshow(cv2.cvtColor(out1.get_image()[:, :, ::-1], cv2.COLOR_BGR2RGB))
                  plt.show()
            # except:
            except Exception as e:
                if debug:
                  print(repr(e))
                  traceback.print_exc()
                # print("Error while processing ",filename.split("/")[-1])
                # print(sys.exc_info()[0])
                # #raise
        except Exception:
          if debug:
            print(filename)
            traceback.print_exc()
      # try:
      #   if v:
      #     out1 = v.draw_instance_predictions(outputs2["instances"].to("cpu"))
      #     # print(output_dir+'predicted_images/'+".".join((filename.split("/")[-1]).split('.')[:-1])+'.WidthMarked.jpg')
      #     cv2.imwrite(os.path.join(output_dir,'predicted_images/'+".".join((filename.split("/")[-1]).split('.')[:-1])+f'WidthM
      #     # plt.figure(figsize=(12,8))
      #     # plt.imshow(cv2.cvtColor(out1.get_image()[:, :, ::-1], cv2.COLOR_BGR2RGB))
      #     # plt.show()
      # # except:
      # except Exception:
      #     if debug:
      #       print(filename)
      #       traceback.print_exc()
      #     # print("Error while processing ",filename.split("/")[-1])
      #     # print(sys.exc_info()[0])
      #     # #raise


  # cols=["Image.ID","Whale.ID","Rostrum.X","Rostrum.Y","Fluke.Middle.X","Fluke.Middle.Y","Total.Length","Area",'Endpoint.Width.
  #       'Endpoint.Width.5.Y1', 'Endpoint.Width.5.X2', 'Endpoint.Width.5.Y2', 'Endpoint.Width.10.X1','Endpoint.Width.10.Y1',
  #       'Endpoint.Width.10.X2', 'Endpoint.Width.10.Y2', 'Endpoint.Width.15.X1','Endpoint.Width.15.Y1', 'Endpoint.Width.15.X2',
  #       'Endpoint.Width.15.Y2', 'Endpoint.Width.20.X1','Endpoint.Width.20.Y1', 'Endpoint.Width.20.X2', 'Endpoint.Width.20.Y2',
  #       'Endpoint.Width.25.X1','Endpoint.Width.25.Y1', 'Endpoint.Width.25.X2', 'Endpoint.Width.25.Y2', 'Endpoint.Width.30.X1',
  #       'Endpoint.Width.30.Y1', 'Endpoint.Width.30.X2', 'Endpoint.Width.30.Y2', 'Endpoint.Width.35.X1','Endpoint.Width.35.Y1',
  #       'Endpoint.Width.35.X2', 'Endpoint.Width.35.Y2', 'Endpoint.Width.40.X1','Endpoint.Width.40.Y1', 'Endpoint.Width.40.X2',
  #       'Endpoint.Width.40.Y2', 'Endpoint.Width.45.X1','Endpoint.Width.45.Y1', 'Endpoint.Width.45.X2', 'Endpoint.Width.45.Y2',
  #       'Endpoint.Width.50.X1','Endpoint.Width.50.Y1', 'Endpoint.Width.50.X2', 'Endpoint.Width.50.Y2', 'Endpoint.Width.55.X1',
  #       'Endpoint.Width.55.Y1', 'Endpoint.Width.55.X2', 'Endpoint.Width.55.Y2', 'Endpoint.Width.60.X1','Endpoint.Width.60.Y1',
  #       'Endpoint.Width.60.X2', 'Endpoint.Width.60.Y2', 'Endpoint.Width.65.X1','Endpoint.Width.65.Y1', 'Endpoint.Width.65.X2',
  #       'Endpoint.Width.65.Y2', 'Endpoint.Width.70.X1','Endpoint.Width.70.Y1', 'Endpoint.Width.70.X2', 'Endpoint.Width.70.Y2',
  #       'Endpoint.Width.75.X1','Endpoint.Width.75.Y1', 'Endpoint.Width.75.X2', 'Endpoint.Width.75.Y2', 'Endpoint.Width.80.X1',
  #       'Endpoint.Width.80.Y1', 'Endpoint.Width.80.X2', 'Endpoint.Width.80.Y2', 'Endpoint.Width.85.X1','Endpoint.Width.85.Y1',
  #       'Endpoint.Width.85.X2', 'Endpoint.Width.85.Y2', 'Endpoint.Width.90.X1','Endpoint.Width.90.Y1', 'Endpoint.Width.90.X2',
  #       'Endpoint.Width.90.Y2', 'Endpoint.Width.95.X1','Endpoint.Width.95.Y1', 'Endpoint.Width.95.X2', 'Endpoint.Width.95.Y2']

  # intersection_df=pd.DataFrame(final_intersection,columns=cols)

  # j=0
  # for i in range(8,intersection_df.shape[1],4):
  #     j+=5
  #     cols=intersection_df.columns[i:i+4]
  #     intersection_df["width_"+str(j)]=np.sqrt((intersection_df[cols[0]]-intersection_df[cols[2]])**2+(intersection_df[cols[1]
  # intersection_df.replace(-1,np.nan,inplace=True)



  # intersection_df.to_csv(os.path.join(output_dir,"predicted_data.csv"))

  # print("All images have been processed. Outputs can be found in ",output_dir)


  # except:
```

```
    #     print("Error while processing ", filename.split("/")[-1])
    #     print(sys.exc_info()[0])
    #     #raise


    cols = ["Image.ID", "Whale.ID", "Rostrum.X", "Rostrum.Y", "Fluke.Middle.X", "Fluke.Middle.Y", "Total.Length", "Area",
          'Endpoint.Width.5.X1', 'Endpoint.Width.5.Y1', 'Endpoint.Width.5.X2', 'Endpoint.Width.5.Y2',
          'Endpoint.Width.10.X1', 'Endpoint.Width.10.Y1', 'Endpoint.Width.10.X2', 'Endpoint.Width.10.Y2',
          'Endpoint.Width.15.X1', 'Endpoint.Width.15.Y1', 'Endpoint.Width.15.X2', 'Endpoint.Width.15.Y2',
          'Endpoint.Width.20.X1', 'Endpoint.Width.20.Y1', 'Endpoint.Width.20.X2', 'Endpoint.Width.20.Y2',
          'Endpoint.Width.25.X1', 'Endpoint.Width.25.Y1', 'Endpoint.Width.25.X2', 'Endpoint.Width.25.Y2',
          'Endpoint.Width.30.X1', 'Endpoint.Width.30.Y1', 'Endpoint.Width.30.X2', 'Endpoint.Width.30.Y2',
          'Endpoint.Width.35.X1', 'Endpoint.Width.35.Y1', 'Endpoint.Width.35.X2', 'Endpoint.Width.35.Y2',
          'Endpoint.Width.40.X1', 'Endpoint.Width.40.Y1', 'Endpoint.Width.40.X2', 'Endpoint.Width.40.Y2',
          'Endpoint.Width.45.X1', 'Endpoint.Width.45.Y1', 'Endpoint.Width.45.X2', 'Endpoint.Width.45.Y2',
          'Endpoint.Width.50.X1', 'Endpoint.Width.50.Y1', 'Endpoint.Width.50.X2', 'Endpoint.Width.50.Y2',
          'Endpoint.Width.55.X1', 'Endpoint.Width.55.Y1', 'Endpoint.Width.55.X2', 'Endpoint.Width.55.Y2',
          'Endpoint.Width.60.X1', 'Endpoint.Width.60.Y1', 'Endpoint.Width.60.X2', 'Endpoint.Width.60.Y2',
          'Endpoint.Width.65.X1', 'Endpoint.Width.65.Y1', 'Endpoint.Width.65.X2', 'Endpoint.Width.65.Y2',
          'Endpoint.Width.70.X1', 'Endpoint.Width.70.Y1', 'Endpoint.Width.70.X2', 'Endpoint.Width.70.Y2',
          'Endpoint.Width.75.X1', 'Endpoint.Width.75.Y1', 'Endpoint.Width.75.X2', 'Endpoint.Width.75.Y2',
          'Endpoint.Width.80.X1', 'Endpoint.Width.80.Y1', 'Endpoint.Width.80.X2', 'Endpoint.Width.80.Y2',
          'Endpoint.Width.85.X1', 'Endpoint.Width.85.Y1', 'Endpoint.Width.85.X2', 'Endpoint.Width.85.Y2',
          'Endpoint.Width.90.X1', 'Endpoint.Width.90.Y1', 'Endpoint.Width.90.X2', 'Endpoint.Width.90.Y2',
          'Endpoint.Width.95.X1', 'Endpoint.Width.95.Y1', 'Endpoint.Width.95.X2', 'Endpoint.Width.95.Y2']

    intersection_df = pd.DataFrame(final_intersection, columns=cols)

    j = 0
    for i in range(8, intersection_df.shape[1], 4):
        j += 5
        cols = intersection_df.columns[i:i+4]
        intersection_df["width_"+str(j)] = np.sqrt((intersection_df[cols[0]]-intersection_df[cols[2]])**2+(intersection_df[cols[1]

    intersection_df.replace(-1, np.nan, inplace=True)

    nan_array = np.empty(intersection_df.shape[0])
    nan_array[:] = np.nan

    intersection_df["Peduncle.X"] = nan_array
    intersection_df["Peduncle.Y"] = nan_array
    intersection_df["Dorsal.Fin.Start.X"] = nan_array
    intersection_df["Dorsal.Fin.Start.Y"] = nan_array
    intersection_df["Dorsal.Fin.End.X"] = nan_array
    intersection_df["Dorsal.Fin.End.Y"] = nan_array
    intersection_df["Fluke.Endpoint.X1"] = nan_array
    intersection_df["Fluke.Endpoint.Y1"] = nan_array
    intersection_df["Fluke.Endpoint.X2"] = nan_array
    intersection_df["Fluke.Endpoint.Y2"] = nan_array
    intersection_df["Blowhole.X"] = nan_array
    intersection_df["Blowole.Y"] = nan_array
    intersection_df["Eye.X1"] = nan_array
    intersection_df["Eye.Y1"] = nan_array
    intersection_df["Eye.X2"] = nan_array
    intersection_df["Eye.Y2"] = nan_array

    intersection_df.to_csv(os.path.join(output_dir, "predicted_data.csv")) # Write all data to predicted_data.csv

    print("* All images have been processed. Outputs can be found in ", output_dir)
```

```
* Begun prediction...
100%                                                 2/2 [00:26<00:00, 12.94s/it]
```




```
* All images have been processed. Outputs can be found in  /content/mnt/MyDrive/base_drive_whales/output/output_2024_2
```

Completion

To view your outputs, navigate to your output directory in your Google Drive and look for the subfolder name that the code cell above printed out. The subfolder has the naming convention of: `output_<date>_<time>` This subfolder will contain a folder of predicted images, a folder of masks, and a CSV file quantifying the predictions.

If your runtime has not timed out you can re-run the module on a different batch of images starting from the beginning of the Running Inference section (and can skip the sections above it as long as the green check marks next to the cells above remains and your base folder has not changed). You can either remove the current batch of images from your `input_dir` and upload the new batch and not change the location. Or simply give a different location path to `input_dir` referring to a new folder with the new images.