

First Step:

Phase 1: Data Collection

We collected crab data from many sources:

- MBARI
- Google open image data set
- Google image search

we used the already existed annotation from Google open image data set and annotated the rest using Labellmg

after we finished labeling we uploaded the images to google drive

Phase 2: Making the data ready for training



after the data was in google drive we used google drive API to pull the data to prepare it for training

Labellmg and Google open image data set generate the annotation in different formate so we had to convert that format to azure custom vision formate

```
def Get_Regions( mianDir ,filename):  
    if os.path.exists( mianDir + filename + ".txt" ):  
        TextFile = open( mianDir + filename + ".txt", "r").read()  
        out_regions = []  
        Regions = TextFile.splitlines()  
        for imgRegion in Regions :  
            x,y,w,h = imgRegion.split(" ")[1:]  
            x,y,w,h = float(x),float(y),float(w),float(h)  
            x,y = x-(w/2) , y-(h/2)  
            oneRegion = Region(tag_id=Crab_tag.id, left=x,top=y,width=w,height=h)  
            out_regions.append(oneRegion)  
    return out_regions
```

after that, we used the custom vision training API to upload the images to azure custom vision

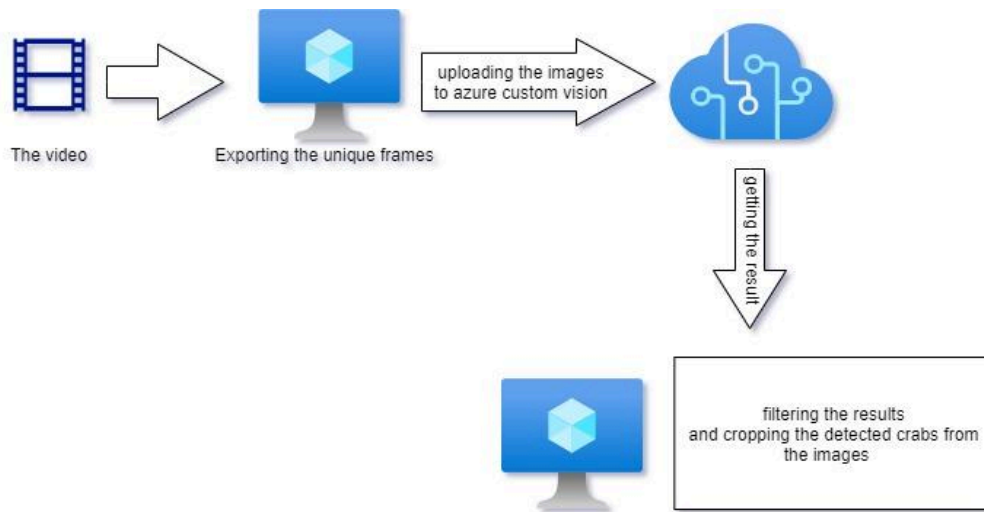
```
import os
import sys
directory = "/content/drive/MyDrive/obj/"
tagged_images_with_regions = []
patch_count = 0
total_count = 0
for filename in os.listdir(directory):

    total_count += 1
    loggingOut = str(total_count) + "/" + "5212" + " " + str((total_count/5212)*100)[:3] + "%"
    sys.stdout.write("\r" + loggingOut)
    sys.stdout.flush()
    if total_count < 2022 :
        continue

    patch_count += 1
    if filename.endswith(".jpg") :
        regions = Get_Regions(directory, filename[:-4] )
        with open(directory + filename , mode="rb") as image_contents:
            tagged_images_with_regions.append(
                ImageFileCreateEntry([name=filename[:-4],
                                     contents=image_contents.read(), regions=regions]))
    if patch_count == 64 : # maximum number of images in one upload is 64
        patch_count = 0
        try:
            Upload_Data(tagged_images_with_regions)
        except:
            print("An exception occurred while uploading")

    tagged_images_with_regions = []
```

Second Step:



It consists of three Phases:

Phase 1: exporting the unique frames from the video

when we analyzed the usage of this program we found that upload every frame in the video and try to track every detection is not efficient

so we decided to analyze the video before detecting and export only the frames that unique

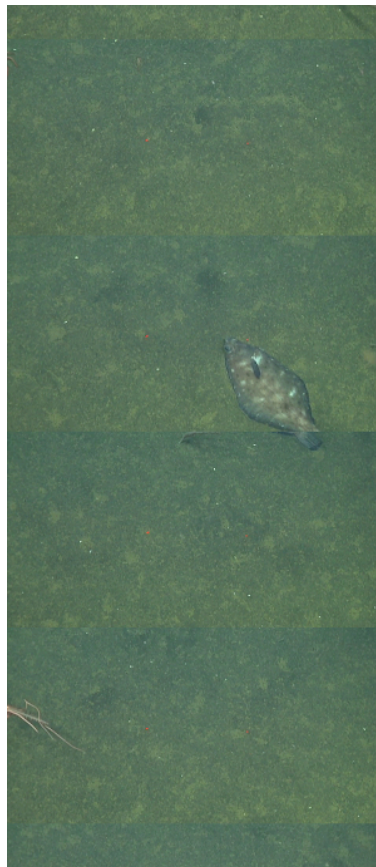
we did this by taking a point on the video



and we track it until it was outside the frame



when this happens we export this frame this technique made every frame unique from the other frame and when we send our images to the custom vision for detection we will have only a small amount of images



this image is constructed from multiple unique frames stacked in each other

the code :

```
save_data = []
import cv2
tracker = []
cap = cv2.VideoCapture("videp.mp4")
# TRACKER INITIALIZATION
success, frame = cap.read()
ImageCount = 0
def DrawNewRIO(img):
    tracker = cv2.TrackerCSRT_create()
    tracker.init(img, (250, 0,100,60))
    success, bbox = tracker.update(img)
    return tracker

tracker = DrawNewRIO(frame)

count = 0
while(cap.isOpened()):
    count +=1

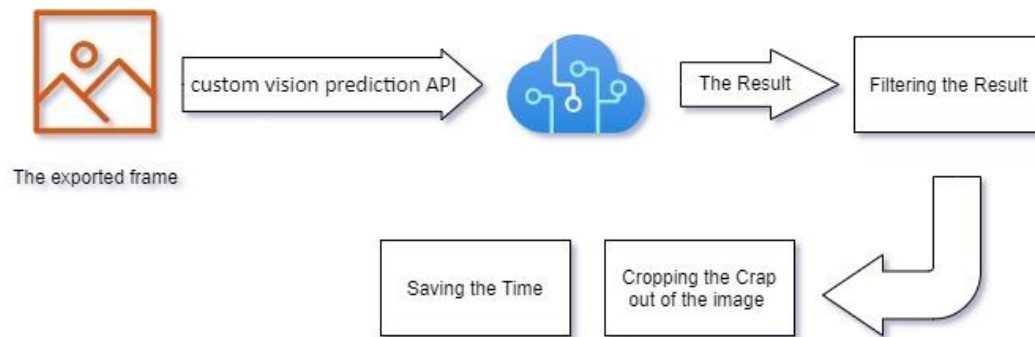
    ret, img = cap.read()
    if ret == False :
        break
    success, bbox = tracker.update(img)
    x, y, w, h = int(bbox[0]),int(bbox[1]),
    int(bbox[2]),int(bbox[3])
    if y > img.shape[0] - 60 :

        tracker =DrawNewRIO(img)

        img1 = img[0:-55, :]
        cv2.imwrite("buffer" +str(count)+ ".jpg" ,img1 )
        data = UpladFrame(count, ImageCount)
        if data :
            ImageCount += 1
            for i in data :
                save_data.append(i)

    if success:
        pass
    else:
        tracker = DrawNewRIO(img)
```

Phase 2: uploading the images and filtering the data



after we get the frame from the last code we upload the frame to the custom vision using the custom vision prediction API

after we receive the result we filter the detection probability and crop the detected crabs

we save the time + the path to the detected crab image to a CSV file for the next phase

the code :

```
import os
import math

def UpladFrame(count, ImageCount):
    p_count = 1
    save_data = []
    with open("buffer" + str(count) + ".jpg", mode="rb") as test_data:
        results = predictor.detect_image_with_no_store(project.id, "Iteration2", test_data)
        image = cv2.imread("buffer" + str(count) + ".jpg")

    for prediction in results.predictions:
        if prediction.probability > 0.9995 and prediction.tag_name == "crab" :

            #convert the azure bbox coordinates to opencv coordinates
            box = prediction.bounding_box
            h,w,_ = image.shape
            p1 =( math.floor( box.left * w ) , math.floor(box.top * h ) )
            p2 = math.floor( (box.left * w )+ box.width* w ) , math.floor( (box.top * h ) + box.height* h )

            #crob the image
            outputimg = image[p1[1]: p2[1] ,p1[0]: p2[0] ]
            cv2.imwrite("imgs9/buffer" + str(ImageCount)+ "_" + str(p_count)+ ".jpg",outputimg )

            #save the data
            save_data.append([ convertFpsToTime(count) , "imgs9/crab" + str(ImageCount)+ "_" + str(p_count)+ ".jpg"])

            p_count += 1

            # print("\t" + convertFpsToTime(count) + " " + prediction.tag_name + ": {0:.2f}% bbox.left = {1:.2f}, b

    os.remove("buffer" + str(count) + ".jpg") #remove the buffer image
    if results.predictions :
        return save_data
```

Phase C: The Instance Segmentation Model

Here we made an instance segmentation model using pytorch and

detectron 2.

At First, we annotated the train image in labelme then we converted the annotation date to COCO format using this script:

Mounting the drive

```
from google.colab import drive
drive.mount("/content/drive")
```

Converting the data from labelme format to COCO format

```
import labelme2coco

labelme_folder = "/content/drive/MyDrive/test"

save_json_path = "/content/drive/MyDrive/test_coco.json"

labelme2coco.convert(labelme_folder, save_json_path)
```

Then we started coding the Detectron2 script:

Importing Dependencies

```
import torch, torchvision
assert torch.__version__.startswith("1.8")

import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt

# import some common detectron2 utilities
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
```

Getting data

```
import detectron2
from detectron2.utils.logger import setup_logger
```

```

setup_logger()

# import some common libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt

# import some common detectron2 utilities
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

```

Mounting drive

```

from google.colab import drive
drive.mount('/content/drive')

```

Registration of data

```

from detectron2.data.datasets import register_coco_instances

for d in ["train", "test"]:
    register_coco_instances(f"crab_{d}", {}, f"/content/drive/MyDrive/Crab/{d}.json", f"/content/drive/MyDrive/Crab/{d}")

```

Testing dataset before training the model

```

import random
from detectron2.data import DatasetCatalog, MetadataCatalog

dataset_dicts = DatasetCatalog.get("crab_train")
crab_metadata = MetadataCatalog.get("crab_train")

for d in random.sample(dataset_dicts, 3):
    img = cv2.imread(d["file_name"])
    v = Visualizer(img[:, :, ::-1], metadata=crab_metadata, scale=0.5)
    v = v.draw_dataset_dict(d)
    plt.figure(figsize = (14, 10))
    plt.imshow(cv2.cvtColor(v.get_image()[ :, :, ::-1], cv2.COLOR_BGR2RGB))
    plt.show()

```

Training the Model

```

from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg
import os

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("crab_train")
cfg.DATASETS.TEST = ()

```



```

cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 1000
cfg.SOLVER.STEPS = []           # do not decay learning rate
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()

```

Using the model for inference

```

cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
cfg.DATASETS.TEST = ("crab_test", )
predictor = DefaultPredictor(cfg)

```

Reading csv output file from phase A

```

import csv
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/crab_imgs/CrabData.csv')

```

Testing the model

```

from detectron2.utils.visualizer import ColorMode
dataset_dicts = DatasetCatalog.get("crab_test")

for index, row in df.iterrows():
    im = cv2.imread('/content/drive/MyDrive/crab_imgs/' + row['ImgName'])

    outputs = predictor(im)
    v = Visualizer(im[:, :, ::-1],
                  metadata=crab_metadata,
                  scale=0.8,
                  instance_mode=ColorMode.IMAGE_BW)
    v = v.draw_instance_predictions(outputs["instances"].to("cpu"))

    scores = outputs["instances"].get('scores')
    scores = scores.to('cpu')
    print('/content/drive/MyDrive/crab_imgs/' + row['ImgName'])
    print('scores : ', scores)

    mask = outputs["instances"].get('pred_masks')
    mask = mask.to('cpu')

    mask = mask.numpy()
    mask = mask.astype(int)
    mask = mask * 255

```

```

cv2.imwrite('output_'+ row['ImgName'], v.get_image()[ :, :, :-1])
cv2.imwrite('mask_'+ row['ImgName'], mask[0])
plt.imshow(cv2.cvtColor(v.get_image()[ :, :, :-1], cv2.COLOR_BGR2RGB))
plt.show()

```

Downloading data as zip files

```

!zip -r /content/mask.zip /content/mask_imgs9
!zip -r /content/output.zip /content/output_imgs9

from google.colab import files
files.download("/content/mask.zip")
files.download("/content/output.zip")

```

then we unzipped the data in the following code folder

```

import numpy as np
import cv2
import csv
import pandas as pd

df = pd.read_csv('CrabData.csv')

crabs = []

for index, row in df.iterrows():
    img = cv2.imread('mask_imgs9/' + row['ImgName'].replace('imgs9/', ''))
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(img_gray, 50, 255, cv2.THRESH_BINARY)
    cv2.imshow('thresh', thresh)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(thresh, contours, 0, (0, 255, 0), 3)

    for cnt in contours:
        (x, y), radius = cv2.minEnclosingCircle(cnt)

        center = (int(x), int(y))
        crab_width = radius * 2
        radius = int(radius)
        cv2.circle(img, center, radius, (0, 255, 0), 2)


        crab_width = crab_width * 22.86 / 507.28

        crabs.append(crab_width)

df['crab_width'] = pd.Series(crabs)

df.to_csv("CrabData.csv", index=False)

```



this code overwrites the csv file adding a new column called “crab_width” following with the width to each crab in order.

Then we convert the csv file to xls using Excel as well as editing the file to look nice and clean