Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

Vision Based Robot Bin Picking Project
Design Details Documentation

Y.W.S.P Amarasinghe	210035A
I.A Withanawasam	210732H
S Thuvaragan	210657G
W.A.O Janandith	210234H
K.D.W Abeyawardana	210014J
L.H.H Maduwantha	200370K
R.M.K.C Jayathissa	210258J
E.M.A.R Niroshan	210433R

Submitted in partial fulfillment of the requirements for the module
EN 2160 Electronic Design Realization

23rd June 2024

Contents

Chapter 1

Software Design Documentation

Introduction

In this report, we present an analysis of the SAM (Segment Anything Model) for its applicability in vision-based bin picking robots. SAM is a deep learning model designed to segment objects within images efficiently. We examine its performance and discuss its potential integration into robotic systems for tasks like bin picking.

1.1 Setting up SAM Model

1.1.1 Local Environment

For Windows

1. Install Python
   - Download and install the latest version of Python from the official website. https://www.python.org/downloads/
   - Make sure to check the box that says "Add Python to PATH" during installation.
2. Install Visual Studio Code
   - Download and install VS Code from the official website https://code.visualstudio.com/
3. Install Git version control system
   - To clone the SAM repository from GitHub, you'll need Git. Download and install Git from https://git-scm.com/
4. Clone SAM repository
   - First create a folder on your computer
   - Inside the folder, open a terminal
   - Copy the link from the github repo and run the following in the terminal you opened.

     ```
     git clone https://github.com/mora-bprs/SAM-model.git
     ```

   - Then drag and drop the folder you created into vs code. Now you can see the code for SAM model is now opened in vs code
5. Install dependancies

- Navigate to the SAM directory

```
cd SAM-model
```

- Install the dependancies by running the below command

```
pip install -r requirements.txt
```

For Linux

1. Install Python
   - Ubuntu usually comes with Python pre-installed. To check the installed version, you can run following code in the terminal

```
python --version
```

   - In the terminal, if Python is not installed then use,

```
sudo apt install python3
```

2. Install Visual Code Studio
   - Download the correct distribution of VScode which is compatible with your Linux-based environment using the official website
     https://code.visualstudio.com/download
3. Install Git version control system

```
sudo apt update
sudo apt install git
```

   - For linux continue step 4 and 5 same as for the windows.

For macOS

1. Install Python
   - Check if Python has been already installed in the environment by running ,

```
    python3 --version
```

   - And if you get an error you have to install Python by going to the official PSF website;
     https://www.python.org/downloads/macos/
     Go to this URL and install the python with the recommended settings and make sure you have selected the "add to path" option when installing .
2. Install Visual Studio Code
   - Download and install VS Code from the official website
     https://code.visualstudio.com/
3. Install Git version control system
   - To install Git , run following in a new terminal

```
$ brew install git
```

The other configurations are the same as others but when running the code make sure that you use

```
python3 filename.py
```

because mac already has python 2 installed and you have to differentiate those softwares.

1.2 SAM Model on Box Image dataset

1.2.1 Download Dataset

- Download the box dataset from the provided Google Drive URL using the gdown command

```
# prompt: download this "https://drive.google.com/file/d/1
    iWaDuDQKftRDZ_poWyEua7j6_leDhDQc/view?usp=sharing"
!gdown --id 1iWaDuDQKftRDZ_poWyEua7j6_leDhDQc
```

- Unzip the downloaded file and assign the path of it to a variable

```
# prompt: # unzip "box_train.zip"
!unzip box_train.zip
```

```
dataset_path = "/content/train"
```

- Import Python libraries and modules required for the code

```
import cv2 #import computer vision lib
import numpy as np #import numpy as np
import torch #import pytorch
import matplotlib.pyplot as plt
import os #importing operating system , this module provides
    a way to interact with operating system including
   accesssing env var and changing the working directory
HOME = os.getcwd()  #get the current work directory using
   the getcwd(), and assign it to home var
print("HOME:", HOME)
%cd {HOME}. #this line changes the current working directory
    to the value of the HOME
```

- Check the code if it's running in Google Colab and if so, set up the environment by importing libraries like torch and torchvision, installing necessary packages, creating directories, and downloading a sample image ('truck.jpg') and pre-trained model weights . This ensures all dependencies are in place for subsequent image processing tasks.

```
using_colab = True
if using_colab:
    import torch
    import torchvision
    print("PyTorch version:", torch.__version__)
    print("Torchvision version:", torchvision.__version__)
    print("CUDA is available:", torch.cuda.is_available())
    import sys
    !{sys.executable} -m pip install opencv-python
       matplotlib onnx onnxruntime
    !{sys.executable} -m pip install 'git+https://github.com
       /facebookresearch/segment-anything.git'

    !mkdir images
    !wget -P images https://raw.githubusercontent.com/
       facebookresearch/segment-anything/main/notebooks/
       images/truck.jpg

    # Download pretrained model
    !wget https://dl.fbaipublicfiles.com/segment_anything/
       sam_vit_h_4b8939.pth
```

1.2.2 Setup

```
import numpy as np
import torch
import matplotlib.pyplot as plt
import cv2
```

- This function is a utility for visualizing binary masks. It accepts a binary mask, an axis object for plotting, and an optional flag for randomizing colors.

  If the random color flag is enabled, it generates a random RGB color with transparency, otherwise, it uses a predefined semi-transparent blue color. The function then reshapes the mask and color array to create a colored mask image, which it displays on the provided axis.

```
def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array
            ([0.6])], axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

def show_points(coords, labels, ax, marker_size=375):
    pos_points = coords[labels==1]
    neg_points = coords[labels==0]
    ax.scatter(pos_points[:, 0], pos_points[:, 1], color='
        green', marker='*', s=marker_size, edgecolor='white',
         linewidth=1.25)
    ax.scatter(neg_points[:, 0], neg_points[:, 1], color='
        red', marker='*', s=marker_size, edgecolor='white',
        linewidth=1.25)

def show_box(box, ax):
    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    ax.add_patch(plt.Rectangle((x0, y0), w, h, edgecolor='
        green', facecolor=(0,0,0,0), lw=2))
```

1.2.3 Using Pretrained Model

- Assign the path to the pre-trained Segment Anything Model (SAM) checkpoint file and the model type to the respective variables

```
sam_checkpoint = "/content/sam_vit_h_4b8939.pth"
model_type = "vit_h"
```

- Determine the device (GPU or CPU) to be used for running the model. If a CUDA-enabled GPU is available, it sets the device to "cuda"; otherwise, it sets the device to "cpu".

```
# prompt: device = "cuda" or cpu
device = "cuda" if torch.cuda.is_available() else "cpu"
```

- Initialize the Segment Anything library by importing required modules and loading a pre-trained SAM model using specified parameters. Transfer the model into the appropriate device (GPU if available, otherwise CPU). Finally, create a predictor object, to facilitate predictions using the loaded model.

```
import sys
sys.path.append("..")
from segment_anything import sam_model_registry,
    SamPredictor

sam_checkpoint = "sam_vit_h_4b8939.pth"
model_type = "vit_h"

sam = sam_model_registry[model_type](checkpoint=
    sam_checkpoint)
sam.to(device=device)

predictor = SamPredictor(sam)
```

1.2.4 Example Image

- The `list_image_files` function recursively traverses a directory and its subdirectories to compile a list of file paths for all images found. It identifies image files by checking their file extensions. The function is invoked with the `dataset_paths` , and the resulting list of image file paths is stored in the `image_paths` variable.

```
def list_image_files(directory):
image_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp']
     # Add more if needed
image_files = []
for root, dirs, files in os.walk(directory):
    for file in files:
        if any(file.lower().endswith(ext) for ext in
            image_extensions):
             image_files.append(os.path.join(root, file))
return image_files

image_paths = list_image_files(dataset_path)
```

- Select 40th image as the sample image and assign it's path to the `sample_image_path` variable.

```
sample_img_path = image_paths[40]
```

- Convert image from the BGR color space (used by OpenCV) to the RGB color space

```
image = cv2.imread(sample_img_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

- The `plot_image` function displays an input image using Matplotlib. It generates a figure with dimensions of 10x10 inches and plot it.

```
def plot_image(image):
    plt.figure(figsize=(10,10))
```

```
        plt.imshow(image)
        plt.axis('on')
        plt.show()

    plot_image(image)
```

Figure 1.1: Example image from the box dataset

- Display the resolution of the image

```
    # prompt: get image resolution
    image_height, image_width, _ = image.shape
    print(f"Image resolution: {image_width}x{image_height}")
```

- Calculate the coordinates of the center point of the image based on its width and height.`input_point` contains the center point coordinates, and `input_label` contains a single positive label (1) corresponding to the center point.

```
    center_point_coords =[int(image_width/2),int(image_height/2)
       ]
```

```
    input_point = np.array([center_point_coords])
    input_label = np.array([1])
```

- Create a new figure with dimensions of 10x10 inches, display a loaded image, plot a center point on the image using a green star marker, and then show the plot with the axis enabled.

```
    plt.figure(figsize=(10,10))
    plt.imshow(image)
    show_points(input_point, input_label, plt.gca())
    plt.axis('on')
    plt.show()
```

Figure 1.2: Example image with a green marker

1.2.5 Mask Prediction

```
    predictor.set_image(image)
```

- Generate multiple masks, scores, and logits for a specified input point. Define the input point's coordinates and labels, instructs the predictor to produce multiple masks, and then invokes the predict method to generate the desired outputs.

```
    masks, scores, logits = predictor.predict(
    point_coords=input_point,
    point_labels=input_label,
    multimask_output=True,
)
```

- Display number of masks with heights and weights.

```
    masks.shape  # (number_of_masks) x H x W
```

- Visualize the mask, its score, and the input point on the input image

```
    def plot_mask_with_score(image, title,mask, score,
       input_point, input_label ):
        plt.figure(figsize=(10,10))
        plt.imshow(image)
        show_mask(mask, plt.gca())
        show_points(input_point, input_label, plt.gca())
        plt.title(f"Mask {title}, Score: {score:.3f}", fontsize
           =18)
        plt.axis('off')
        plt.show()
```

```
    for i, (mask, score) in enumerate(zip(masks, scores)):
        plot_mask_with_score(image, i+1, mask, score,
           input_point, input_label)
```

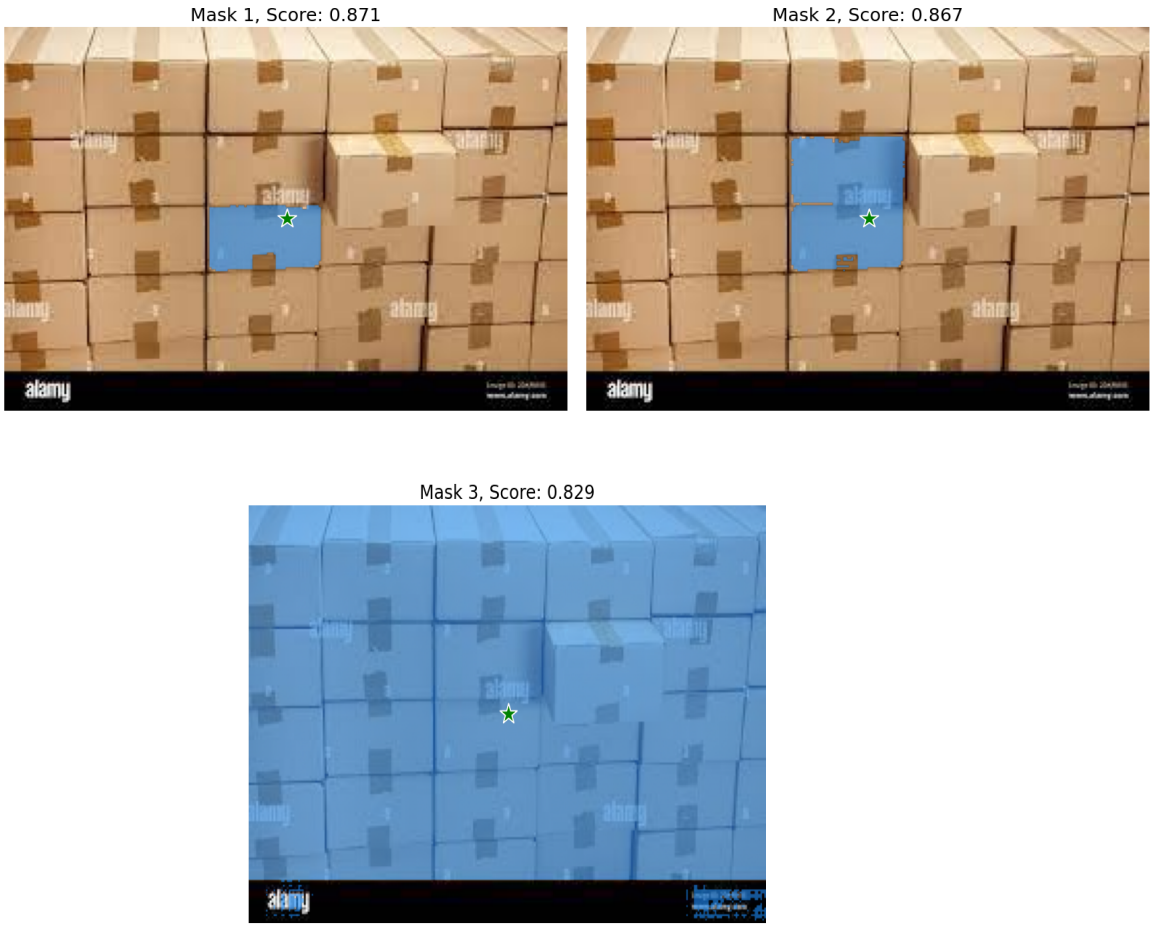


Figure 1.3: Generated Mask 03

- Find the mask with the highest score among the generated masks and visulaize it.

```
import numpy as np

def get_mask_with_highest_score(masks, scores, logits):
    # Ensure that masks, scores, and logits are numpy arrays
    masks = np.array(masks)
    scores = np.array(scores)
    logits = np.array(logits)

    # Find the index of the mask with the highest score
    max_score_index = np.argmax(scores)

    # Get the mask with the highest score
    mask_with_highest_score = masks[max_score_index]

    # Get the score and logits corresponding to the highest
       score
    highest_score = scores[max_score_index]
    logits_for_highest_score = logits[max_score_index]

    return mask_with_highest_score, highest_score,
       logits_for_highest_score
```

```
mask_with_highest_score, highest_score,
    logits_for_highest_score =
get_mask_with_highest_score(masks, scores, logits)
```

```
plot_mask_with_score(image, "best", mask_with_highest_score,
    highest_score, input_point, input_label)
```

```
sample_mask.shape
```

```
image.shape
```

Figure 1.4: Generated Mask

- Plot a bounding box around the True region in the mask.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches

def plot_square(mask):
    # Find the indices where the mask is True
    true_indices = np.argwhere(mask)

    # Get the bounding box of the True region
    top_left = np.min(true_indices, axis=0)
    bottom_right = np.max(true_indices, axis=0)

    # Calculate the width and height of the bounding box
    width = bottom_right[1] - top_left[1]
    height = bottom_right[0] - top_left[0]

    # Create a figure and axis
    fig, ax = plt.subplots(1)

    # Plot the mask
    ax.imshow(mask, cmap='gray')
```

```
        # Create a rectangle patch
        rect = patches.Rectangle((top_left[1], top_left[0]),
            width, height, linewidth=1, edgecolor='r', facecolor=
            'none')
        # Add the rectangle patch to the axis
        ax.add_patch(rect)

        # Show the plot
        plt.show()

    # Example usage:
    # Assuming you have your mask array named "mask"
    plot_square(sample_mask)
```

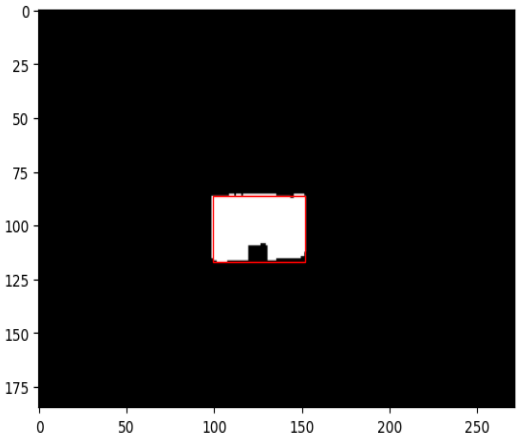


Figure 1.5: Generated Mask 03

- Get the coordinates of the bounding box corners and print the coordinates.

```
    import numpy as np

    def get_bounding_box_coordinates(mask):
        # Find the indices where the mask is True
        true_indices = np.argwhere(mask)

        # Get the bounding box of the True region
        top_left = np.min(true_indices, axis=0)
        bottom_right = np.max(true_indices, axis=0)

        # Calculate the width and height of the bounding box
        width = bottom_right[1] - top_left[1]
        height = bottom_right[0] - top_left[0]

        # Calculate the coordinates of the corners
        top_right = (top_left[1] + width, top_left[0])
        bottom_left = (top_left[1], top_left[0] + height)
        bottom_right = (top_left[1] + width, top_left[0] +
            height)
```

```
        return top_left, top_right, bottom_left, bottom_right

    # Example usage:
    # Assuming you have your mask array named "mask"
    top_left, top_right, bottom_left, bottom_right =
        get_bounding_box_coordinates(sample_mask)
    print("Top left:", top_left)
    print("Top right:", top_right)
    print("Bottom left:", bottom_left)
    print("Bottom right:", bottom_right)
```

- Get the coordinates of the bounding box corners with an adjustment for the top-left corner coordinates.

```
    import numpy as np

    def get_bounding_box_coordinates_updated(mask):
        # Find the indices where the mask is True
        true_indices = np.argwhere(mask)

        # Get the bounding box of the True region
        top_left = np.min(true_indices, axis=0)
        bottom_right = np.max(true_indices, axis=0)

        # Calculate the width and height of the bounding box
        width = bottom_right[1] - top_left[1]
        height = bottom_right[0] - top_left[0]

        # Calculate the coordinates of the corners
        top_right = (top_left[1] + width, top_left[0])
        bottom_left = (top_left[1], top_left[0] + height)
        bottom_right = (top_left[1] + width, top_left[0] +
            height)

        top_left[0] = top_right[0] - (bottom_right[0]-
            bottom_left[0])
        top_left[1] = (top_right[1]-bottom_right[1]) +
            bottom_left[1]

        return top_left, top_right, bottom_left, bottom_right
    # Example usage:
    # Assuming you have your mask array named "mask"
    top_left, top_right, bottom_left, bottom_right =
        get_bounding_box_coordinates_updated(sample_mask)
    print("Top left:", top_left)
    print("Top right:", top_right)
    print("Bottom left:", bottom_left)
    print("Bottom right:", bottom_right)
```

- Mark the corners of a bounding box on an input image using different colored markers. Then save the annotated image to a specified path. Call the function with adjusted bounding box coordinates obtained from another function.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches

def annotate_square_corners(image, top_left, top_right,
    bottom_left, bottom_right, save_path):
    # Create a figure and axis
    fig, ax = plt.subplots(1)

    # Plot the original image
    ax.imshow(image)

    # Annotate the corners
    ax.plot(top_left[0], top_left[1], 'ro')
    # Top Left corner
    ax.plot(top_right[0], top_right[1], 'go')
    # Top Right corner
    ax.plot(bottom_left[0], bottom_left[1], 'bo')
    # Bottom Left corner
    ax.plot(bottom_right[0], bottom_right[1], 'yo')
    # Bottom Right corner

    # Show the plot
    plt.show()

    # Save the plot to the specified path
    plt.savefig(save_path)

    # Close the plot to release resources
    plt.close()

# Example usage:
# Assuming you have your original image as "image" and the
    corners' coordinates obtained from get_square_corners
top_left, top_right, bottom_left, bottom_right =
    get_bounding_box_coordinates_updated(sample_mask)
annotate_square_corners(image, top_left, top_right,
    bottom_left, bottom_right, "/content/
    sample_image_annotated.jpg")
```

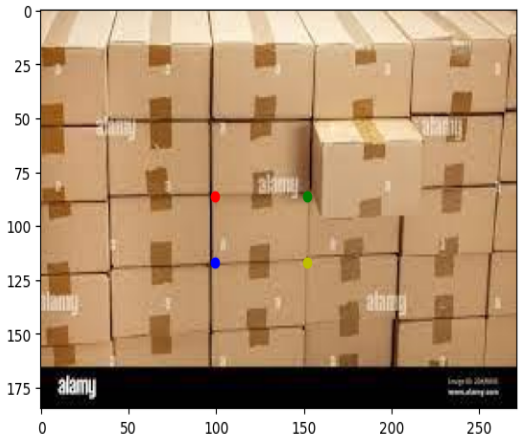


Figure 1.6: Generated Mask 03

- A modified version of the function that solely saves the annotated image without displaying it.

```
import matplotlib.pyplot as plt

def annotate_square_corners(image, top_left, top_right,
    bottom_left, bottom_right, save_path):
    # Create a figure and axis
    fig, ax = plt.subplots(1)

    # Plot the original image
    ax.imshow(image)

    # Annotate the corners
    ax.plot(top_left[0], top_left[1], 'ro')
    # Top Left corner
    ax.plot(top_right[0], top_right[1], 'go')
    # Top Right corner
    ax.plot(bottom_left[0], bottom_left[1], 'bo')
    # Bottom Left corner
    ax.plot(bottom_right[0], bottom_right[1], 'yo')
    # Bottom Right corner

    # Save the plot to the specified path
    plt.savefig(save_path)

    # Close the plot to release resources
    plt.close()

# Example usage:
# Assuming you have your image and corner coordinates
top_left, top_right, bottom_left, bottom_right =
    get_bounding_box_coordinates_updated(sample_mask)
annotate_square_corners(image, top_left, top_right,
    bottom_left, bottom_right, "/content/
    sample_image_annotated.jpg")
```

1.2.6 Pipeline

- Use the pre-trained model to create bounding boxes around objects in images. This function calculates the center point, generates masks and scores, selects the best mask, and computes bounding box coordinates. It offers options to display intermediate results.

```
    def get_box_coordinates(image_path:str,showOriginalImage=
        False, showPoints=True, showMasksWithScores=True,
        showPlotMaskWithHighestScore=True):
      # load image
      img = cv2.imread(image_path)
      img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

      # plot original image
      if showOriginalImage:
        plot_image(img)

      # get image dimensions
      img_height, img_width, _ = img.shape

      # get centre point coordinates
      center_point_coords = [ int(img_width/2), int(img_height
          /2)]
      input_point = np.array([center_point_coords])
      input_label = np.array([1])

      if showPoints:
        plt.figure(figsize=(10,10))
        plt.imshow(img)
        show_points(input_point, input_label, plt.gca())
        plt.axis('on')
        plt.show()


      # draw a rectangle that makes the center point of image

      # generate masks in the relevant area
      predictor.set_image(img)

      img_masks, img_masks_scores, img_masks_logits = predictor.
          predict(
        point_coords=input_point,
        point_labels=input_label,
        multimask_output=True,
      )

      if showMasksWithScores:
        for i, (mask, score) in enumerate(zip(img_masks,
            img_masks_scores)):
          plot_mask_with_score(img, i+1, mask, score,
              input_point, input_label)


      # get the mask with the highest score
```

```
    img_mask_with_highest_score, img_mask_highest_score,
        img_mask_logits_for_highest_score =
        get_mask_with_highest_score(img_masks, img_masks_scores
        , img_masks_logits)

    if showPlotMaskWithHighestScore:
      plot_square(img_mask_with_highest_score)

    # get the rectangular boxes
    img_top_left, img_top_right, img_bottom_left,
        img_bottom_right = get_bounding_box_coordinates(
        img_mask_with_highest_score)

    # get the coordinates of the rectangular bounding box
    annotate_square_corners(img, img_top_left, img_top_right,
        img_bottom_left, img_bottom_right)
```

```
time_list = []
```

- Iterate over image paths, measures processing time for each, and calculates the average time across all images.

```
for img_path in image_paths:
  start_time = time.time()
  get_box_coordinates(img_path,showOriginalImage=False,
      showPoints=False, showMasksWithScores=False,
      showPlotMaskWithHighestScore=False)

  time_list.append(time.time()-start_time)
  print(f"Time taken for operation : {time_list[-1]}")
```

```
avg_time = sum(time_list)/len(image_paths)
print("Average time for inference :", avg_time, " s")
```

1.2.7 Solution for getting an error for top left corner coordinates

- Mount Google Drive to access files and sets up an output path for saving annotated images. It checks if the folder already exists and creates it if necessary

```
# prompt: mount google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
output_images_path = "/content/drive/MyDrive/annotated_boxes
    /"
```

```
try:
    os.makedirs(output_images_path)
except FileExistsError:
    print("folder already exists")
```

- Take an image path and various boolean flags to control the display of intermediate results. It processes the image, calculates its center point, generates masks and scores, selects the best mask, computes bounding box coordinates, annotates the image with the bounding box corners, and saves it.

```
    def get_box_coordinates_updated(image_path:str,
        showOriginalImage=False, showPoints=True,
        showMasksWithScores=True, showPlotMaskWithHighestScore=
        True):
      img_name = image_path.split("/")[-1]
      # load image
      img = cv2.imread(image_path)
      img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

      # plot original image
      if showOriginalImage:
        plot_image(img)

      # get image dimensions
      img_height, img_width, _ = img.shape

      # get centre point coordinates
      center_point_coords = [ int(img_width/2), int(img_height
          /2)]
      input_point = np.array([center_point_coords])
      input_label = np.array([1])

      if showPoints:
        plt.figure(figsize=(10,10))
        plt.imshow(img)
        show_points(input_point, input_label, plt.gca())
        plt.axis('on')
        plt.show()


      # draw a rectangle that makes the center point of image

      # generate masks in the relevant area
      predictor.set_image(img)

      img_masks, img_masks_scores, img_masks_logits = predictor.
          predict(
        point_coords=input_point,
        point_labels=input_label,
        multimask_output=True,
      )

      if showMasksWithScores:
        for i, (mask, score) in enumerate(zip(img_masks,
            img_masks_scores)):
          plot_mask_with_score(img, i+1, mask, score,
              input_point, input_label)
```

```
    # get the mask with the highest score
    img_mask_with_highest_score, img_mask_highest_score,
        img_mask_logits_for_highest_score =
        get_mask_with_highest_score(img_masks, img_masks_scores
        , img_masks_logits)

    if showPlotMaskWithHighestScore:
      plot_square(img_mask_with_highest_score)

    # get the rectangular boxes
    img_top_left, img_top_right, img_bottom_left,
        img_bottom_right = get_bounding_box_coordinates_updated
        (img_mask_with_highest_score)

    # get the coordinates of the rectangular bounding box

    annotate_square_corners(img, img_top_left, img_top_right,
        img_bottom_left, img_bottom_right, output_images_path+
        img_name)
```

- Iterate over a list of image paths, calling the `get_box_coordinates_updated` function with parameters to suppress the display of intermediate results. It calculates the average time taken for inference across all images with these updated parameters, aiming to evaluate the performance impact of suppressing intermediate result display during image processing.

```
get_box_coordinates_updated(image_paths[0],showOriginalImage
    =False, showPoints=False, showMasksWithScores=False,
    showPlotMaskWithHighestScore=False)

updated_time_list = []

for img_path in image_paths:
  # start_time = time.time()
  get_box_coordinates_updated(img_path,showOriginalImage=
      False, showPoints=False, showMasksWithScores=False,
      showPlotMaskWithHighestScore=False)

  # updated_time_list.append(time.time()-start_time)
  # print(f"Time taken for operation : {time_list[-1]}")

updated_avg_time = sum(updated_time_list)/len(image_paths)
print("Average time for inference :", updated_avg_time, " s"
    )
```

1.3 Fast SAM model

1.3.1 Environment setup

- Define the environment characteristics to download the Fast SAM model. If the google colab environment is used, set COLAB variable to true.

```
import os
import gdown
```

```
import shutil

###############
COLAB = False  # True if Google Colab
###############
####################
INITIALIZED = True  # False if running for the first time
####################
####################
TESTING_SAM = True  # WIP: True if testing SAM, warn!:
    processing time and data usage
####################

# Other Configurations
########################
DETAILED_LOGS = True  # True for detailed logs
DEFAULT_COLORS = False  # True for not overriding your colors
RESET_FASTSAM = False  # True to reset FastSAM repo
RESET_DATASET = False  # True to reset box_train dataset
RESET_WEIGHTS = False  # True to reset weights
########################


def log(*args):
    if DETAILED_LOGS:
        print("log:", *args)


if COLAB:
    if not os.system("pip install yachalk"):
        log("yachalk installed")
else:
    if not os.system("pip install -r requirements.txt"):
        log("local pip requirements installed")

from yachalk import chalk as c  # noqa: E402

c.enable_full_colors()

by = c.bold.yellow_bright if DEFAULT_COLORS else c.bold.hex("#
    f0deb2")
br = c.bold.red_bright if DEFAULT_COLORS else c.bold.hex("#
    df8ca4")
bg = c.bold.green_bright if DEFAULT_COLORS else c.bold.hex("#
    abd6a0")
bb = c.bold.blue_bright if DEFAULT_COLORS else c.bold.hex("#92
    b3f4")
bm = c.bold.magenta_bright if DEFAULT_COLORS else c.bold.hex("#
    dcb6d6")
bc = c.bold.cyan_bright if DEFAULT_COLORS else c.bold.hex("#9
    cdae9")
bp = c.bold.hex("#c5a7f2")

print(by("Google Colab") if COLAB else bm("Local"), "Environment
```

```
    Configured")
ROOT = os.getcwd() if not COLAB else "/content"
print(bc("Working Directory:"), ROOT)
```

- Download the Fast SAM model. Fast SAM is a YOLOv8-seg based fast segmentation model.

```
# download FastSAM
if RESET_FASTSAM or not os.path.exists("lib/FastSAM"):
    if os.path.exists("lib/FastSAM"):
        shutil.rmtree("lib/FastSAM", ignore_errors=False)
    if not os.system(
        "git clone https://github.com/CASIA-IVA-Lab/FastSAM.git
           lib/FastSAM"
    ):
        log(bg("FastSAM repository cloned"))
else:
    print("skipping: FastSAM already downloaded")

if TESTING_SAM:
    log(bg("installing SAM dependencies"))
    if not os.system("pip install onnx onnxruntime"):
        log(bg("onnx and onnxruntime installed"))
    if not os.system(
        "pip install 'git+https://github.com/facebookresearch/
           segment-anything.git'"
    ):
        log(bg("segment-anything installed"))

if COLAB:
    if not os.system("pip install segment-anything-fast"):
        log(bg("segment-anything-fast installed"))
if not INITIALIZED:
    if not os.system("pip install -r lib/FastSAM/requirements.
       txt"):
        log(bg("FastSAM requirements installed"))
else:
    print(by("skipping: dependencies already installed"))
```

- Import python libraries such as pytorch, numpy and matplotlib to train, evaluate and visualize the model. Use the CUDA environment if available.

```
import cv2
import numpy as np
import torch
import torchvision
from matplotlib import patches, pyplot as plt
from random import randint


print(f"{by('PyTorch')} v{torch.__version__}")
print(f"{by('Torchvision')} v{torchvision.__version__}")
print(f"{by('OpenCV')} v{cv2.__version__}")
CUDA = torch.cuda.is_available()
MPS = torch.backends.mps.is_available()
```

```
print(f"{bg('CUDA')} Available: {bg(CUDA) if CUDA else br(CUDA)}
    ")
print(f"{bb('MPS')} Available: {bg(MPS) if MPS else br(MPS)}")

# TODO: check for CUDA and MPS and handle device accordingly for
    SAM and FastSAM
device = "cuda" if CUDA else "cpu"
```

1.3.2 Test dataset setup

- Download the custom box data set from the provided Google Drive URL. This contains both box images and masks.

```
if not os.path.exists("train") or RESET_DATASET:
    shutil.rmtree("train", ignore_errors=True)
    gdown.download(
        "https://drive.google.com/uc?id=1
           iWaDuDQKftRDZ_poWyEua7j6_leDhDQc",
        "box_train.zip",
        quiet=False,
    )
    os.system("unzip box_train.zip")
    os.remove("box_train.zip")
else:
    print(bg("skipping: dataset already downloaded"))

DATASET_PATH = f"{ROOT}/train"

if not os.path.exists(f"{ROOT}/annotate"):
    os.mkdir(f"{ROOT}/annotate")
```

1.3.3 Pre trained weights setup

- Download the FastSAM pretrained weights. Two model versions of the model are available with different sizes.FastSAM will be referred as FastSAM-x and FastSAM-s.

```
if RESET_WEIGHTS:
    if os.path.exists("weights/FastSAM-s.pt"):
        os.remove("weights/FastSAM-s.pt")
    if os.path.exists("weights/FastSAM-x.pt"):
        os.remove("weights/FastSAM-x.pt")
    if os.path.exists("weights/sam_vit_h_4b8939.pt"):
        os.remove("weights/sam_vit_h_4b8939.pt")
if not os.path.exists(f"{ROOT}/weights"):
    os.mkdir(f"{ROOT}/weights")
if not os.path.exists("weights/FastSAM-s.pt"):
    gdown.download(
        "https://drive.google.com/uc?id=10
           XmSj6mmpmRb8NhXbtiuO9cTTBwR_9SV",
        "weights/FastSAM-s.pt",
        quiet=False,
    )
if not os.path.exists("weights/FastSAM-x.pt"):
    gdown.download(
```

```
        "https://drive.google.com/uc?id=1m1sjY4ihXBU1fZXdQ-Xdj-
           mDltW-2Rqv",
        "weights/FastSAM-x.pt",
        quiet=False,
    )
print(bg("FastSAM Weights Configured"))

if TESTING_SAM
    if not os.path.exists("weights/sam_vit_h_4b8939.pt"):
        print(br("Downloading SAM weights (this may take a while
           ...)"))
        if not os.system(
            "wget https://dl.fbaipublicfiles.com/
               segment_anything/sam_vit_h_4b8939.pth -P weights/
               "
        ):
            print(bg("SAM weights downloaded"))
    else:
        print(by("skipping: SAM weights already downloaded"))
```

- Load the SAM models to the device.

```
from fastsam import FastSAM, FastSAMPrompt

model_fastsam_x = FastSAM(f"{ROOT}/weights/FastSAM-x.pt")
model_fastsam_s = FastSAM(f"{ROOT}/weights/FastSAM-s.pt")

if TESTING_SAM:
    from segment_anything import sam_model_registry,
       SamPredictor

    model_sam = SamPredictor(
        sam_model_registry["vit_h"](f"{ROOT}/weights/
           sam_vit_h_4b8939.pth").to(
            device=device
        )
    )
    print(by("SAM ViT model loaded"))
print(by("FastSAM models loaded and ready for action"))
```

1.3.4 Helper Functions

- Define the hyper-parameters and identifiers.

```
DETAILED_LOGS = True
TARGET_SIZE = 320 if TESTING_SAM else 960
FASTSAM_X = 0
FASTSAM_S = 1
SAM = 2  # WIP
```

- Define the basic utility functions to crop, resize, get the center coordinates and define the image paths.

```
ef crop_n_resize(img_path, target_size):
    """
    Crop and Resize the image to a square image of target_size
        using OpenCV
    """
    img = cv2.imread(img_path)
    h, w = img.shape[:2]
    min_dim = min(h, w)
    x = (w - min_dim) // 2
    y = (h - min_dim) // 2
    cropped = img[y : y + min_dim, x : x + min_dim]
    resized = cv2.resize(cropped, (target_size, target_size))
    if DETAILED_LOGS:
        h_new, w_new = resized.shape[:2]
        log(f"{w}x{h} -> {w_new}x{h_new}")
        # fig, ax = plt.subplots(1, 2, figsize=(10, 10))
        # ax[0].imshow(img)
        # ax[1].imshow(resized)
        # plt.show()
    return resized


def get_center_coords(image):
    """
    Get the center coordinates of the image from an OpenCV image
        object
    """
    h, w = image.shape[:2]
    center_coords = [int(w / 2), int(h / 2)]
    log(bb("Getting Center Coordinates..."))
    log(f"Image resolution: {w}x{h}")
    log(f"Center Coordinates: {center_coords}")
    return np.array([center_coords])


def list_img_paths(directory):
    """
    Returns a list of paths for all image files in a directory
    """
    img_ext = [".jpg", ".jpeg", ".png", ".gif", ".bmp"]
    img_files = []
    for root, dirs, files in os.walk(directory):
        for file in files:
            if any(file.lower().endswith(ext) for ext in img_ext
                ):
                img_files.append(os.path.join(root, file))
    return img_files
```

- Define the plotting helper functions and main functions.

```
def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array
            ([0.6])], axis=0)
```

```
    else:
        color = np.array([30 / 255, 144 / 255, 255 / 255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)


def show_points(coords, labels, ax, marker_size=375):
    pos_points = coords[labels == 1]
    neg_points = coords[labels == 0]
    ax.scatter(
        pos_points[:, 0],
        pos_points[:, 1],
        color="green",
        marker="*",
        s=marker_size,
        edgecolor="white",
        linewidth=1.25,
    )
    ax.scatter(
        neg_points[:, 0],
        neg_points[:, 1],
        color="red",
        marker="*",
        s=marker_size,
        edgecolor="white",
        linewidth=1.25,
    )


def show_box(box, ax):
    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    ax.add_patch(
        plt.Rectangle((x0, y0), w, h, edgecolor="green",
            facecolor=(0, 0, 0, 0), lw=2)
    )

def plot_image(image):
    plt.figure(figsize=(10, 10))
    plt.imshow(image)
    plt.axis("on")
    plt.show()


def plot_images(image1, image2, image3, image4):
    fig, ax = plt.subplots(2, 2, figsize=(10, 10))
    ax[0, 0].imshow(image1)
    ax[0, 0].axis("on")
    ax[0, 1].imshow(image2)
    ax[0, 1].axis("on")
    ax[1, 0].imshow(image3)
    ax[1, 0].axis("on")
    ax[1, 1].imshow(image4)
```

```
    ax[1, 1].axis("on")
    plt.show()


def plot_images_with_center_coords(image1, image2, image3,
    image4):
    fig, ax = plt.subplots(2, 2, figsize=(10, 10))
    ax[0, 0].imshow(image1)
    ax[0, 0].axis("on")
    show_points(*get_center_coords(image1), 1, ax[0, 0])
    ax[0, 1].imshow(image2)
    ax[0, 1].axis("on")
    show_points(*get_center_coords(image2), 1, ax[0, 1])
    ax[1, 0].imshow(image3)
    ax[1, 0].axis("on")
    show_points(*get_center_coords(image3), 1, ax[1, 0])
    ax[1, 1].imshow(image4)
    ax[1, 1].axis("on")
    show_points(*get_center_coords(image4), 1, ax[1, 1])
    plt.show()
```

- Define the model interface functions. These functions will help to generate masks, get the masks with highest score and plot the masks.

```
def fastsam_maskgen(
    model, image, device=device, retina_masks=True, imgsz=
        TARGET_SIZE, conf=0.4, iou=0.9
):
    """
    utility function to easily generate masks from FastSAM
        models
    """
    if model == FASTSAM_X:
        return FastSAMPrompt(
            image,
            model_fastsam_x(
                image,
                device=device,
                retina_masks=retina_masks,
                imgsz=imgsz,
                conf=conf,
                iou=iou,
            ),
            device=device,
        ).point_prompt(points=get_center_coords(image),
            pointlabel=[1])[0]
    elif model == FASTSAM_S:
        return FastSAMPrompt(
            image,
            model_fastsam_s(
                image,
                device=device,
                retina_masks=retina_masks,
                imgsz=imgsz,
```

```
                conf=conf,
                iou=iou,
            ),
            device=device,
        ).point_prompt(points=get_center_coords(image),
           pointlabel=[1])[0]
    else:
        return "Invalid model"


def get_mask_with_highest_score(masks, scores, logits):
    # Ensure that masks, scores, and logits are numpy arrays
    masks = np.array(masks)
    scores = np.array(scores)
    logits = np.array(logits)

    # Find the index of the mask with the highest score
    max_score_index = np.argmax(scores)

    # Get the mask with the highest score
    mask_with_highest_score = masks[max_score_index]

    # Get the score and logits corresponding to the highest
       score
    highest_score = scores[max_score_index]
    logits_for_highest_score = logits[max_score_index]

    return mask_with_highest_score, highest_score,
       logits_for_highest_score


def plot_mask_with_score(image, title, mask, score, input_point,
    input_label):
    plt.figure(figsize=(10, 10))
    plt.imshow(image)
    show_mask(mask, plt.gca())
    show_points(input_point, input_label, plt.gca())
    plt.title(f"Mask {title}, Score: {score:.3f}", fontsize=18)
    plt.axis("off")
    plt.show()


def plot_images_with_mask(model, image1, image2, image3, image4)
   :
    fig, ax = plt.subplots(2, 2, figsize=(10, 10))
    ax[0, 0].imshow(image1)
    ax[0, 0].imshow(fastsam_maskgen(model, image1), alpha=0.5)
    ax[0, 1].imshow(image2)
    ax[0, 1].imshow(fastsam_maskgen(model, image2), alpha=0.5)
    ax[1, 0].imshow(image3)
    ax[1, 0].imshow(fastsam_maskgen(model, image3), alpha=0.5)
    ax[1, 1].imshow(image4)
    ax[1, 1].imshow(fastsam_maskgen(model, image4), alpha=0.5)
    plt.show()
```

```
def plot_diff_masks(image, m1, m2):
    fig, ax = plt.subplots(1, 2, figsize=(10, 10))
    ax[0].imshow(image)
    ax[0].imshow(fastsam_maskgen(m1, image), alpha=0.5)
    ax[0].set_title("FastSAM-x", fontsize=18)
    ax[0].axis("off")
    ax[1].imshow(image)
    ax[1].imshow(fastsam_maskgen(m2, image), alpha=0.5)
    ax[1].set_title("FastSAM-s", fontsize=18)
    ax[1].axis("off")

    plt.show()


print(bg("Functions All Set! Ready to go!"))
```

1.3.5 Verification of the functions and the dataset

- This code snippet will demonstrate the functions and data set, which must be ensured before evaluating the models.

```
# getting all image paths in the box_train dataset
img_paths = list_img_paths(DATASET_PATH)
IMG_COUNT = len(img_paths)
log(f"{bb(IMG_COUNT)} images found in the dataset")

# cropping and resizing all images to TARGET_SIZE and appending
   to img_arr
img_arr = []
for i, img_pth in enumerate(img_paths):
    img_arr.append(crop_n_resize(img_pth, TARGET_SIZE))
    img_arr[i] = cv2.cvtColor(img_arr[i], cv2.COLOR_BGR2RGB)
print(
    f"{bg(len(img_arr))} images cropped and resized to {br(
       TARGET_SIZE,'x', TARGET_SIZE)}"
)

# collection of 10 random guesses for the demo
WILD_GUESSES = np.random.randint(IMG_COUNT, size=10)
log(bg("10 Random Guesses Collected: "), WILD_GUESSES)

log(
    "testing",
    {bc(TARGET_SIZE, "x", TARGET_SIZE)},
    "images from",
    {bc(WILD_GUESSES[1], "to", WILD_GUESSES[1] + 4)},
)
plot_images_with_center_coords(*img_arr[WILD_GUESSES[1] :
   WILD_GUESSES[1] + 4])
```

1.3.6 Evaluating models

- Evaluating FastSAM-x

```
log(bg("FastSAM-x evaluation..."))
plot_images_with_mask(FASTSAM_X, *img_arr[WILD_GUESSES[1] :
    WILD_GUESSES[1] + 4])
```

- Evaluating FastSAM-s

```
log(bg("FastSAM-s evaluation..."))
plot_images_with_mask(FASTSAM_S, *img_arr[WILD_GUESSES[1] :
    WILD_GUESSES[1] + 4])
```

- Evaluating SAM:WIP

```
def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array
            ([0.6])], axis=0)
    else:
        color = np.array([30 / 255, 144 / 255, 255 / 255, 0.6])
    h, w = mask.shape[-2:]

    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)


def fastsam_maskgen(
    model, image, device=device, retina_masks=True, imgsz=
        TARGET_SIZE, conf=0.4, iou=0.9
):
    """
    utility function to easily generate masks from FastSAM
        models
    """
    if model == FASTSAM_X:
        return FastSAMPrompt(
            image,
            model_fastsam_x(
                image,
                device=device,
                retina_masks=retina_masks,
                imgsz=imgsz,
                conf=conf,
                iou=iou,
            ),
            device=device,
        ).point_prompt(points=get_center_coords(image),
            pointlabel=[1])[0]
    elif model == FASTSAM_S:
        return FastSAMPrompt(
            image,
            model_fastsam_s(
                image,
```

```
                device=device,
                retina_masks=retina_masks,
                imgsz=imgsz,
                conf=conf,
                iou=iou,
            ),
            device=device,
        ).point_prompt(points=get_center_coords(image),
           pointlabel=[1])[0]
    else:
        return "Invalid model"


def plot_images_with_mask(model, image1, image2, image3, image4)
    :
    fig, ax = plt.subplots(2, 2, figsize=(10, 10))
    ax[0, 0].imshow(image1)
    ax[0, 0].imshow(fastsam_maskgen(model, image1), alpha=0.5)
    ax[0, 1].imshow(image2)
    ax[0, 1].imshow(fastsam_maskgen(model, image2), alpha=0.5)
    ax[1, 0].imshow(image3)
    ax[1, 0].imshow(fastsam_maskgen(model, image3), alpha=0.5)
    ax[1, 1].imshow(image4)
    ax[1, 1].imshow(fastsam_maskgen(model, image4), alpha=0.5)
    plt.show()
```

- Test the SAM model by choosing a random image and generating the mask. Set the TESTING_SAM variable to True

```
if TESTING_SAM:
    testing_img = img_arr[WILD_GUESSES[1]]
    center_coords = get_center_coords(testing_img)
    log(testing_img.shape)
    log(bg("SAM evaluation..."))

    model_sam.set_image(testing_img)
    masks, scores, logits = model_sam.predict(
        point_coords=center_coords,
        point_labels=np.array([1]),
        multimask_output=True,
    )
    # plot the SAM mask over the image without using user
       defined functions
    fig, ax = plt.subplots(1, 1, figsize=(10, 10))
    ax.imshow(testing_img)
    show_mask(masks[0], ax)
    show_points(center_coords, np.array([1]), ax)
    plt.show()

    fastsam_mask = FastSAMPrompt(
        testing_img,
        model_fastsam_s(
            testing_img,
            device=device,
```

```
            retina_masks=True,
            imgsz=TARGET_SIZE,
            conf=0.4,
            iou=0.9,
        ),
        device=device,
    ).point_prompt(points=center_coords, pointlabel=[1])

    print(masks.shape)  # (number_of_masks) x H x W
    print(fastsam_mask.shape)
```

1.3.7 Coordinate detection

- Plot the square around the box

```
wild_guess = randint(0, len(img_arr) - 1)
sample_image = img_arr[wild_guess]
sample_mask = fastsam_maskgen(1, sample_image)

if DETAILED_LOGS:
    print(f"Random Image: {wild_guess}")
    print(f"Image Array Element Type: {type(sample_image)}")
    print(f"Image Array Element Shape: {sample_image.shape}")
    print(f"Sample Mask Type: {type(sample_mask)}")
    print(f"Sample Mask Shape: {sample_mask.shape}")

def plot_square(mask):
    # find the indices where the mask is true
    true_indices = np.argwhere(mask)
    # get the bounding box of the true region
    top_l = np.min(true_indices, axis=0)
    bot_r = np.max(true_indices, axis=0)
    # calculate the width and height of the bounding box
    w = bot_r[1] - top_l[1]
    h = bot_r[0] - top_l[0]

    fig, ax = plt.subplots(1)
    # plot the mask
    ax.imshow(mask, cmap="gray")
    # create a rectangle patch
    rect = patches.Rectangle(
        (top_l[1], top_l[0]), w, h, linewidth=2, edgecolor="r",
            facecolor="none"
    )
    # add the rectangle patch to the axis
    ax.add_patch(rect)
    plt.show()

plot_square(sample_mask)
```

- Get bounding box coordinates

```
def get_bounding_box_coordinates(mask):
    # find the indices where the mask is true
```

```
    true_indices = np.argwhere(mask)

    # get the bounding box of the true region
    top_l = np.min(true_indices, axis=0)
    bot_r = np.max(true_indices, axis=0)

    # calculate the width and height of the bounding box
    w = bot_r[1] - top_l[1]
    h = bot_r[0] - top_l[0]

    # calculate the coordinates of the corners
    top_r = (top_l[1] + w, top_l[0])
    bot_l = (top_l[1], top_l[0] + h)
    bot_r = (top_l[1] + w, top_l[0] + h)
    top_l = (top_r[0] - (bot_r[0] - bot_l[0]), (top_r[1] - bot_r
        [1]) + bot_l[1])

    if DETAILED_LOGS:
        print("Bounding Box Coordinates")
        print("Top left:", top_l)
        print("Top right:", top_r)
        print("Bottom left:", bot_l)
        print("Bottom right:", bot_r)

    return [top_l, top_r, bot_l, bot_r]
bound_coords = get_bounding_box_coordinates(sample_mask)
```

- Annotate square corners

```
def annotate_square_corners(
    image, top_left, top_right, bottom_left, bottom_right,
        save_path
):
    # Create a figure and axis
    fig, ax = plt.subplots(1)

    # Plot the original image
    ax.imshow(image)

    # Annotate the corners
    ax.plot(top_left[0], top_left[1], "ro")  # Top Left corner
    ax.plot(top_right[0], top_right[1], "go")  # Top Right
        corner
    ax.plot(bottom_left[0], bottom_left[1], "bo")  # Bottom Left
         corner
    ax.plot(bottom_right[0], bottom_right[1], "yo")  # Bottom
        Right corner

    # Save the plot to the specified path
    plt.savefig(save_path)
    plt.show()


annotate_square_corners(
```

```
    sample_image,
    *get_bounding_box_coordinates(sample_mask),
    f"{ROOT}/annotate/box_img_{wild_guess}.jpg",
)
```

1.3.8 Pipeline WIP

- Getting the box coordinates from fastsam

```
    def get_box_coordinates_fast_sam(
        image_path: str,
        model_path: str,
        isFastSAM=True,
        showOriginalImage=False,
        showPoints=True,
        showPlotMaskWithHighestScore=True,
    ):
        # load image
        img = cv2.imread(image_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # plot original image
        if showOriginalImage:
            plot_image(img)

        # get image dimensions
        img_height, img_width, _ = img.shape

        # get centre point coordinates
        center_point_coords = [int(img_width / 2), int(
           img_height / 2)]
        input_point = np.array([center_point_coords])
        input_label = np.array([1])

        if showPoints:
            plt.figure(figsize=(10, 10))
            plt.imshow(img)
            show_points(input_point, input_label, plt.gca())
            plt.axis("on")
            plt.show()

        # draw a rectangle that makes the center point of image

        # generate the mask in the relevant area
        if isFastSAM:
            model_fast_sam = FastSAM(model_path)
            fast_sam_predictor = model_fast_sam(
                img,
                device=device,
                retina_masks=True,
                imgsz=img_width,
                conf=0.4,
                iou=0.9,
            )
```

```
            fast_sam_prompt_process = FastSAMPrompt(img,
                fast_sam_predictor, device=device)

            # point prompt
            # points default [[0,0]] [[x1,y1],[x2,y2]]
            # point_label default [0] [1,0] 0:background, 1:
                foreground
            img_mask = fast_sam_prompt_process.point_prompt(
                points=input_point, pointlabel=input_label
            )

            # plot_mask_with_score(img, "FastSAM output",
                img_mask, input_point, input_label )

            # reshape image mask
            # print(img_mask.shape)
            img_mask = np.transpose(img_mask, (1, 2, 0))
            # print(img_mask.shape)

        if showPlotMaskWithHighestScore:
            plot_square(img_mask)

        # get the rectangular boxes
        img_top_left, img_top_right, img_bottom_left,
            img_bottom_right = (
            get_bounding_box_coordinates(img_mask)
        )

        # get the coordinates of the rectangular bounding box
        annotate_square_corners(
            img,
            img_top_left,
            img_top_right,
            img_bottom_left,
            img_bottom_right,
            "/content/annotated_box_image.jpg",
        )
```

```
        time_list = []

        # test on one image
        import time

        start_time = time.time()
        get_box_coordinates_fast_sam(
            img_paths[5],
            model_path=fast_sam_x_checkpoint,
            isFastSAM=True,
            showOriginalImage=False,
            showPoints=False,
            showPlotMaskWithHighestScore=True,
        )

        time_list.append(time.time() - start_time)
```

```
        print(f"Time taken for operation : {time_list[-1]}")
```

1.3.9 Benchmarking WIP

- test

```
    # test on one image
    import time

    start_time = time.time()
    get_box_coordinates_fast_sam(
        img_paths[5],
        model_path=fast_sam_s_checkpoint,
        isFastSAM=True,
        showOriginalImage=False,
        showPoints=False,
        showPlotMaskWithHighestScore=False,
    )

    time_list.append(time.time() - start_time)
    print(f"Time taken for operation : {time_list[-1]}")
```

```
        time_list = []
```

```
        output_images_path = "/content/drive/MyDrive/
            annotated_boxes_fast_sam/"
```

```
    try:
        os.makedirs(output_images_path)
    except FileExistsError:
        print("folder already exists")
```

```
    def get_box_coordinates_updated(
        image_path: str,
        showOriginalImage=False,
        showPoints=True,
        showMasksWithScores=True,
        showPlotMaskWithHighestScore=True,
    ):
        img_name = image_path.split("/")[-1]
        # load image
        img = cv2.imread(image_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # plot original image
        if showOriginalImage:
            plot_image(img)

        # get image dimensions
        img_height, img_width, _ = img.shape

        # get centre point coordinates
```

```
        center_point_coords = [int(img_width / 2), int(
            img_height / 2)]
        input_point = np.array([center_point_coords])
        input_label = np.array([1])

        if showPoints:
            plt.figure(figsize=(10, 10))
            plt.imshow(img)
            show_points(input_point, input_label, plt.gca())
            plt.axis("on")
            plt.show()

        # draw a rectangle that makes the center point of image

        # generate masks in the relevant area
        predictor.set_image(img)

        img_masks, img_masks_scores, img_masks_logits =
            predictor.predict(
            point_coords=input_point,
            point_labels=input_label,
            multimask_output=True,
        )

        if showMasksWithScores:
            for i, (mask, score) in enumerate(zip(img_masks,
                img_masks_scores)):
                plot_mask_with_score(img, i + 1, mask, score,
                    input_point, input_label)

        # get the mask with the highest score
        (
            img_mask_with_highest_score,
            img_mask_highest_score,
            img_mask_logits_for_highest_score,
        ) = get_mask_with_highest_score(img_masks,
            img_masks_scores, img_masks_logits)

        if showPlotMaskWithHighestScore:
            plot_square(img_mask_with_highest_score)

        # get the rectangular boxes
        img_top_left, img_top_right, img_bottom_left,
            img_bottom_right = (
            get_bounding_box_coordinates(
                img_mask_with_highest_score)
        )

        # get the coordinates of the rectangular bounding box

        annotate_square_corners(
            img,
            img_top_left,
            img_top_right,
```

```
            img_bottom_left,
            img_bottom_right,
            output_images_path + img_name,
        )
```

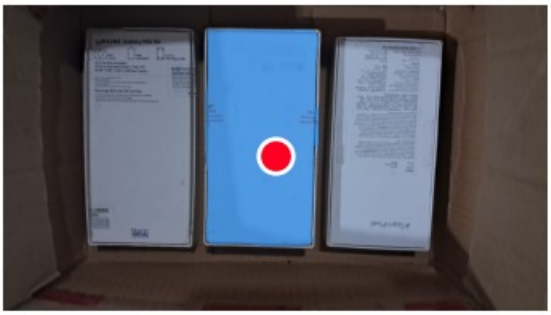
```
    get_box_coordinates_updated(
        img_paths[0],
        showOriginalImage=False,
        showPoints=False,
        showMasksWithScores=False,
        showPlotMaskWithHighestScore=False,
    )
```

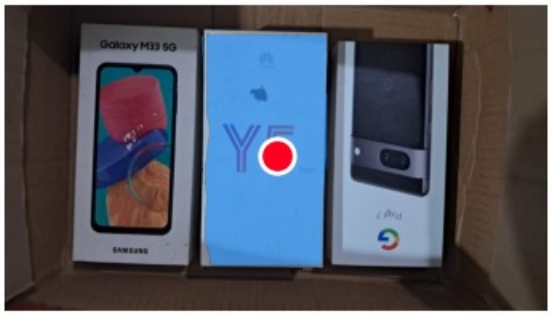
```
        updated_time_list = []
```

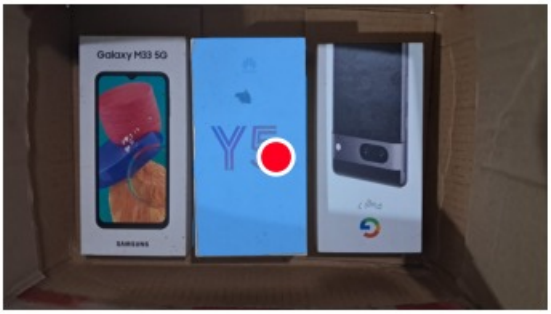
```
    for img_path in img_paths:
        # start_time = time.time()
        get_box_coordinates_updated(
            img_path,
            showOriginalImage=False,
            showPoints=False,
            showMasksWithScores=False,
            showPlotMaskWithHighestScore=False,
        )

        # updated_time_list.append(time.time()-start_time)
        # print(f"Time taken for operation : {time_list[-1]}")
```

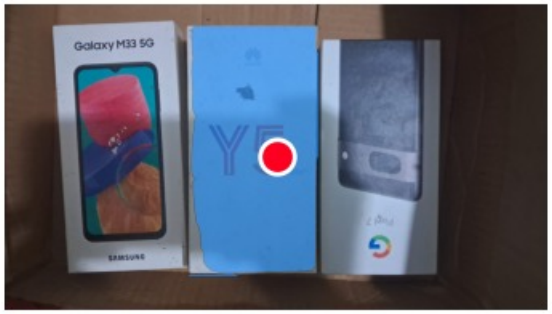
```
    updated_avg_time = sum(updated_time_list) / len(img_paths)
    print("Average time for inference :", updated_avg_time, " s"
       )
```

1.3.10 Final Results on Our Phone Boxes Setup










STM32
Note10


STM32
Development Tools
Note10


STM32
Development Tools

Chapter 2

Gripper Electromechanical Design Documentation

2.1 PCB Design

2.1.1 Introduction

In its current prototype state, the Bin Picking Robot Control Board is an important step toward the realization of an effective bin picking robot system. This PCB prototype, which is intended to give priority to coordinate display, pressure measurement, and gripper control, paves the way for improving productivity, streamlining bin selecting procedures, and guaranteeing accuracy when manipulating objects.

2.1.2 Design Specifications

1. **Power Supply**: The PCB design functions within a 12V to 24V DC power system to support a range of power input scenarios and ensure compatibility with conventional power sources.
2. **External Communication:**
    - **OLED Display Port:** Provides a connection for an OLED display module to be connected, allowing for visual feedback and status updates while the device is in use.
    - **Membrane Sensor Port:** Offers a specific interface for attaching a membrane sensor, enabling environmental sensing or human input.
    - **Pressure Sensor Port:** Has a connection to attach a pressure sensor, which enables the PCB to gauge and track the amount of pressure in the area where bins are picked.
    - **Stepper Motor Port:** This feature allows you to connect a stepper motor, which is necessary to drive the gripper mechanism precisely and controllably.
    - **UART Communication Ports (TX, RX):** Provides UART communication ports for data exchange and command execution when establishing serial connectivity with external devices, like a computer or host system.
3. **Power-On Switch**: This feature simplifies the starting procedure and improves user comfort by using a single power switch to turn on the PCB.
4. **Gripper Control Buttons (BTNs):** These buttons (BTNs) allow manual operation and item handling by manipulating the gripper mechanism.

2.1.3 Component Selection

Power Regulators

We have chosen three power regulators for our project to ensure e4icient power management. These include a switching voltage regulator and two low-dropout (LDO) voltage regulators. Each regulator was selected based on its specific features and suitability for our design needs, providing stable and reliable power to di4erent system parts.

Switching Voltage Regulator We have chosen this switching voltage regulator for its superior efficiency and suitability for our power supply requirements.

- Non-synchronous buck DC/DC converter
- Input range: 6 V to 100 V
- Output current: 2.5 A

LDO Voltage Regulators For low-dropout regulation, we selected two LDO voltage regulators for their specific attributes tailored to different parts of our project.

- LDO Voltage Regulator (3.3V, 150 mA)
  - Output Voltage: 3.3 V
  - Output Current: 150 mA
  - Key Attributes: Low noise, 1.5% tolerance, fixed output
- LDO Voltage Regulator (5V, 800 mA)
  - Output Voltage: 5 V
  - Output Current: 800 mA
  - Key Attributes: Higher current capacity, low dropout voltage

The combination of these voltage regulators provides a comprehensive solution for our power management needs, ensuring efficiency, stability, and precision across different sections of our project. The chosen components are well-suited for SMD/SMT mounting, making them compatible with our PCB design and assembly process.

Microcontroller: ATmega328P-AU

Our project has chosen the ATmega328P-AU microcontroller from Microchip's AVR family as the primary microcontroller. This decision is based on a thorough evaluation of its features and capabilities compared to other market microcontrollers.

- Simplicity: The ATmega328P-AU is built on the AVR architecture, known for its simplicity, ease of use, and extensive community support, particularly in educational projects. This MCU is well-documented, making it ideal for programming and development.
- Memory I/O: When examining the alternatives, the ATmega2560 has 86 pins, the STM has 37, and the ATmega328 has 23. Out of these options, the ATmega328P-AU oQers a well-balanced configuration with 32 KB of flash memory, 2 KB of SRAM, 1 KB of EEPROM, and 23 general-purpose I/O pins. These specifications align well with our project's requirements, ensuring suQicient resources without excessive unused pins or memory capacity.
- Clock Frequency and Power: The microcontroller supports a maximum clock frequency of 20 MHz and operates within a wide voltage range of 1.8V to 5.5V. Its relatively low power consumption makes it suitable for various applications while ensuring energy eQiciency.

Table 2.1: Microcontroller Comparison

Specification	**ATMEGA328P-AU**	**ATMEGA2560-16AU**	**STM32F103C8T6**
Manufacturer	Microchip Technology	Microchip Technology	STMicroelectronics
Core	AVR	AVR	ARM Cortex M3
Data RAM Size	2 kB	8 kB	20 kB
Program Memory Size	32 kB	256 kB	64 kB
Max. Clock Frequency	20 MHz	16 MHz	72 MHz
Number of I/Os	23 I/O	86 I/O	37 I/O

- Availability and Cost: The ATmega328P-AU is widely available and comes at a relatively low cost, making it a cost-eQective choice, especially for small-scale projects. Its availability in the market ensures scalability and ease of procurement.
- Communication Interfaces: ATmega328P-AU supports UART and I2C interfaces for seamless communication with external devices like sensors and displays, aligning perfectly with project requirements.

Based on thorough evaluation and comparison, the ATmega328P-AU was selected as the most suitable microcontroller for our project. While the STM32 is also a viable option, our team's extensive experience and familiarity with the ATmega328P-AU make it the ideal choice to achieve our project objectives eQectively.

Motor Controller: DRV8825

Chosen the DRV8825 motor controller after comparing it with others, as it is suitable for our stepper motor with a step angle of 1.80° and meets our project requirements effectively.

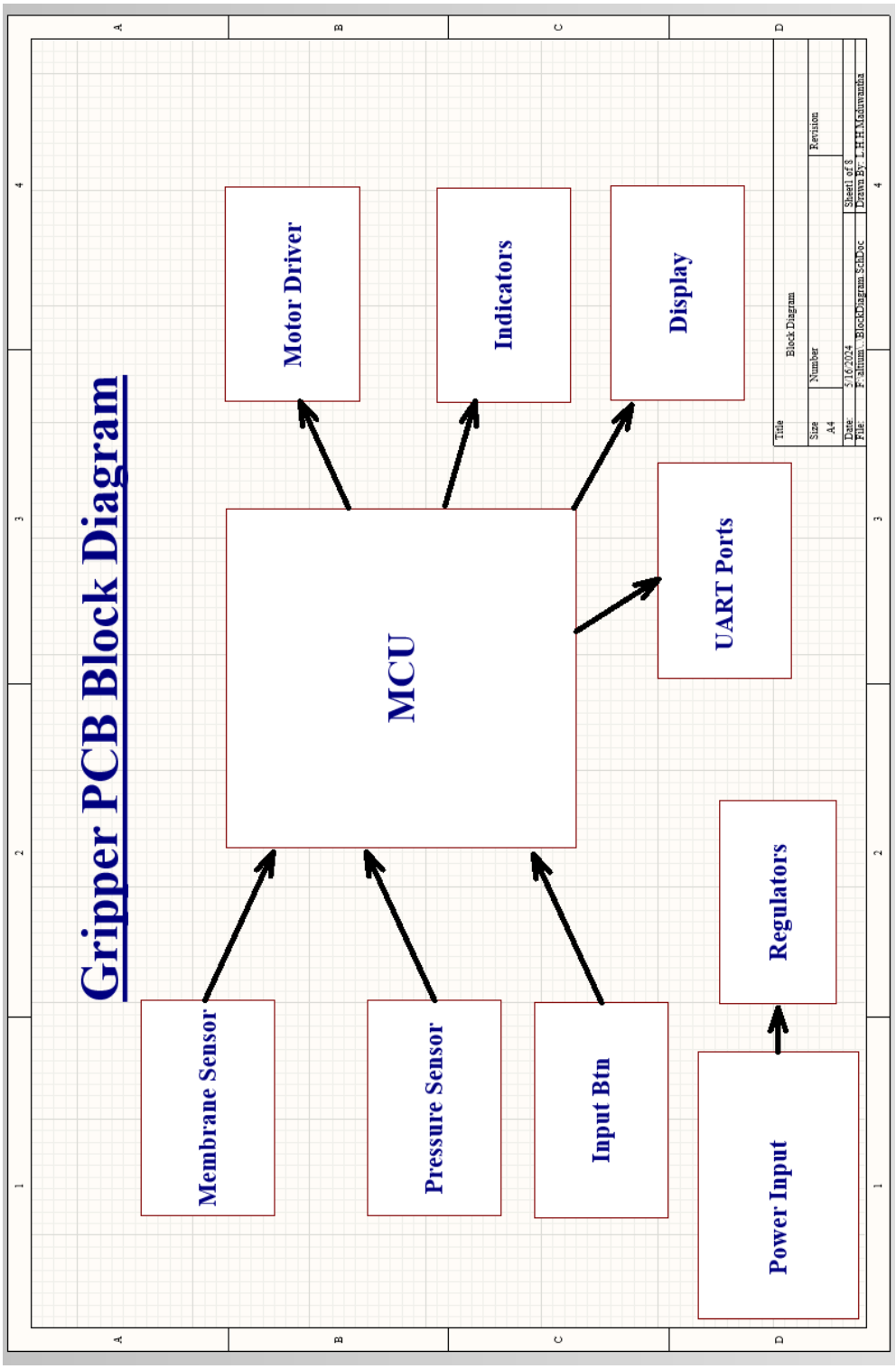
- Operating voltage range: The DRV8825 motor controller supports an operating voltage range of 8.2 V to 45 V, suitable for our 12 VDC stepper motor.
- Output current: It can handle an output current of 2.5 A, ensuring compatibility with our stepper motor's current rating of 1.30 A.
- Control : The DRV8825 offers integrated micro-stepping, allowing for precise control over step resolution and smoother motor movements. A reduction in the size of the motor steps using the micro-stepping technique can deliver smoother movement at low speeds, enhancing overall performance.

Table 2.2: Motor Controller Comparison

Specification	**TMC2208**	**A4988**	**DRV8825**
Operating Supply Voltage	4.6 V to 5.25 V	8 V to 35 V	8.2 V to 45 V
Output Current	2 A	2 A	2.5 A
Features	-	-	Integrated micro-stepping
Max. Operating Temperature	+125°C	+85°C	+85°C
Min. Operating Temperature	-40°C	-20°C	-40°C

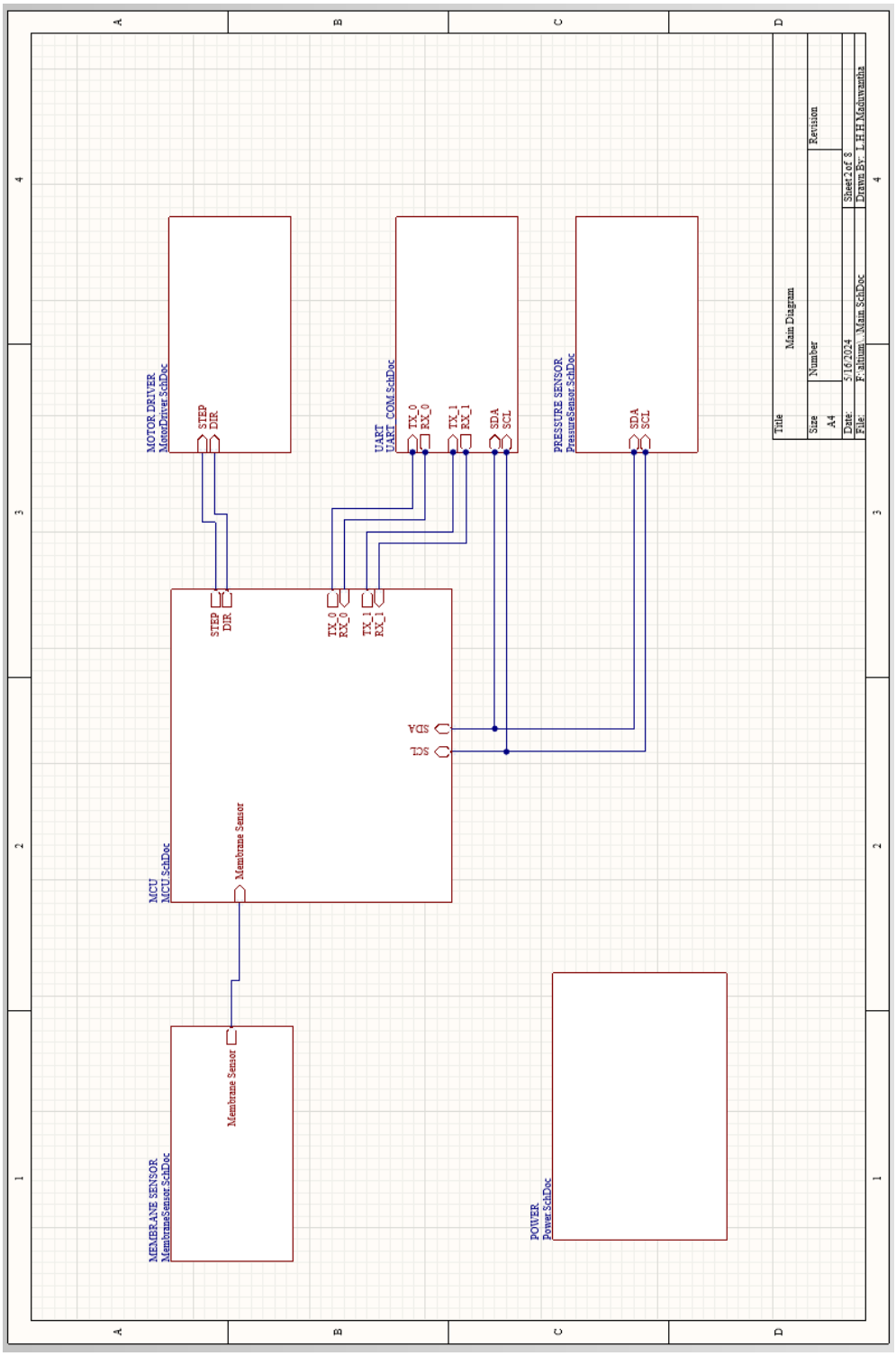
After thorough evaluation and comparison, the DRV8825 was selected as our project's most suitable motor driver. While the TMC2208-LA-T and A4988SETTR-T are also viable options, the TMC2208 package may present challenges during soldering onto the PCB. Moreover, the integrated micro-stepping feature of the DRV8825PWP, combined with its wider operating voltage range, makes it the ideal choice to eQectively achieve our project objectives

2.1.4 Block Diagram

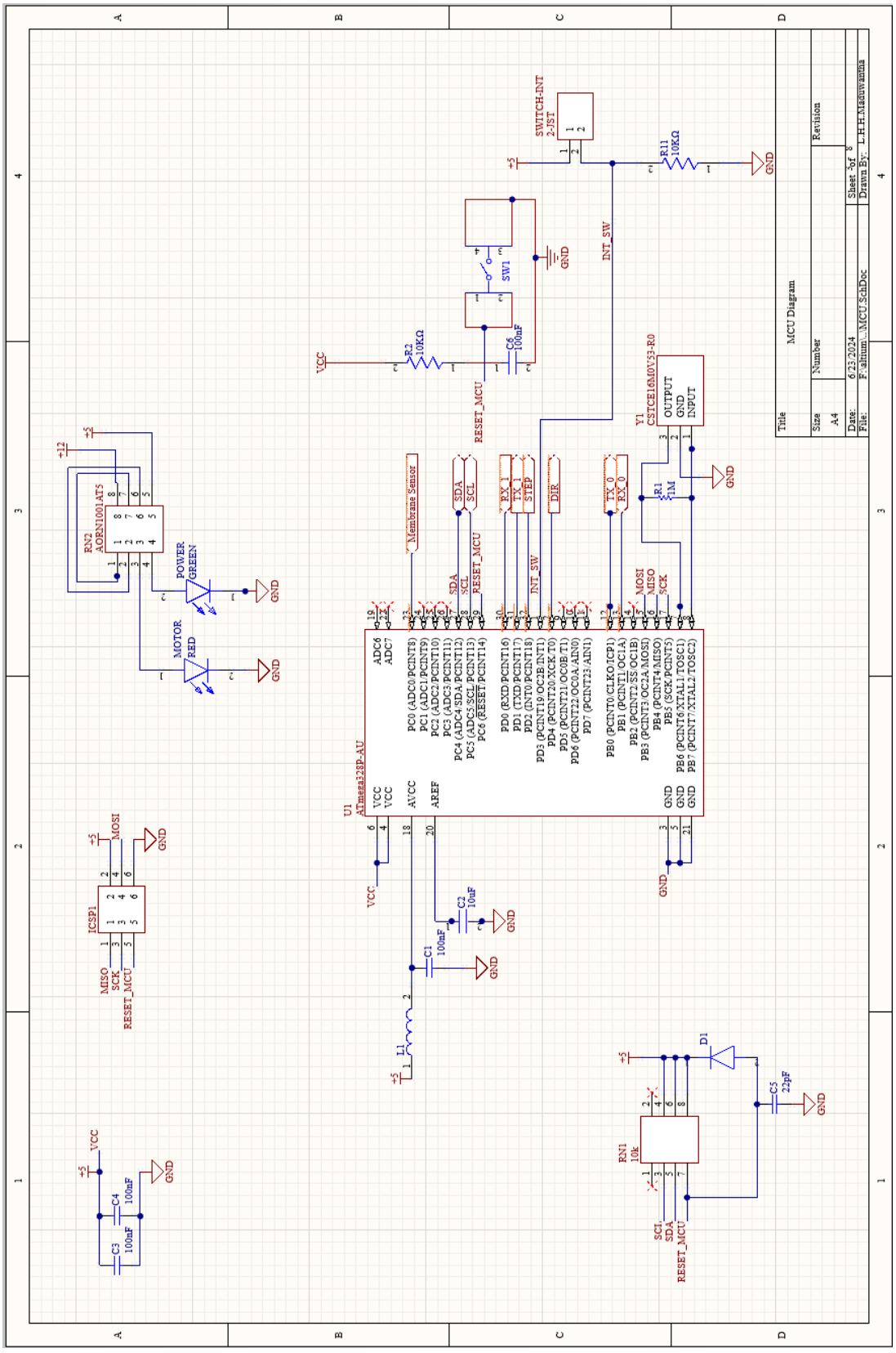


2.1.5 Schematic Diagrams

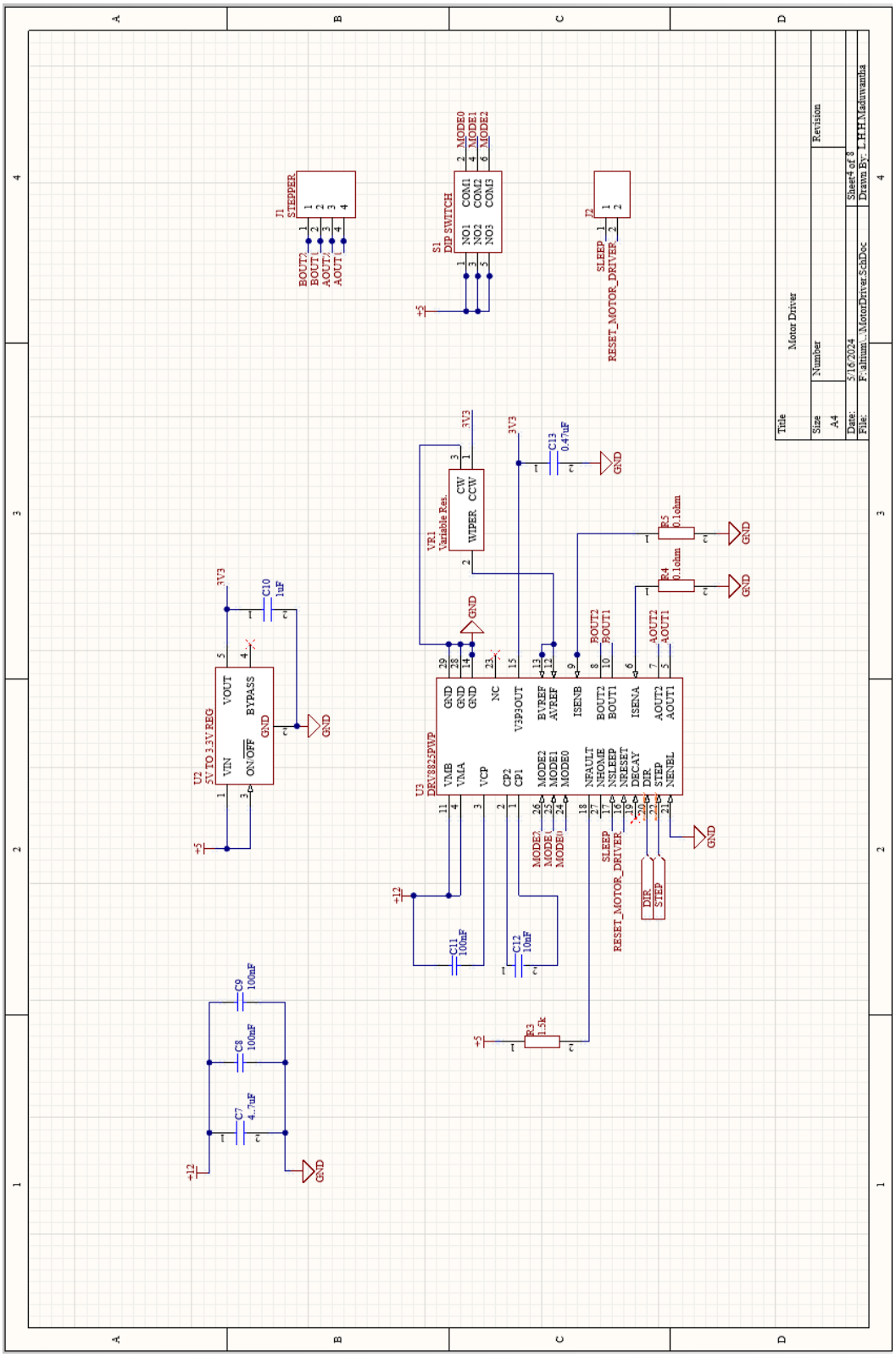
Overall Schematic Diagram



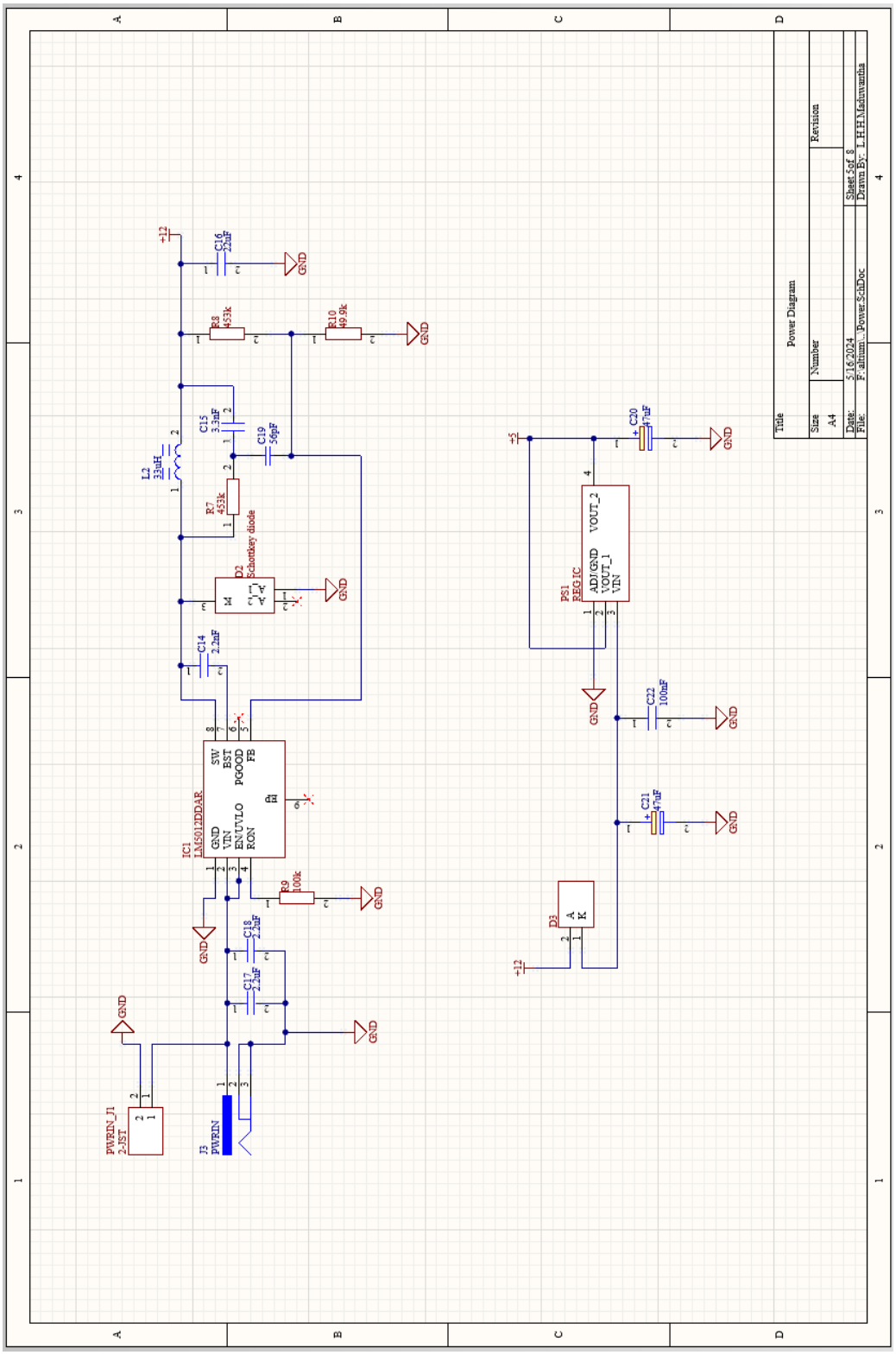
MCU Schematic Diagram



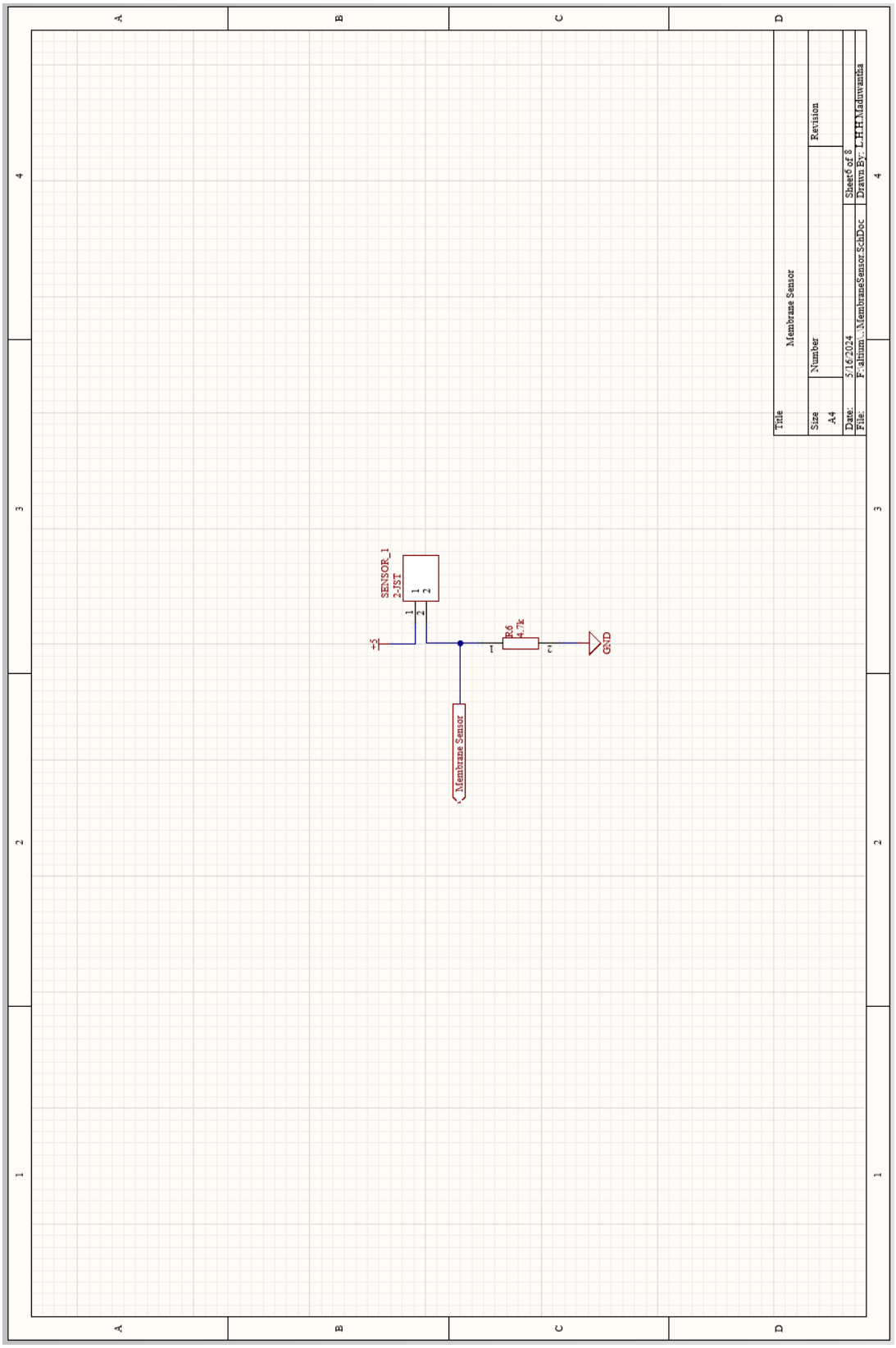
Motor Driver Schematic Diagram



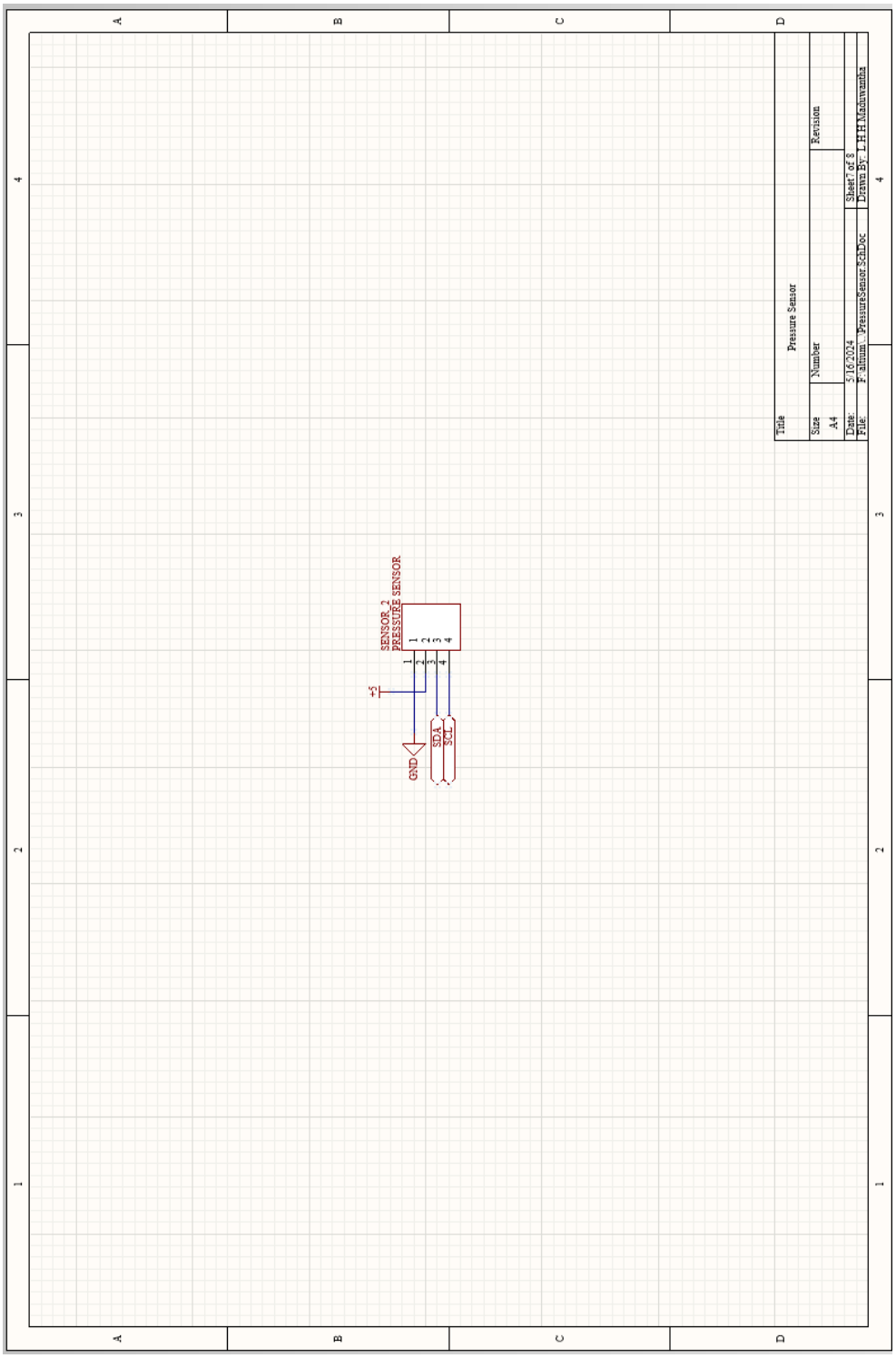
Power Schematic Diagram



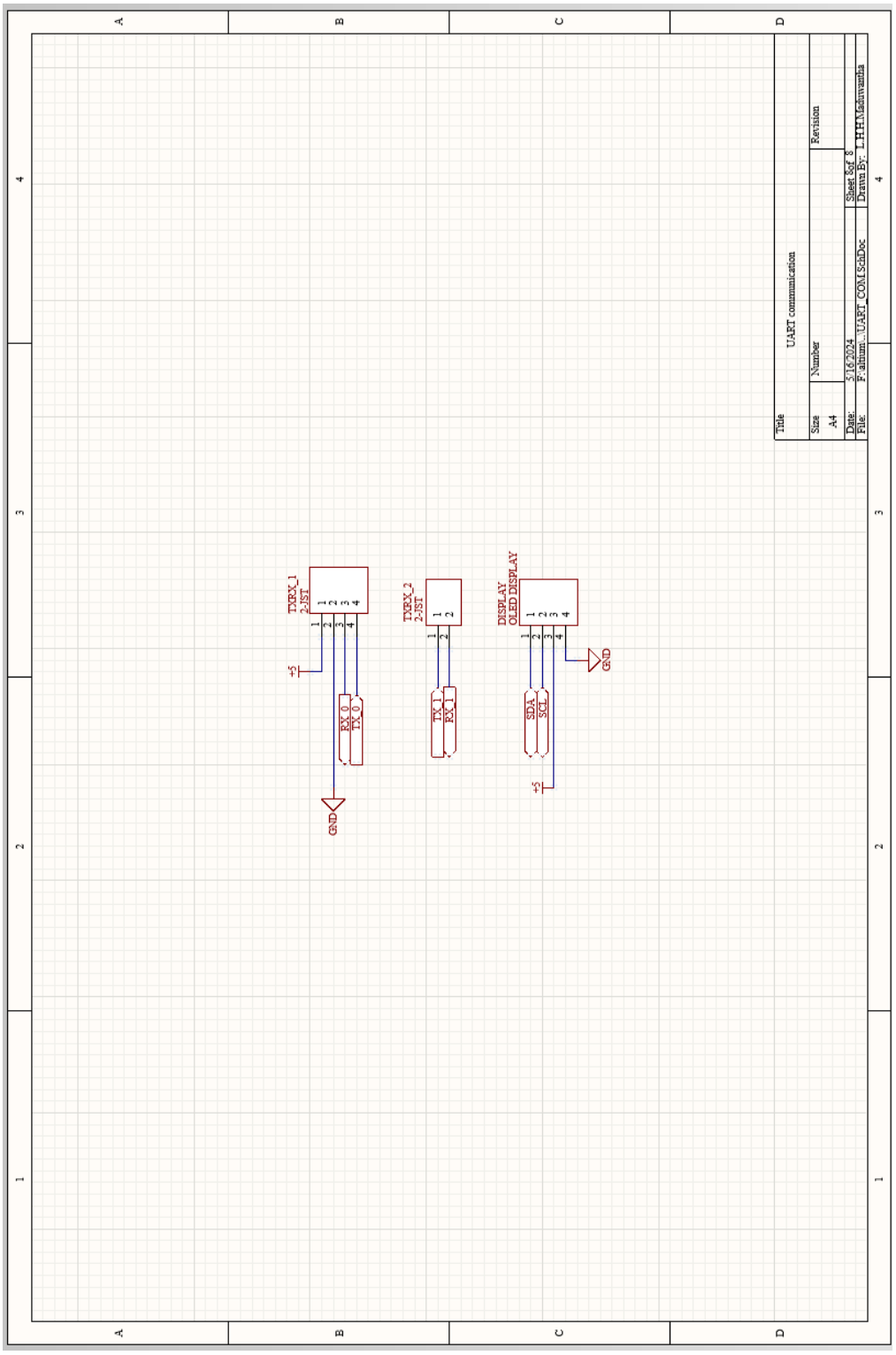
Membrane Sensor Ports Schematic



Pressure Sensor Ports Schematic

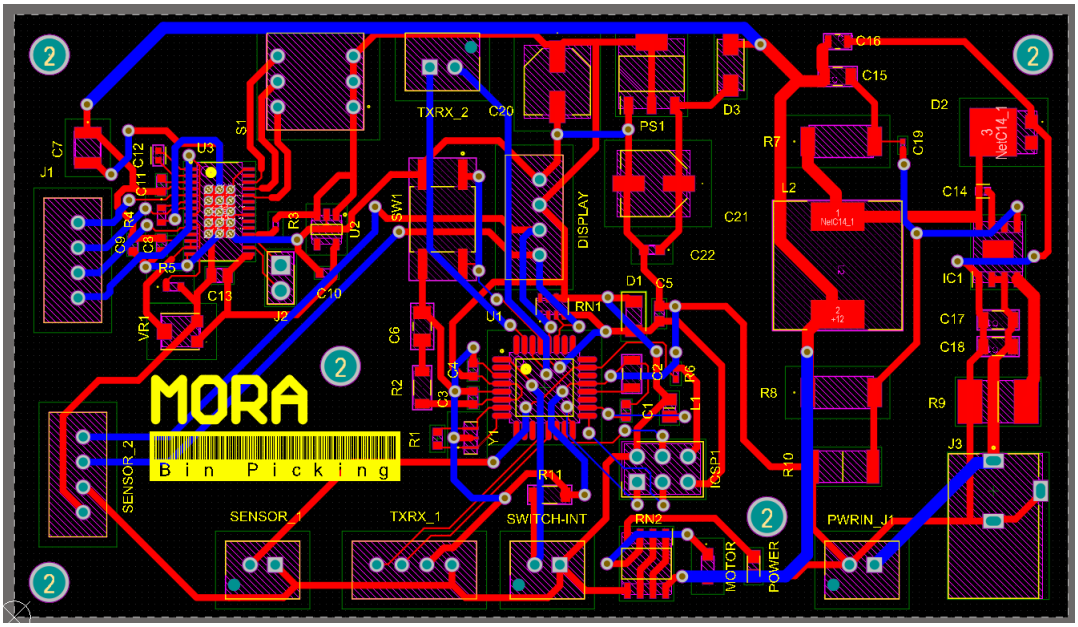


UART Communication Ports Schematic

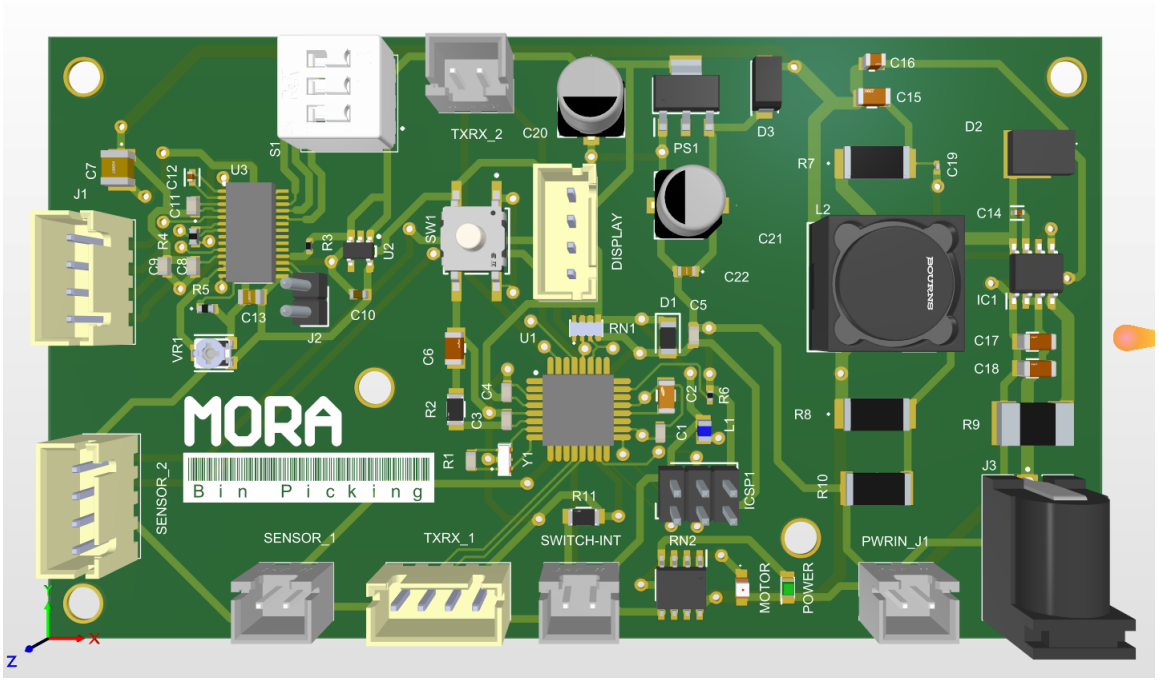


2.1.6 PCB Layout

PCB 2D Layout



PCB 3D Layout



2.1.7 Fabrication Details

- Base Material: FR-4
- Material Type: FR4-Standard TG 135-140
- Outer Copper Weight: 1 oz
- PCB Thickness: 1.6 mm
- Layers: 2
- Appearance Quality: Class 2 Standard
- Silkscreen Technology: Ink-jet/Screen Printing Silkscreen
- Board Outline Tolerance: ±0.2mm (Regular)

2.1.8 Additional Design Specifications

Routing Widths

- **High Current Nets:** Power lines, motor driver nets, and other high-current-carrying traces use 30 mil and 50 mil routing widths.
- **Data Lines:** Data lines use a 10 mil copper width.

Vias for Testing

Strategically placed vias throughout the PCB layout facilitate testing and debugging during the manufacturing and assembly process.

Voltage Regulation

- **12V Voltage Regulator:** Adjusts input voltage to 12V for stepper motors.
- **5V Voltage Regulator:** Steps down input voltage to 5V for components requiring a 5V supply.
- **3.3V Regulator:** Supplies voltage to the DRV8825 motor driver.

DIP Switch for Step Angle Adjustment

Incorporates a DIP switch for modifying the stepper motor's step angle, providing flexibility and control for the gripper mechanism.

2.1.9 PCB Testing

Net/Trace Name	From	To	Connection Status	Comments
VCC	Ps1	Mcu, Display, U2, Motor Driver, Icsp1, Rn2, Switch Int, Txrx1, Sensor1, Sensor2	Connected	Power
NetC17_1	Pwrin_J1, J3	Ic1	Connected	Power
+12V	L2	Motor Driver	Connected	Power
+3V3	U2	Motor Driver, VR1	Connected	Power
Gnd	Pwrin_J1	Power_Led, Motor Led, Mcu, Motor Driver, Sensor2, Display, Txrx1, Icsp1, Sw1, U2, Vr1	Connected	Power
SDA	Mcu	Display, Sensor1	Connected	I2C pins
SCL	Mcu	Display	Connected	I2C pins
Reset_Mcu	Mcu	Icsp1, Sw1	Connected	
Rx1	Mcu	Txrx2	Connected	UART pins
Tx1	Mcu	Txrx2	Connected	Output pins
Tx0	Mcu	Txrx2	Connected	Output pins
Rx0	Mcu	Txrx2	Connected	Output pins
MOSI	Mcu	Icsp1	Connected	UART pins
MISO	Mcu	Icsp1	Connected	UART pins
SCK	Mcu	Icsp1	Connected	UART pins
DIR	Mcu	Motor Driver	Not Connected	Motor driver step & dir pins
STEP	Mcu	Motor Driver	Not Connected	Motor driver step & dir pins
Netr1_1	Mcu	Crystal Oscillator_In		
Netr1_2	Mcu	Crystal Oscillator_Out	Connected	
Int_sw	Mcu	Switch_Int	Connected	Output btn
NetRN2_1	Rn2	Rn2	Not Connected	Resistor network
Netmotor_1	Rn2	Motor_Led	Connected	Motor Led
Netpower_1	Rn2	Power_Led	Connected	Power Led
Reset_md	Motor driver	J2	Connected	
Sleep	Motor driver	J2	Connected	
Mode0, Mode1, Mode2	Motor driver	S1	Connected	Stepper motor mode selection
Aout1, Bout2, Aout1, Bout2	Motor driver	J1	Connected	Stepper motor outputs
Aref, Bref	VR1	Motor Driver	Connected	Motor driver reference voltages
NetR6_1	MCU	Sensor_1	Connected	Membrane sensor

Appendix A

Daily Log

29 January - 4 February

- Discussing the main objective of the project.
- Identifying the segmentation and object detection architectures to explore (SegNet, DeepLab, UNet, CornerNet).
- Starting the fundamental research on each architecture.

5 February - 11 February

- Reading the documentation and GitHub repositories related to each architecture.
- We engaged in a group discussion to identify the main objectives of the project and our exposure to these objectives. Following this discussion, we compiled the ideas into a detailed self-evaluation report.

12 February - 28 February

- We successfully implemented the DeepLab architecture, and the UNet model showed promising results. However, we encountered significant challenges in implementing CornerNet and SegNet due to incomplete documentation.
- We explored alternatives like YOLO, ENet, and fully connected neural networks(FCN) and discussed their performance.
- We discussed the fundamentals of the electronic and PCB design approaches. We identified FPGA as a possible alternative.
- As a group we discussed the progress, next steps, stakeholders, and the user journeys. We compiled our findings into a detailed report.

29 February - 6 March

- We discussed the main components and requirements of the electronic and enclosure designs.
- UNet successfully segmented the standard car images. However, both Unet and DepLab struggled to segment the boxes. Thus, we identified the need to fine-tune those architectures with a custom box data set.

March 6 - March 12

- We started collecting box images from the internet. We made a data set of 400 box images with different lighting, orientations, and textures. A publically available RoboFlow data set was used to improve the diversity of the data set.

13 March - 30 March

- We created the segmentation masks for the box data set. UNet, FCN, and DeepLab were fine-tuned using this custom data set.
- We found the Vision Transformer(ViT) based segment anything(SAM) architecture to be an excellent alternative.

31 March - 8 April

- We customized SAM and FastSAM-s models to detect boxes.
- We discussed the possible designs for the gripper and possible electronic designs. We chose the best plausible design.

9 April - 14 April

- We thoroughly analyze the feasibility of each gripper design according to the flexibility, ruggedness, power efficiency, and electronic design.
- PCB designing process was started with the help and collective effort of all team members.

15 April - 21 April

- We completed designing the printed circuited board. After a thorough discussion, we corrected our mistakes and sent it for the manufacturing process.

22 April - 30 April

- We received the PCB. We assembled the components and did the preliminary tests.
- We started designing the enclosure using SolidWorks.

Appendix B

Other Neural Network Trials

B.1 DeepLab

B.1.1 Introduction

In this project, our aim was to develop a computer vision-based bin picking system. Through meticulous research, experimentation, we endeavored to engineer a solution which will be able to revolutionize efficiency and precision in industrial settings.

*Objective of our task Given an image of a box, the coordinates of the box should be returned.

B.1.2 Reviewing DeepLabV1 Research Paper - 1st of March 2023

paper title : **DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs**

Our project commenced with an in-depth review of the DeepLab research paper, a seminal work in the field of semantic image segmentation. DeepLab employs a fully convolutional neural network (FCN) architecture coupled with atrous convolution to capture object details at multiple scales. Atrous convolution enables the network to enlarge the field of view without increasing the number of parameters, thus preserving computational efficiency. By comprehensively understanding the principles elucidated in the DeepLab paper, we aimed to leverage its insights for our bin picking project.

Although there are many revisions of the same model, in general, DeepLab is a state-of-art deep learning system for semantic image segmentation built on top of **Caffe**.

It combines

1. atrous convolution to explicitly control the resolution at which feature responses are computed within Deep Convolutional Neural Networks
2. atrous spatial pyramid pooling to robustly segment objects at multiple scales with filters at multiple sampling rates and effective fields-of-views
3. densely connected conditional random fields (CRF) as post processing

B.1.3 Types of Segmentation

Before digging into details of our implementation, we have to understand different types of segmentation types.

1. **Thresholding**: Segments pixels based on intensity levels above or below a chosen threshold, dividing the image into binary regions.

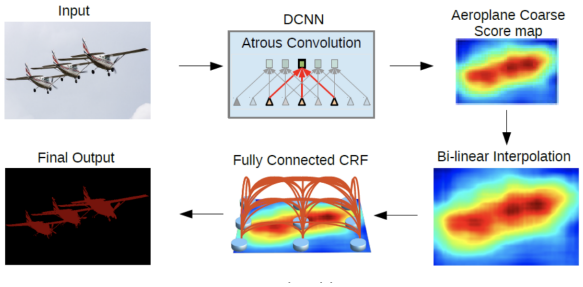


Figure B.1: data flow in generic DeepLab model

2. **Edge Detection**: Identifies boundaries between objects by detecting abrupt changes in intensity or color, often using techniques like Sobel or Canny edge detection.
3. **Semantic Segmentation**: Assigns semantic labels to each pixel, categorizing them into meaningful classes like objects or background, commonly employed in tasks like image understanding and scene parsing.
4. **Instance Segmentation**: Not only categorizes pixels into semantic classes but also distinguishes between individual object instances, crucial for applications requiring precise object identification and tracking.

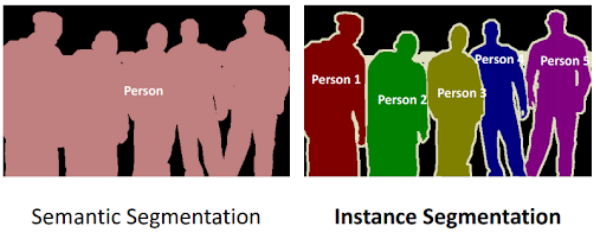


Figure B.2: Comparison between instance and semantic segmentation

B.1.4 Ambiguities with DeepLab models

There were several issues and ambiguities that we ran into. We were on a mission to find a pretrained model of DeepLab. Then we got to know there were three versions of DeepLab.

1. **Version 1**
   (a) paper : *Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs*
   (b) *model architecture with pretrained weights*
   (c) Performance : DeepLab currently achieves 73.9% on the challenging PASCAL VOC 2012 image segmentation task
2. **Version 2**
   (a) paper : *Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs*
   (b) *Official Code Repository*
   (c) *model architecture with pretrained weights*

(d) Performance : DeepLabv2 currently achieves 79.7% on the challenging PASCAL VOC 2012 semantic image segmentation task.

3. **Version 3**

(a) paper : *Rethinking Atrous Convolution for Semantic Image Segmentation*

(b) *model architecture with pretrained weights*

(c) Perormance :

(d) Datasets used

i. Cityscapes

ii. Pascal Context

iii. Pascal VOC 2012

Because there were many options to choose from we were in an ambiguity for which one to choose. Therefore, we checked the results in the benchmarks of all those models

	mean	aero plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	dining table	dog	horse	motor bike	person	potted plant	sheep	sofa	train	tv/ monitor	submission date
** SegNeXt ** [?]	**90.6**	**98.3**	**85.0**	**97.6**	**88.3**	**91.3**	97.5	91.4	98.3	60.4	96.7	**85.0**	95.7	**98.2**	94.2	92.7	82.5	97.3	77.7	93.1	84.3	19-Sep-2022
** EfficientNet-L2 + NAS-FPN + Noisy Student ** [?]	90.5	98.0	84.8	89.6	88.2	91.0	**98.3**	93.0	**98.5**	57.5	**98.4**	81.8	**98.4**	98.0	**95.8**	93.2	**83.2**	**97.8**	75.0	91.8	**90.0**	15-Jun-2020
DeepLabv3+_JFT [?]	89.0	97.5	77.9	96.2	80.4	90.8	**98.3**	**95.5**	97.6	58.8	96.1	79.2	95.0	97.3	94.1	**93.8**	78.5	95.5	74.4	93.8	81.6	09-Feb-2018
** RecoNet152_coco ** [?]	89.0	97.3	80.4	96.5	83.8	89.5	97.6	95.4	97.7	50.1	96.8	82.6	95.1	97.7	95.1	92.6	80.2	95.2	71.7	92.1	83.8	26-Oct-2019
SRC-B-MachineLearningLab [?]	88.5	97.2	78.6	97.1	80.6	89.7	97.4	93.7	96.7	59.1	95.4	81.1	93.2	97.5	94.2	92.9	73.5	93.3	74.2	91.0	85.0	19-Apr-2018
DeepLabv3+_AASPP [?]	88.5	97.4	80.3	97.1	80.1	89.3	97.4	94.1	96.9	**61.9**	95.1	77.2	94.2	97.5	94.4	93.0	72.4	93.8	72.6	93.3	83.3	22-May-2018
** SepaNet ** [?]	88.3	97.2	80.2	96.2	80.0	89.2	97.3	94.7	97.7	48.6	95.0	81.6	95.2	97.5	95.1	92.7	79.5	95.4	68.8	90.9	83.4	25-Oct-2019
** EMANet152 ** [?]	88.2	96.8	79.4	96.0	83.6	88.1	97.1	95.0	96.6	49.4	95.4	77.8	94.8	96.8	95.1	92.0	79.3	95.9	68.5	91.7	85.6	15-Aug-2019
** SpDConv2 ** [?]	88.1	96.9	79.7	96.8	80.2	87.8	98.0	92.3	96.0	57.2	95.8	82.1	92.3	97.3	93.6	93.0	71.4	92.3	75.8	90.7	83.8	06-Jan-2021
** KSAC-H ** [?]	88.1	97.2	79.9	96.3	76.5	86.5	97.5	94.5	96.9	54.8	95.3	81.4	93.7	97.2	94.0	92.8	77.3	94.4	73.5	91.1	83.4	26-Oct-2019
MSCI [?]	88.0	96.8	76.8	97.0	80.6	89.3	97.4	93.8	97.1	56.7	94.3	78.3	93.5	97.1	94.0	92.8	72.3	92.6	73.6	90.8	85.4	08-Jul-2018
** A new feature fusion method: FillIn ** [?]	88.0	97.1	80.8	96.7	77.6	89.2	97.4	92.2	96.9	58.3	94.3	79.4	93.1	97.3	94.4	93.2	73.6	93.0	72.6	89.7	83.4	25-May-2020
ExFuse [?]	87.9	96.8	80.3	97.0	82.5	87.8	96.3	92.6	96.4	53.3	94.3	78.4	94.1	94.9	91.6	92.3	81.7	94.8	70.3	90.1	83.8	22-May-2018
DeepLabv3+ [?]	87.8	97.0	77.1	97.1	79.3	89.3	97.4	93.2	96.6	56.9	95.0	79.2	93.1	97.0	94.0	92.8	71.3	92.9	72.4	91.0	84.9	09-Feb-2018

As you cann see in the above figure, DeepLab V3 has performed best among all hte deepLab benchmakrs. Therefore we chose DeepLabV3 as the model.

Since DeepLab model is original coded in Caffe, we wanted to find an implementation of that model in Python. Then we found DeepLabv3 model is inbuilt in the Official PyTorch Computer Vision Library.

There also, had many models for the same DeepLabv3 model architecture because of the variation of the backbone of the model.

1. ResNet-50
2. ResNet-101
3. MobileNet-V3

Since computational cost is important for us, we determined to use DeepLabV3 with the backbone of Resnet-50.

The following code is to choosing the relevant version of the model.

```
import torch

model = torch.hub.load('pytorch/vision:v0.10.0', 'deeplabv3_resnet50
    ', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', '
    deeplabv3_resnet101', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', '
    deeplabv3_mobilenet_v3_large', pretrained=True)
model.eval()
```

B.1.5 Infering the Model using Transfer Learning - 8th March 2024

Leveraging transfer learning, we proceeded to infer the DeepLabV3 model pretrained on **PASCAL VOC Dataset**.

This dataset contains with targeted labels as followings:

- *Person:* person
- *Animal:* bird, cat, cow, dog, horse, sheep
- *Vehicle:* aeroplane, bicycle, boat, bus, car, motorbike, train
- *Indoor:* bottle, chair, dining table, potted plant, sofa, tv/monitor

Other datasets involved in this training Cityscapes & Pascal Context

You can see there is no class for box in any of the datasets, which we need for our objective of the task. Therefore using **Transfer Learning** on a pre-trained method was the apparent easiest method we saw.

Transfer learning is a machine learning technique where a model developed for a particular task is reused as the starting point for a model on a second task. It involves taking a pre-trained model, which has been trained on a large dataset for a specific task, and fine-tuning it on a different but related task.

By fine-tuning the DeepLab model on our specific dataset, we anticipated rapid convergence and improved performance.

As guided by our supervisor, we were advised used a bottom-up approach when identifying corners of the boxes images.

The following code is for plotting the mask on a given input image

```
import torchvision.transforms as T
from torchvision import models
from PIL import Image
import matplotlib.pyplot as plt

# Define a transformation to preprocess the input image
transform = T.Compose([
    T.Resize((256, 256)),  # Resize the image to the required input
        size of the model
    T.ToTensor(),          # Convert PIL image to PyTorch tensor
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
        0.225])  # Normalize the image
])


# Load and preprocess the input image
image = Image.open(sample_dir + "img1.jpg" )
input_tensor = transform(image).unsqueeze(0)  # Add batch dimension


# Perform inference
with torch.no_grad():
    output = model(input_tensor)['out'][0]  # 'out' is the key for
        the segmentation mask

# Convert the output tensor to a numpy array
segmentation_mask = output.argmax(0).cpu().numpy()
```

```
# Function to get segment masks using DeepLabV3 model
def get_segments_from_deeplabv3(sample_dir, img_name):
# Load and preprocess the input image
  image = Image.open(sample_dir + img_name )

  input_tensor = transform(image).unsqueeze(0)  # Add batch
     dimension


  # Perform inference
  with torch.no_grad():
      output = model(input_tensor)['out'][0]  # 'out' is the key for
          the segmentation mask

  # Convert the output tensor to a numpy array
  segmentation_mask = output.argmax(0).cpu().numpy()

  # Visualize the input image and segmentation mask
  fig, axes = plt.subplots(1, 2, figsize=(12, 6))

  # Display input image
  axes[0].imshow(image)
  axes[0].set_title('Input Image')
  axes[0].axis('off')

  # Display segmentation mask
  axes[1].imshow(segmentation_mask, cmap='jet', vmin=0, vmax=20)  #
     Adjust vmin and vmax according to your classes
  axes[1].set_title('Segmentation Mask')
  axes[1].axis('off')

  plt.show()
```

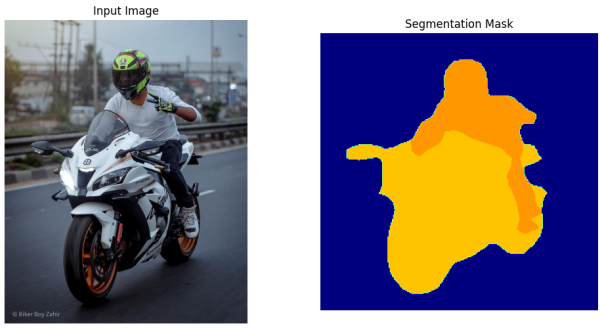


Figure B.3: Left : Input image; Right : Masks generated by DeepLabv3

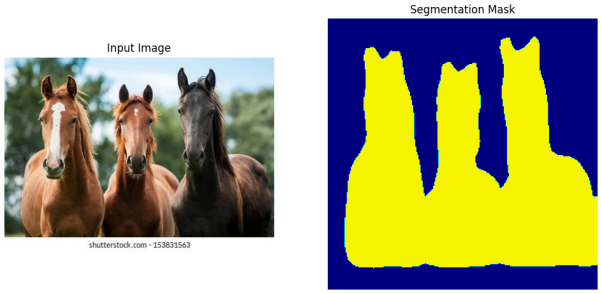


Figure B.4: Left : Input image; Right : Masks generated by DeepLabv3

To see whether the model is actually working, we wanted to inference this model on a image with classes which were in the original dataset. Here, you can see the model successfully segments image into relevant classes. Person, Motorbike and horse respectively.

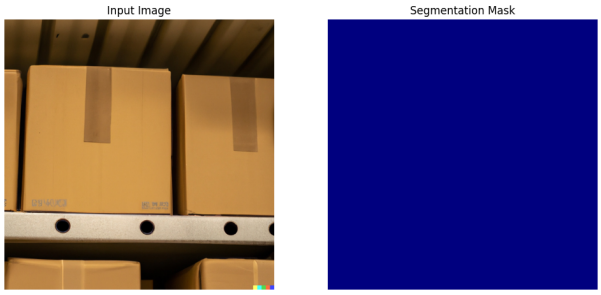


Figure B.5: Left : Input image; Right : Masks generated by DeepLabv3

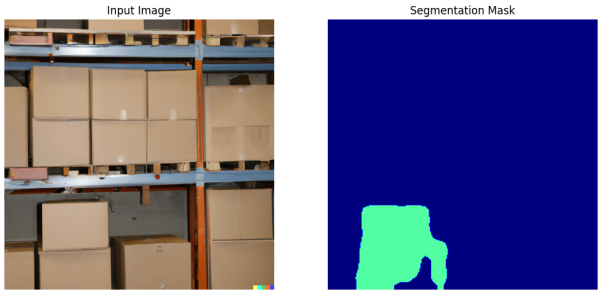


Figure B.6: Left : Input image; Right : Masks generated by DeepLabv3

In the above, it is obvious that model is not trained on any images of boxes and it is clear that the pretrained model has not seen images with boxes before. It is verified that the transfer learning is the best option. For transfer-learning it is necessary to have a custom dataset. Then we move onto making a new custom dataset with images of boxes.

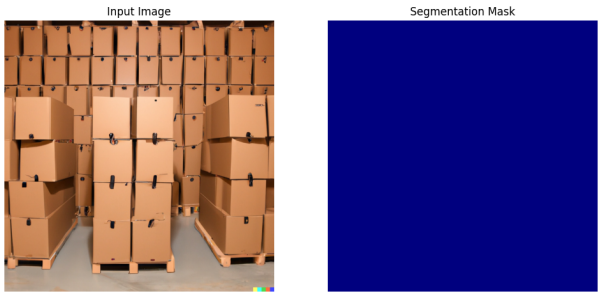


Figure B.7: Left : Input image; Right : Masks generated by DeepLabv3

B.1.6 Researching Alternative Models - YOLOV7 - 16th March 2024

To address the shortcomings identified with DeepLab, we embarked on researching alternative models suited to object detection tasks. YOLO (You Only Look Once) emerged as a promising candidate due to its real-time performance and high accuracy. YOLO operates by dividing the input image into a grid and predicting bounding boxes and class probabilities directly from the grid cells, thereby streamlining the inference process and improving efficiency. We wanted to know that this YOLOv7 model is pretrained on a dataset which contains boxes. Here also we got the same results.

Figure B.8: Inference results from Yolo v7

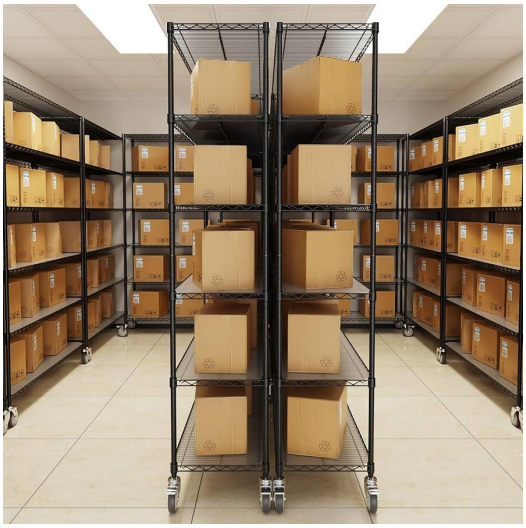
Figure B.9: Inference results from Yolo v7

Figure B.10: Inference results from Yolo v7

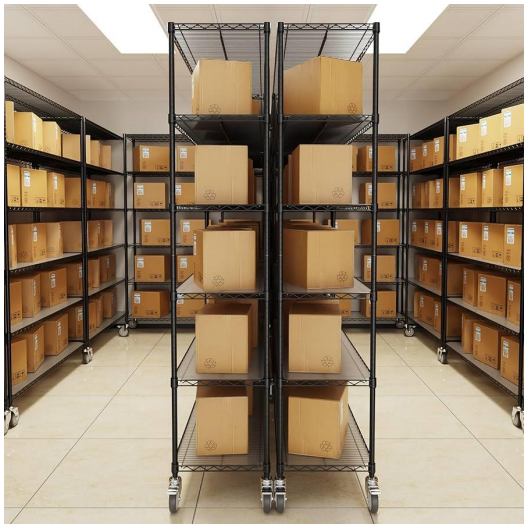
Here, the boxes have been detected as "refregirator", "laptop",

B.1.7 Dataset Creation - 18th March 2024

Recognizing the importance of high-quality training data, we initiated the creation of a custom dataset tailored to our specific application domain. This dataset comprised annotated images depicting various types of boxes commonly encountered in bin picking scenarios. Annotating these images with bounding box coordinates facilitated the training of our object detection models, ensuring they could accurately localize and classify boxes in real-world environments.



(a) Subfigure 1



(b) Subfigure 2

Figure B.11: Two subfigures

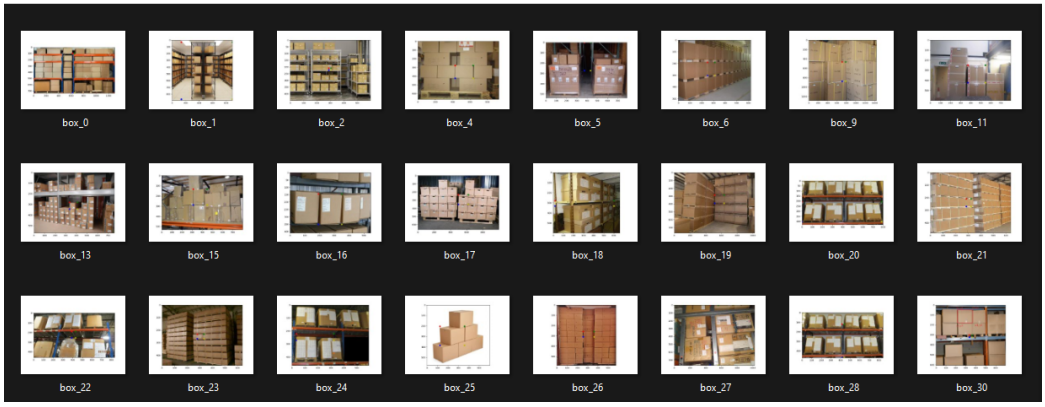


Figure B.13: Snapshot of original images in our custom dataset

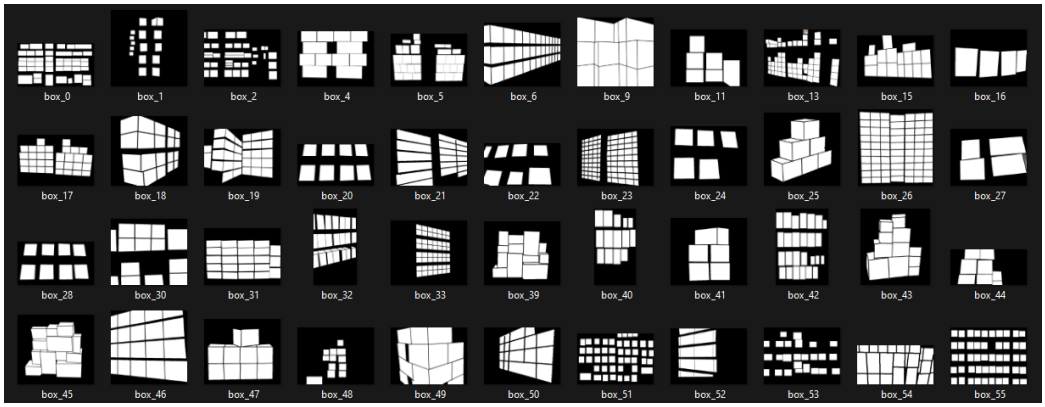
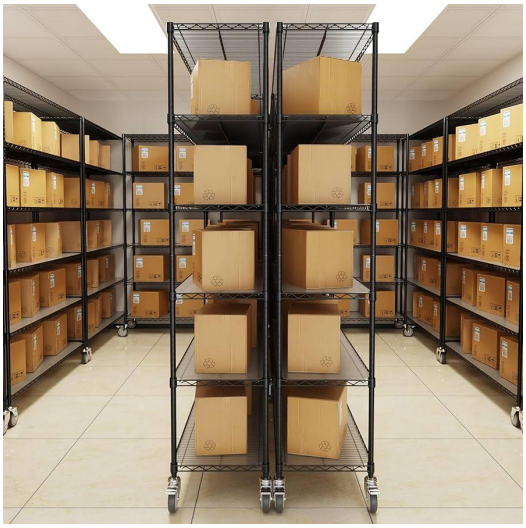
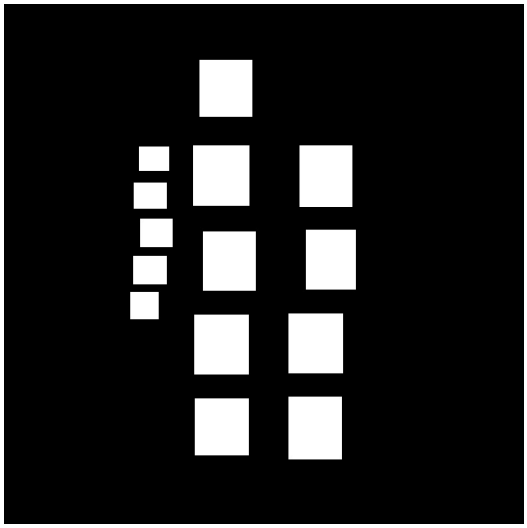


Figure B.14: Snapshot of annotated images in our custom dataset



(a) Sample image from our custom dataset



(b) Corresponding mask for the image

Figure B.12: Sample image pair from the custom dataset

B.1.8 Utilizing SAM Model - 23rd March 2024

As an alternative approach, we explored the SAM (Segment Anything) Model, which obviates the need for extensive dataset annotation and training. The SAM Model leverages self-attention mechanisms to capture global context information and infer object locations without explicit supervision. By leveraging the pre-trained SAM Model, which is already adept at box detection tasks, we aimed to expedite the deployment of our bin picking solution while minimizing the overhead associated with dataset creation. We halted the implementatation of Transfer Learning on DeepLabv3 and moved onto implementing of SAM model, because of its promising results.

Here only downside is using a transformer based architecture, which is computationally expensive. In the following figure, this gives us a promising result than DeepLabv3.

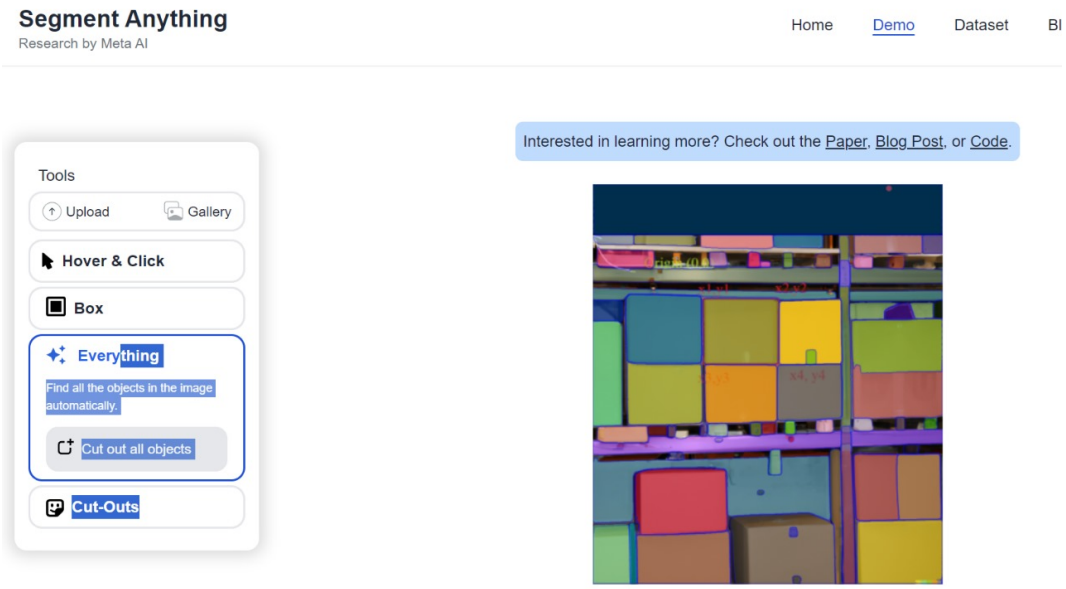
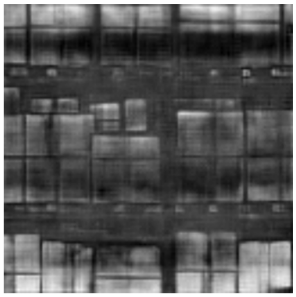


Figure B.15: Snapshot of Demo from SAM model on a image containg boxes

B.1.9 Finetuning DeepLab model on box image dataset - 4th April 2024

(a) Sample input image



(b) Generated output mask for the input image

Figure B.16: Fine-tuning the DeepLab model

B.2 SegNet

B.2.1 Introduction and Literary Review

Segnet is a deep CNN model that has two distinct sections. The bottom section, also called an encoder, down-samples the input to generate features representative of the input. The top decoder section up-samples the features to create per-pixel classification. Each section is composed of a sequence of Conv-BN-ReLU blocks. These blocks also incorporate pooling or unpooling layers in down-sampling and up-sampling paths respectively. Below figure shows the arrangement of the layers in more detail. SegNet uses the pooling indices from the max-pooling operation in the encoder to determine which values to copy over during the max-unpooling operation in the decoder. While each element of an activation tensor is 4-bytes (32-bits), an offset within a 2x2 square can be stored using just 2-bits. This is more efficient in terms of memory used since these activations need to be stored while the model runs. [7]

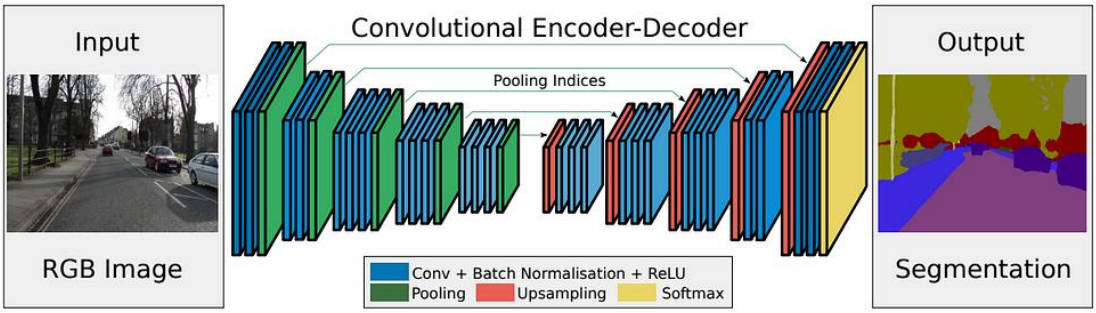


Figure B.17: SegNet Architecture Diagram

B.2.2 Implementation Trials 7/3/2024

A pre-trained SegNet model [6] was tried to get setup in order to get to know the model implementations with SegNet. Here is a step-by-step process followed and the challenges faced along the way.

The Implementation was derived from an existing implementation found on the web.

First, the following libraries and packages were imported to the Google Colab environment.

```
Cv2
Numpy
Pandas
Pickle
Tensorflow
```

After that dataset needed to be categorized as below to feed the dataset to the model without errors.

```
data_path = 'Dataset/'

train_path = "Dataset/train/"
train_label_path = "Dataset/trainannot/"

valid_path = "Dataset/val/"
valid_label_path = "Dataset/valannot/"

test_path = "Dataset/test/"
```

```
test label. _path = "Dataset/testannot/"
```

During compilation, an error was found stating an indexing issue in Matplolib's GridSpec to arrange a grid of subplot for visualizing image data.

```
IndexError: index 0 is out of bounds for axis 0 with size 0
```

The error was fixed by having the random index generator to generate values starting from q1 instead of 0.

During the transition to a Roboflow dataset (more details in dataset chapter) the implementation of SegNet model by the subgroup was incompatible with the dataset generated by Roboflow and another obstacle was faced.

B.3 ENet

Later in our review of literature published related to segmentation models, we came across ENet (Efficient Neural Network), which is termed as a real-time semantic segmentation neural network designed for efficient processing on embedded systems and mobile devices. It was developed by researchers at Samsung AI Center and first presented in the paper "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation" published in 2016 [6].

Reviewing the research paper of ENet, it was found that performance wise and efficiency wise it is better than SegNet [5].

Network	1024x512	1280x720	Parameters	Model Size
ENet	20.4 ms	32.9 ms	0.36 M	1.5 MB
SegNet	66.5 ms	114.3 ms	29.4 M	117.8 MB

Table B.1: A comparison of computational time, number of parameters and model size required for ENet and SegNet [5]

The key advantages of ENet are its high speed, low computational requirements, and smaller memory footprint compared to existing models, while maintaining comparable accuracy. Considering the viability of the model for our project which mainly focuses on deploying a reliable model to identify boxes in an industrial environment, a suggestion was made at a discussion with the supervisor to consider this model, as it would allow us to deploy such a model in edge environments where memory and process is a constrained resource. The suggestion was reviewed and approved by the supervisor later and it lead to the transition from SegNet to ENet.

B.3.1 Network Architecture

The ENet architecture follows an encoder-decoder style topology with repeated modules of main and lateral branches. The encoder path captures higher-level semantics through a sequence of pooling and convolution operations, while the lateral connections preserve spatial information from earlier stages. The core building blocks are bottleneck modules with varying complexities [5]:

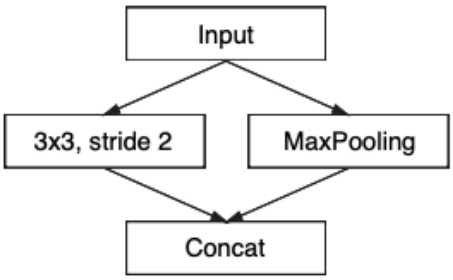


Figure B.18: ENet initial block

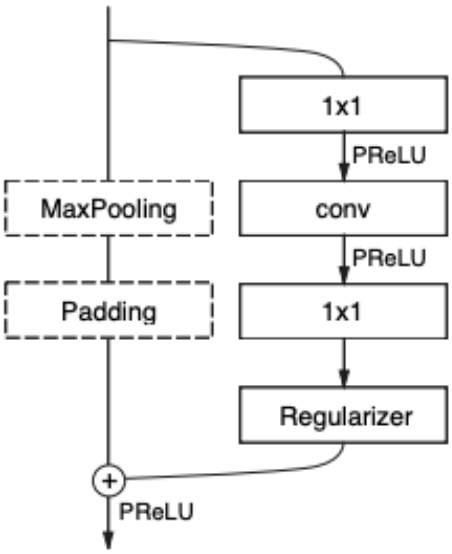


Figure B.19: ENet bottleneck module

The following implementation in Pytorch achieves the network architecture provided by the paper [4]

Initial Block: Combines max-pooling and projection paths

Regular Bottleneck: Applies identity mappings and 1x1 projections

Downsampling Bottleneck: Reduces spatial dimensions with striding.

Dilated Bottleneck: Employs dilated convolutions to expand receptive fields.

Upsampling Bottleneck: Bilinear upsampling and fusion of encoder features.

This highly efficient architecture allows ENet to achieve real-time inference speeds while maintaining accuracy comparable to larger models like SegNet.

B.4 Model Transition Plan 13/03/2024

The following were planned to be acted on to create a model based on the original paper on ENet.

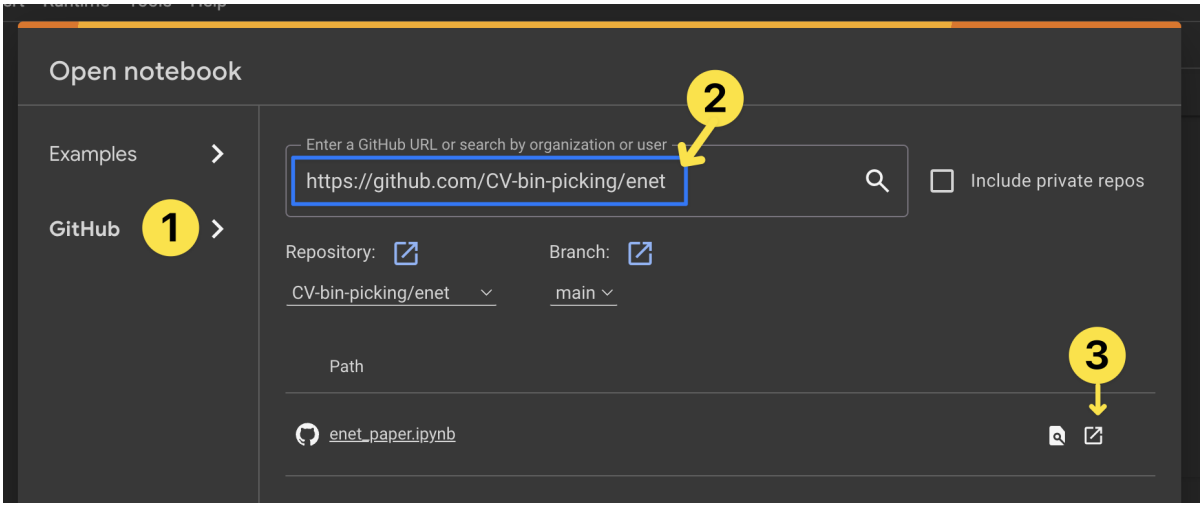
1. Obtain the datasets mentioned in the paper (Cityscapes, CamVid, SUN RGB-D) and prepare them for training and testing.
2. Implement the ENet architecture as described in the paper, including the encoder, decoder, and bottleneck modules. Using deep learning framework PyTorch
3. Apply the design choices mentioned in the paper, such as early down-sampling, dilated convolutions, and Spatial Dropout.
4. Train the ENet model on the datasets using the Adam optimization algorithm and the custom class weighing scheme described in the paper.
5. Evaluate the trained model's performance on the test sets of the datasets, using metrics like class average accuracy, mean Intersection over Union (IoU), and inference time.
6. Compare the results with existing models like SegNet, as done in the paper, and analyze the trade-offs between accuracy and processing time.
7. Optionally, run the trained ENet model on embedded devices like the NVIDIA Jetson TX1 to assess its real-time performance and resource requirements.

B.5 ENet Paper Implementation 23/03/2024

Paper implementation of ENet on CamVid dataset [4]. This guide assumes basic familiarity with notebooks and will include a brief setup process to get started with Google Colab.

Notebook Environment Setup: Google Colab

1. Go to `colab.research.google.com` -¿ File -¿ Open Notebook -¿ Search for the notebook from the Github Repo `https://github.com/CV-bin-picking/enet` and open it.



2. Connect to GPU Runtime: In the menubar, go to Runtime -¿ Change runtime type. In the pop-up window, Runtime type as Python -¿ Select T4 GPU as the hardware accelerator -¿ Click Save.

   A Google Account is required. Colab interface is constantly changing, and it will autodetect recommended configurations for the notebook at launch. User is expected to do the best in either cases as GPU will improve the training time dramatically.

3. Importing dependencies: Execute the first cell in the notebook to prepare the python environment by importing required dependencies. *You can ignore the warning about the notebook being not authored by Google after you've reviewed the source code*

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
from torch.optim.lr_scheduler import StepLR
import cv2
import os
from tqdm import tqdm
from PIL import Image
```

4. Configuring root directory: Change the root_path variable string to "/content" before executing the second cell. This will configure the root directory where the model searches for the dataset and dumps the trained models. *This step is allows the program to be executed in Colab as well as locally in your python environment with appropriate configuration.*

```
root_path = "/content" # change to '/content' for google colab
```

5. Configuring CamVid dataset: Uncomment the code in the third cell and execute it to download the CamVid dataset and extract it in the Colab root directory.

```
!wget https://www.dropbox.com/s/pxcz2wdz04zxocq/CamVid.zip?dl=1
   -O CamVid.zip
!unzip CamVid.zip
```

This is initially commented to prevent accidentally downloading the CamVid dataset during bulk runs which might conflict with other datasets if you were to test them in the same notebook, which our subgroup is doing to transition to our own dataset

When the first three cells are successfully ran, the Google Colab environment is setup with the dataset and now we can create the network models.

ENet Model

Full extend of the code is not presented here for the purpose of brevity, it is provided via the Github repo or the appendix

1. Creating the network blocks: Execute the next five cells to create the required architecture blocks proposed earlier for the network setup using Pytorch.

```
class InitialBlock(nn.Module):
    def __init__ (self,in_channels = 3,out_channels = 13):
        super().__init__()
        self.maxpool = nn.MaxPool2d(kernel_size=2,
                                    stride = 2,
                                    padding = 0)
        self.conv = nn.Conv2d(in_channels,
                              out_channels,
                              kernel_size = 3,
                              stride = 2,
                              padding = 1)
        self.prelu = nn.PReLU(16)
        self.batchnorm = nn.BatchNorm2d(out_channels)

    def forward(self, x):
        main = self.conv(x)
        main = self.batchnorm(main)
        side = self.maxpool(x)
        # concatenating on the channels axis
        x = torch.cat((main, side), dim=1)
        x = self.prelu(x)
        return x
```

```
class UBNeck(nn.Module):
    def __init__(self, in_channels, out_channels, relu=False,
       projection_ratio=4):
        super().__init__()
        # Define class variables
        # Upsampling

    def forward(self, x, indices):
        x_copy = x
        # Side Branch
        # Main Branch
        # summing the main and side branches
        return x
```

```
class RDDNeck(nn.Module):
    def __init__(self, dilation, in_channels, out_channels,
       down_flag, relu=False, projection_ratio=4, p=0.1):
```

```
        super().__init__()
        # Define class variables
        # calculating the number of reduced channels

    def forward(self, x):
        bs = x.size()[0]
        x_copy = x
        # Side Branch
        # Main Branch
        # Summing main and side branches
        x = x + x_copy
        x = self.prelu3(x)
        if self.down_flag:
            return x, indices
        else:
            return x
```

```
class ASNeck(nn.Module):
    def __init__(self, in_channels, out_channels,
       projection_ratio=4):
    # Asymetric bottleneck:
        super().__init__()
        # Define class variables

    def forward(self, x):
        bs = x.size()[0]
        x_copy = x
        # Side Branch
        # Main Branch
        # Summing main and side branches
        x = x + x_copy
        x = self.prelu3(x)
        return x
```

```
class ENet(nn.Module):
    def __init__(self, C):
        super().__init__()
        # Define class variables
        self.C = C # number of classes
        # The initial block
        self.init = InitialBlock()
        # Five bottlenecks
        # Final ConvTranspose Layer

    def forward(self, x):
        # The initial block
        x = self.init(x)
        # Five bottlenecks
        # Final ConvTranspose Layer
        x = self.fullconv(x)
        return x
```

2. Instantiate the ENet model and attach Pytorch to GPU compute

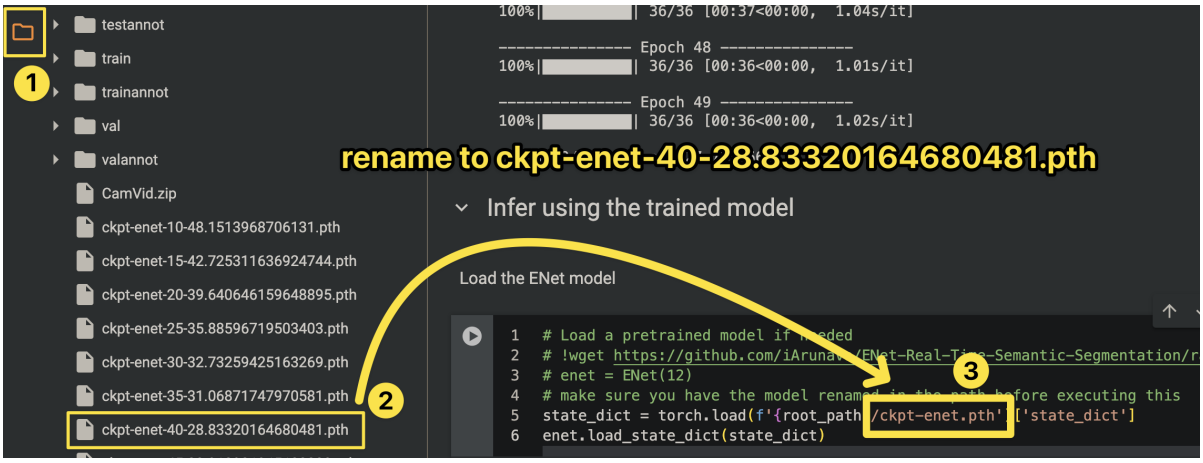
```
enet = ENet(12) # instantiate a 12 class ENet for CamVid
# logic to check and attach to gpu computes in different
    platforms
enet = enet.to(device)
```

3. Defining the loader
4. Defining class weights
5. Defining hyper parameters
6. Training Loop *Training will take a long time even in Google Colab, with the free tier and adjusting to the loads, the current implementation in CamVid dataset runs for about 3 to 5 hours.*

Executing the above cells will define the required Pytorch objects and dump the trained models in configured intervals into the root directory. You have the option to interrupt the training when you have obtained an iteration with satisfactory specifications to check it with the rest of the processes. This is useful for quickly changing certain parameters to evaluate the differences.

Infer using the model

At the end of the training loop many ENet model files would've been dumped into the root folder. We can use the next infer code in the following way by renaming the variable name of the file to load according to the file name in the explorer or by doing vice versa. For example let's assume we want to load the model that is dumped during the 8th checkpoint (40th iteration).



Testing on an image

1. Loading the image: Here you can change the "fname" variable to load the image file from your testing dataset. The code here has been filled with an existing image from the CamVid dataset.

```
fname = 'Seq05VD_f05100.png'
tmg_ = plt.imread(f'{root_path}/test/' + fname)
tmg_ = cv2.resize(tmg_, (512, 512), cv2.INTER_NEAREST)
tmg = torch.tensor(tmg_).unsqueeze(0).float()
tmg = tmg.transpose(2, 3).transpose(1, 2).to(device)
enet.to(device)
```

```
with torch.no_grad():
    out1 = enet(tmg.float()).squeeze(0)
```

2. Generating the class label representation: Execute the following cells in the respective section to generate a label plot for the testing image

```
smg_ = Image.open(f'{root_path}/testannot/' + fname)
smg_ = cv2.resize(np.array(smg_), (512, 512), cv2.INTER_NEAREST)

mno = 8 # Should be between 0 - n-1 | where n is the number of
    classes

figure = plt.figure(figsize=(20, 10))
plt.subplot(1, 3, 1)
plt.title('Input Image')
plt.axis('off')
plt.imshow(tmg_)
plt.subplot(1, 3, 2)
plt.title('Output Image')
plt.axis('off')
plt.imshow(out2[mno, :, :])
plt.show()
```

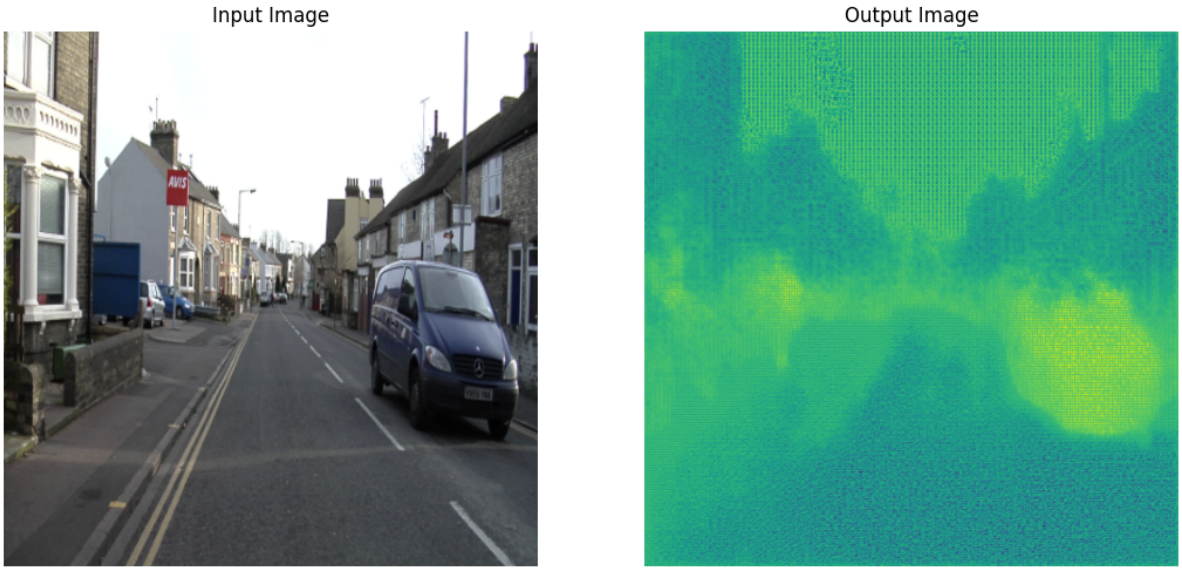


Figure B.20: Image segmentation was noticeable but far from perfect.

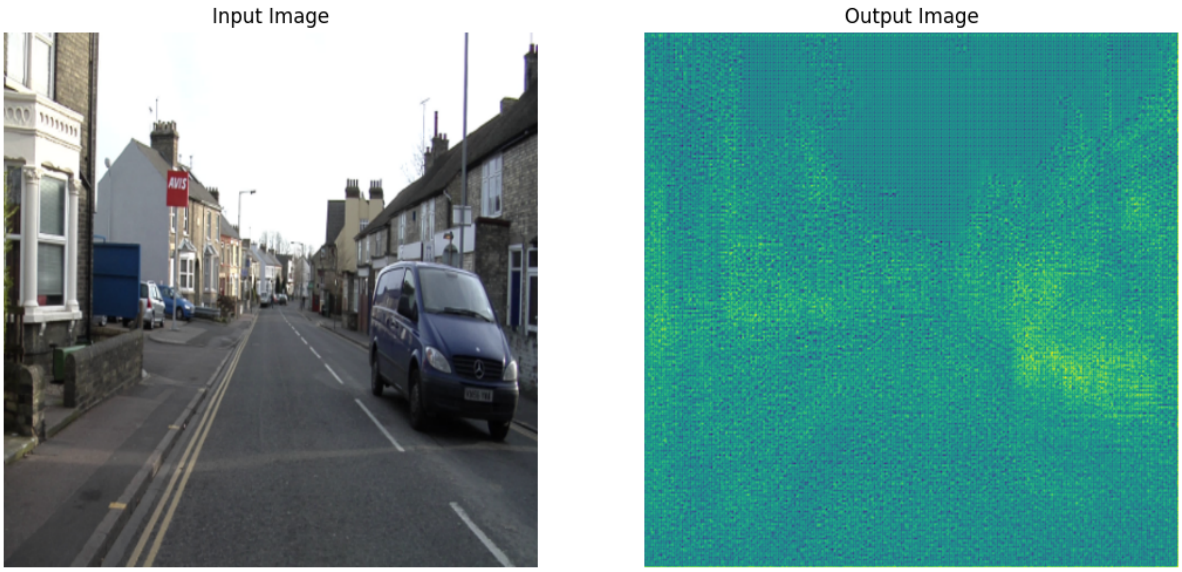


Figure B.21: Unexpected results during a testing stage done at a later date

3. Color mapping and decoding: Executing the following code generates the segmented image and shows the ground truth which was already prepared to compare and observe the performance of the trained model.

```
b_ = out1.data.max(0)[1].cpu().numpy()

def decode_segmap(image):
    Sky = [128, 128, 128]
    Building = [128, 0, 0]
    Pole = [192, 192, 128]
    Road_marking = [255, 69, 0]
    Road = [128, 64, 128]
    Pavement = [60, 40, 222]
    Tree = [128, 128, 0]
    SignSymbol = [192, 128, 128]
    Fence = [64, 64, 128]
    Car = [64, 0, 128]
    Pedestrian = [64, 64, 0]
    Bicyclist = [0, 128, 192]

    label_colours = np.array([Sky, Building, Pole, Road_marking,
        Road,
                              Pavement, Tree, SignSymbol, Fence,
                                  Car,
                              Pedestrian, Bicyclist]).astype(np.
                                  uint8)
    r = np.zeros_like(image).astype(np.uint8)
    g = np.zeros_like(image).astype(np.uint8)
    b = np.zeros_like(image).astype(np.uint8)
    for l in range(0, 12):
        r[image == l] = label_colours[l, 0]
        g[image == l] = label_colours[l, 1]
        b[image == l] = label_colours[l, 2]
```

```
    rgb = np.zeros((image.shape[0], image.shape[1], 3)).astype(
        np.uint8)
    rgb[:, :, 0] = b
    rgb[:, :, 1] = g
    rgb[:, :, 2] = r
    return rgb

# decoding the images
true_seg = decode_segmap(smg_)
pred_seg = decode_segmap(b_)

figure = plt.figure(figsize=(20, 10))
plt.subplot(1, 3, 1)
plt.title('Input Image')
plt.axis('off')
plt.imshow(tmg_)
plt.subplot(1, 3, 2)
plt.title('Predicted Segmentation')
plt.axis('off')
plt.imshow(pred_seg)
plt.subplot(1, 3, 3)
plt.title('Ground Truth')
plt.axis('off')
plt.imshow(true_seg)
plt.show()
```

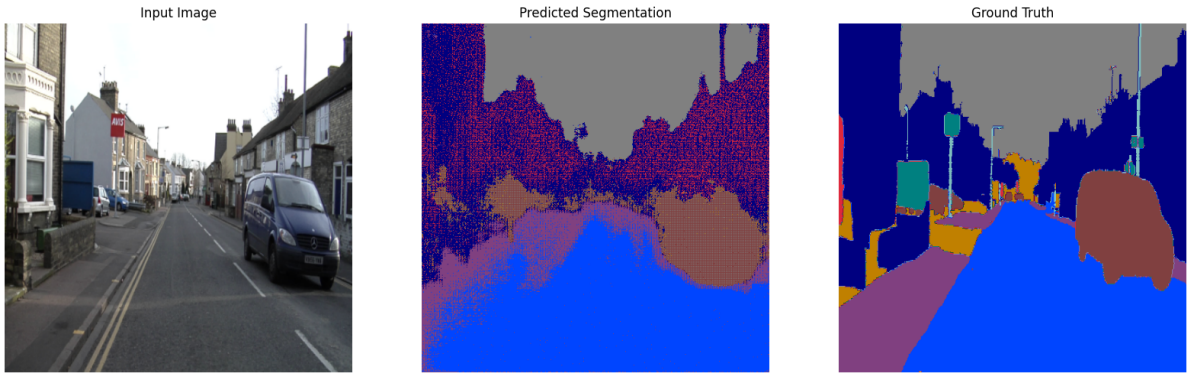


Figure B.22: The initial test results were promising after a training session with 100 iterations

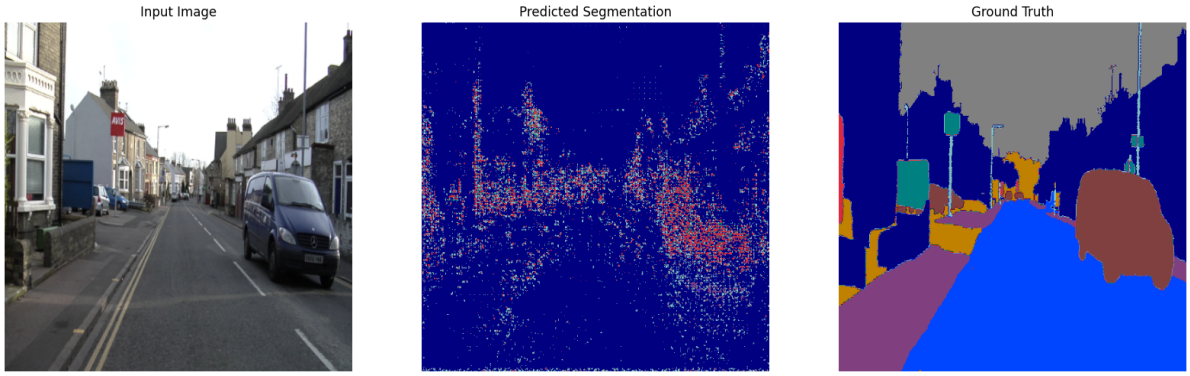


Figure B.23: The results were diminished in quality during another testing session with 100 iterations

Saving the model and or checkpoints

When there is a noticeable improvement observed, it is always a good idea to save the relevant parameters and model. The following code will help to save the relevant model to the root directory while also saving the parameter file in the Pytorch format which can then be used to do testing or for deployment or can be converted into a required format for improvement in other platforms. Any file in the Google Colab can be downloaded by navigating to the file directory as mentioned before and right clicking on the relevant file and clikcing download.

```
# Save the parameters
checkpoint = {
    'epochs' : e,
    'state_dict' : enet.state_dict()
}
torch.save(checkpoint, f'{root_path}/ckpt-enet-1.pth')
# Save the model
torch.save(enet, f'{root_path}/model.pt')
```

B.5.1 Engineering Principles followed during the Implementation

Model Optimization: The model leverages techniques like dimensionality reduction through bottleneck projections, parameter sharing through parallel dilated convolutions, and compact encoder-decoder topology to reduce computational complexity.

Hardware Acceleration: By implementing the model in PyTorch and running it on Google Colab, we can take advantage of hardware acceleration through CUDA and cuDNN libraries on NVIDIA GPUs provided by Google's cloud infrastructure.

Data Parallelism: The training loop is parallelized across multiple batches through PyTorch's DataLoader abstraction, enabling efficient utilization of the available GPU memory.

Checkpointing: The training process supports periodic checkpointing of model weights, allowing training to be resumed from a saved state in case of interruptions or failures.

Modular Design: The model implementation is structured into reusable components like bottleneck modules, making it easier to experiment with architectural variations or adapt the codebase for different tasks.

Visualization: The notebook includes code for visualizing input images, ground truth segmentation masks, and model predictions, facilitating debugging and qualitative evaluation of results.

This implementation was possible within a short time because of the incredible work done on this existing Pytorch implementation [4] by user iArunava and the notebooks from Kaggle by ajax0564 [1], Davidtvs' repo on Pytorch implementation of ENet [3] were also referred to get more understanding.

With the initial steps completed successfully, we have achieved the first two objectives of obtaining the required datasets and implementing the ENet architecture faithfully in PyTorch. The subsequent goals involve training the model using the prescribed techniques, evaluating its performance against benchmarks, assessing inference speed on embedded hardware if viable, and ultimately comparing the results to existing models like SegNet. By methodically following the outlined plan, we aim to leverage ENet's computational efficiency to develop a reliable and lightweight semantic segmentation model tailored for deployment in resource-constrained industrial environments. Proceeding with the remaining objectives, we will meticulously train, optimize and validate the ENet implementation, culminating in a thorough analysis of its real-world applicability for our box identification task.

B.6 CornerNet

[h]

Hei Law, Jia Deng European Conference on Computer Vision (ECCV), 2018 CornerNet is a cutting-edge approach for object detection that departs from traditional methods. Instead of relying on bounding boxes, it represents objects by detecting their corner keypoints. This approach offers several advantages, including potentially higher accuracy and the ability to handle objects with non-rectangular shapes.

B.6.1 System Requirements

Before diving into the setup process, ensure your system meets the following requirements:

- **Operating System:** While not explicitly mentioned, CornerNet is likely developed for Linux-based systems. If you're using Windows or macOS, consider using a virtual machine or cloud environment with Linux installed.
- **NVIDIA GPU:** CornerNet leverages the processing power of an NVIDIA GPU for efficient training and evaluation.
- **CUDA Toolkit:** CUDA is a parallel computing platform from NVIDIA that allows software to access the GPU's capabilities. Make sure you have a compatible version of CUDA installed on your system.

B.6.2 Setting Up the Environment (19 - 28 February 2024)

- **Install Anaconda:** Anaconda is a popular scientific Python distribution that simplifies package management. Download and install the latest version of Anaconda from the official website: `https://www.anaconda.com/`
- **Create a conda Environment:** Conda environments isolate project dependencies, preventing conflicts with other software on your system. Use the following command to create a new environment named `CornerNet`
  `conda create --name CornerNet --file conda_packagelist.txt`

  Replace `conda_packagelist.txt` with the actual filename provided in the CornerNet repository. This file lists the necessary Python packages for running the code.

  Once created, activate the environment using:
  `source activate CornerNet`
- **Compiling Code:** CornerNet utilizes custom C++ code for specific functionalities. You'll need to compile these components for your system:

– **Navigate to the corner pooling layer directory:**
`cd <CornerNet dir>/models/py_utils/_cpools/`

Compile the code using:
`python setup.py install --user`

– **Non-Maximum Suppression (NMS):**
NMS is a technique commonly used in object detection to eliminate redundant bounding boxes. Navigate to the NMS directory and compile the code:
`cd <CornerNet dir>/external`
`Make`

Note: While conda environments effectively manage dependencies, some complexities arose during compilation and installation due to CornerNet's specific requirements.

- `UnsatisfiableError: The following specifications could not be found`
This error occurs due to the conflicting version constraints between packages. We tried to solve the above error using, `-c` flag with `conda install` to specify a channel that might offer compatible versions.
- `PackageNotFoundError: Package not found in any channels: models`
This error indicates that conda cannot locate a specific package or dependency in its channels. We used `conda config --show channels` to verify the packages.

You can directly mount the CornerNet repository to google colab using,
`! git clone https://github.com/princeton-vl/CornerNet`

B.6.3 Download MS COCO Data: (1 - 6 March 2024)

The CornerNet code is evaluated on the COCO dataset. You'll need to download the training/validation data split used in the paper (originally from Faster R-CNN). The download link can be found in the original CornerNet GitHub repository. `https://github.com/princeton-vl/CornerNet`

- Unzip the downloaded file and place the annotation files (containing object labels and locations) under `<CornerNet dir>/data/coco`
- Download the image datasets (2014 Train, 2014 Val, 2017 Test) from the official MS COCO website `https://cocodataset.org/#download`
- Create three directories named trainval2014, minival2014, and testdev2017 under `<CornerNet dir>/data/coco/images/`.
- Copy the training, validation, and testing images to the corresponding directories according to the annotation files. Ensure the images and annotations are paired correctly.

Note: The COCO dataset is a valuable resource for training and evaluating object detection models. CornerNet is compatible with the COCO dataset. We were able to successfully download the COCO dataset to our local environment.

Local Download Considerations: Downloading the COCO dataset can be challenging due to its size. Consider these factors if downloading locally:

- **Storage Space:** Ensure your local environment has sufficient storage to accommodate the entire dataset.
- **Download Time:** Downloading a large dataset over a slow network connection can be time-consuming.

Alternative: Google Colab
If storage or download limitations exist, consider using Google Colab, a free cloud environment with pre-installed libraries. Refer to the following steps for using the COCO dataset within Google Colab.

- **Upload to Google Drive:** Upload the COCO dataset folder from your local machine to your Google Drive.
- **Mount Google Drive in Colab:**
  `from google.colab import drive`
  `drive.mount('/content/gdrive')`
- **Access COCO Dataset:** Once mounted, you can access your COCO dataset using the `path /content/gdrive/MyDrive/Path/to/your/coco/dataset` (replace "MyDrive/Path/to/your/-coco/dataset" with the actual location of your dataset within your Drive).

Alternative: FiftyOne data set loader
The FiftyOne library provides a well-supported and efficient method for loading COCO datasets within your software. FiftyOne is the officially recommended approach according to the COCO website.

Here are the key benefits of using FiftyOne:

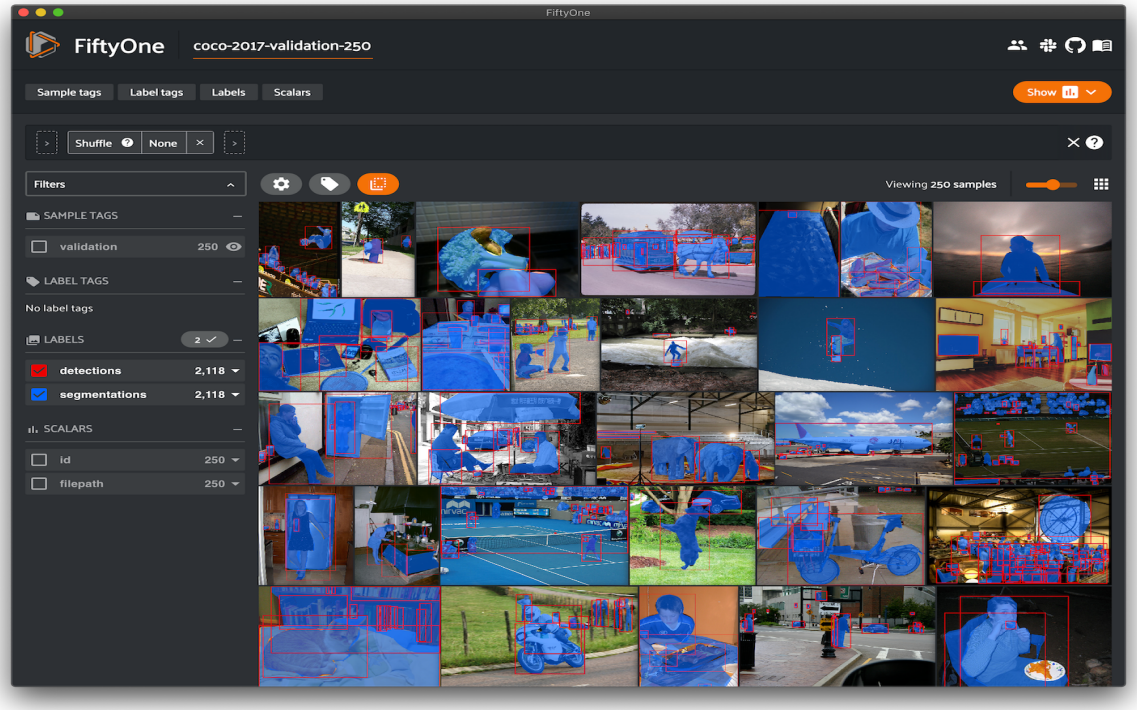
- **Simplified Workflow:** FiftyOne offers a streamlined API for loading COCO data, including images and annotations (bounding boxes, segmentation masks, etc.).
- **Flexibility:** FiftyOne supports various COCO label types, allowing you to load the specific data relevant to your needs.
- **Efficiency:** FiftyOne is optimized for handling large datasets, ensuring smooth operation during the loading process.

For detailed instructions on using FiftyOne with COCO datasets, refer to the FiftyOne documentation: `https://docs.voxel51.com/integrations/coco.html` Use the following steps to implement FiftyOne library.

- **Install the FiftyOne library** python !pip install fiftyone
- **Use the following code to load the coco data set**

```
import fiftyone as fo\\
import fiftyone.zoo as foz
dataset = foz.load_zoo_dataset(
  "coco-2017",
  split="validation",
  max_samples=50,
  shuffle=True,
```

```
)
session = fo.launch_app(dataset)

dataset = foz.load_zoo_dataset(
    "coco-2017",
    split="validation",
    label_types=["detections", "segmentations"],
)

session.dataset = dataset
```



B.6.4 Training a CornerNet Model: (8 - 24 March 2024)

- **Train using `train.py` script:** `python train.py <model>` Replace `<model>` with the name of the model you want to train (e.g., CornerNet).
- **Pre-built Configurations and Models:** The repository provides a configuration file (`CornerNet.json`) and a corresponding model file (CornerNet.py) for the baseline CornerNet model. You can use this directly for training.
- **Using a Pre-trained Model:** The repository also offers a pre-trained CornerNet model, trained for 500k iterations using 10 Titan X GPUs. This can be a good starting point for evaluation or fine-tuning. Download the pre-trained model and place it under `<CornerNet dir>/cache/nnet/CornerNet`. You might need to create this directory if it doesn't exist.

Note: We identified several areas for improvement in the clarity of the original CornerNet documentation. These ambiguities led to challenges during the installation and configuration process. To enhance the user experience, we have addressed these ambiguities and provided more detailed instructions within our current documentation.

- `ModuleNotFoundError`: This occurs due to The libraries required for your training script might not be installed in the current Colab runtime environment.

There are many other documentation errors in CornerNet. The user may face many `DirectoryNotFoundErrors`.

B.6.5 Evaluating a CornerNet Model: (24 - March 2024)

Evaluation using `test.py` script:
`python test.py <model> --testiter <iter> --split <split>`

- Replace `<model>` with the model name (e.g., `CornerNet`).
- Replace `<iter>` with the iteration number of the model (e.g. 500000 for the pre-trained model).
- Replace `<split>` with the dataset split you want to evaluate on (trainval2014, minival2014, or testdev2017).

Prior to using your model for inference, you'll need to compile the custom pooling layers written in C++. This compilation process translates the C++ code into machine-readable instructions optimized for your specific hardware.

Our implementation utilizes the g++ compiler within a Google Colaboratory (Colab) environment for compilation. Colab provides a convenient platform for compiling C++ code without requiring local setup.

Use the following code to compile the pooling layers.
`!apt-get install g++`
`!g++ top_pool.cpp -o top_pool.exe` **Use the following code to load the image for inference**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

def get_corner_from_canny_edges(img_name):
  # Load image from URL or upload it to Colab
  # Make sure to replace 'box_image.jpg' with the path to your image
  image = cv2.imread(sample_dir + img_name)

  # Convert the image to RGB (matplotlib uses RGB format)
  image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

  # Preprocess image if necessary (e.g., convert to grayscale)
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

  # Define threshold values
  threshold1 = 50
  threshold2 = 150

  # Detect edges
  edges = cv2.Canny(image, threshold1, threshold2)

  # Find contours
  contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.
     CHAIN_APPROX_SIMPLE)
```

```
# Filter contours
# (You might need to implement filtering based on your specific
    criteria)

# Approximate contours to polygons
for contour in contours:
    epsilon = 0.04 * cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, epsilon, True)
    if len(approx) == 4:  # Check if it's a quadrilateral (i.e., a
        box)
        box_corners = approx.reshape(-1, 2)
        # print("Box corners:", box_corners)

        # Annotate image with coordinates
        for (x, y) in box_corners:
            cv2.circle(image_rgb, (x, y), 5, (0, 255, 0), -1)
            # cv2.putText(image_rgb, f'({x}, {y})', (x - 50, y -
                10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255)
                , 2)

# Display annotated image
plt.imshow(image_rgb)
plt.axis('off')
plt.show()
```