The Impact of Computer Vision on Assisting Visually Impaired People

Aaryan Wanjari

5/27/24

Abstract

In today's world, artificial intelligence has been a significant new innovation. One of the prime aspects of artificial intelligence is its ability to perform computer vision, which is using machine learning to allow a computer to detect things, via camera or sensors. Computer vision has seen significant development in the past years which can be used to help people who struggle with vision. This research focuses on developing computer vision algorithms to output positioning data for people to see. It includes several steps in the coding process, such as creating an object detection model, a positioning algorithm, and finally manipulating videos to project an output. The goal of this research will be to create a working program that can take a sample video and output correct positioning instructions for a user. This work will hopefully transform the way the visually impaired can navigate the world.

Introduction

Visually impaired people have always suffered with problems with their sight. According to the World Health Organization, "at least 2.2 billion around the world have near or distance vision impairment" [1]. These problems make it hard for visually impaired people to live normally as they struggle to get around without assistance from others. But, what if there was a solution? For someone who struggles with their vision, their biggest hope is an opportunity to see or get around by themselves. To solve this problem visual aids are used to help. For this project the main focus will be on the problem of visually impaired people's ability to navigate outside. Currently people who struggle with sight use canes as they walk around, swinging it left and right to have an idea of what is in their way. While this improves their situation, it is not ideal because of factors such as space and damage. For example, in outdoor places, space is very open. This makes it so that a cane is useless as a human's wingspan is much smaller than the space outdoors. For the factor of damage, what seems to be the problem with these canes is that they need to hit objects because that is how a visually impaired human knows something is in their way. The problem occurs when these canes hit other people or items that are really expensive such as cars. The ideal solution would be to create a device to completely replace this cane. The use of technology can be explored to solve this problem.

People have begun developing several devices to try to accomplish this task, through methods such as sonar or lidar. However, a new development has been explored which is through artificial intelligence, more specifically computer vision. Computer vision is a form of artificial intelligence and machine learning that makes computers detect and classify objects. Like other forms of artificial intelligence, computer vision takes in multiple accords of training data, in the form of images, and uses it to train its model. After that, using sensory devices (cameras, sensors) the model is able to classify and locate items seen through the device [2]. For example, computer vision can be used to detect everyday objects like pencils, erasers, etc. It also can be used for more complex problems like facial recognition and making cars drive by itself. For the use of the problem of creating a device that can help visually impaired people navigate around safely, computer vision can be used to detect objects in given images. By using train and test images, computer vision models can be trained to create, what we call, "bounding boxes" over objects in a picture. This makes a computer differentiate what is an object vs. background in images. By integrating this technology, a device can be created to detect objects in an image in live time, and then based on its relative position generate an automated response telling the visually impaired person where to go in order to avoid a collision with the object. This research

is important because it is the next step to furthering technology to assist visually impaired people, and can potentially help create a more permanent and effective solution to counter their limitations. This project aims to harness computer vision to create a program that can direct a visually impaired user to go left, right. or straight in an outdoor setting.

Dataset

For this project the dataset required to contain image data for computer vision. This data would training data with an input of images and an output of labels for the image. Since the project is meant for an outdoor technology for visually impaired people, a dataset with a lot of outdoor image data is required. The dataset that would achieve all these functions happened to be in the COCO dataset, which includes cars, trucks, bikes, streetlights, etc. [3]. The algorithm that was required for object detection was YOLO [4]. And YOLO uses the COCO dataset to achieve its training data for the model. COCO is highly credible because it has been sponsored by many big name tech companies such as Microsoft and Facebook. The dataset consists of 80 object categories and 91 stuff categories. The total amount of images in the dataset is around 330,000 with 5 captions per image. The mass amount of data that COCO contains is ideal for a computer vision project of this level because it will allow for almost any object to get detected that resides outdoors. However, one limitation that COCO contains is there is no data for trees to be detected. This is a problem because trees are big items in the outdoors that should be detected by a device meant for this project. So, for future versions of this algorithm, more datasets would be required to fill in classifications for more images in the outdoors.

```
52
53 labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck", \
54              "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", \
55              "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", \
56              "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", \
57              "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard", \
58              "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana", \
59              "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake", \
60              "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse", \
61              "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator", \
62              "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]
63
```

Methodology / Models

To solve this problem the process is broken up into three steps. First, a model was created to identify all the objects in a given frame and highlight them via bounding boxes. Then, an algorithm was created to identify where in the frame the object is relative to the center of the image (which would be where the center of the person using the device would be). Finally, a program to run the algorithm on each frame of a video was created to have a final output of this project. An additional output of the video would be adding text on the video, specifying what direction a user would need to go (left, right, or straight).

To begin, source code consisting of Inspirit AI's computer vision google colab folder is added into the colab. The source code includes many functions used to plot an image, crop it, and generate bounding boxes. More importantly the code adds the COCO computer vision data set into the google colab. What is required to happen next is building the computer vision model. First the image needs to be plotted. After that it needs to be cropped to a shape of (416, 416, 3). This is required because the computer vision model requires pictures with that shape to generate bounding boxes. The next step is to load the model. The model that is used for the image recognition is a neural network called tensorflow.keras from YOLO. This model is trained on the

data from the COCO dataset to be able to recognize objects in an image such as cars, trucks, bikes, etc. Once the model has been loaded and trained, it could be used on images. The first image that is tested is an image consisting of many objects, to see if the model can detect images with good accuracy. The model's initial run was a success, and it allowed for the next step to start.

```
import tensorflow as tf

# Load model
darknet = tf.keras.models.load_model(model_path)

yolo_outputs = darknet.predict(new_image)
print(len(yolo_outputs))
print(yolo_outputs[0].shape)
print(yolo_outputs[1].shape)
print(yolo_outputs[2].shape)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
1/1 [==============================] - 5s 5s/step
3
(1, 13, 13, 255)
(1, 26, 26, 255)
(1, 52, 52, 255)
```

```
obj_thresh = 0.4
nms_thresh = 0.45

# Decode the output of the network
boxes = decode_netout(yolo_outputs, obj_thresh, anchors, image_h, image_w, net_h, net_w)

# Suppress non-maximal boxes
boxes = do_nms(boxes, nms_thresh, obj_thresh)

# Draw bounding boxes on the image using labels
image_detect = draw_boxes(image_pil, boxes, labels)

plt.figure(figsize=(12,12))
plt.imshow(image_detect)
plt.show()
```

```
car 1.00 (653, 189) (720, 243)
person 0.88 (325, 189) (348, 258)
person 0.93 (297, 185) (328, 253)
car 0.99 (625, 187) (688, 233)
traffic light 0.43 (224, 123) (239, 153)
car 0.98 (462, 180) (635, 256)
car 0.48 (375, 181) (534, 243)
bus 0.91 (353, 153) (555, 233)
truck 0.95 (0, 161) (268, 278)
```

Based on the bounding boxes generated using the computer vision model, the next task is to determine the relative position of a user to an object in that person's path. To achieve this, multiple functions needed to be created. The first function creates an array of bounding boxes that pulls information from the boxes that the computer vision code creates on an image. Once,

there was information on each bounding box, then more specific functions could be created. These functions consist of code to get the center of a bounding box, get the minimum and maximum x-coordinates of a bounding box, and get the minimum and maximum y-coordinates of a bounding box. To test this code, code is written to add green circles representing center points on the image created by the computer vision model. This results in bounding boxes surrounding each detected image, with a green circle in the middle of each of these boxes. Next, a function is created to determine if an object in an image is in the center of the image (meaning it is in the user's path). The name of the function is called list_locations, and it takes in the bounding boxes as an input. This function uses the bounding boxes list to get the x values of every bounding box, and then compares it to the x value of the midpoint of the whole image. If the midpoint of the image is in between the minimum and maximum x-coordinates of a bounding box, this means that the box is in the center. This algorithm is run for each bounding box in the image, and an array is returned consisting of numerical values 1 and 0. 1 represents that a box is in the center, and 0 represents that a box is out of the range.

After that, a function to finally determine what direction for a user to go is created. This function is called left_or_right and it takes the bounding boxes and the returned values from the list_locations function as parameters. First, variables are created to store information of the first bounding box that a for loop iterates through. This information consists of the dimensions and positioning data. Then, using this data, an if condition is created to determine position if and only if these specific conditions are met: the box is in center (the list_locations function returned a 1 for that particular box), minimum y-coordinate of the box (the bottom of the box) is greater than the 65 percent of the height of the whole image, and the width of the box is greater than 25 percent of the full image width. If these conditions are satisfied it means that the object that the box is bounding is close enough to the user for the user to make a decision of whether to go left or right. If the condition is met the function will compare the x value of the center of the bounding box to the x value of center of the image. If the center of the box lies to the left of the

center of the image, then the function returns 0 meaning that the user should go right because the object is more to the left of the person. On the contrary, if the center of the box is to the right of the center of the image, then the function returns 1 meaning that the user should go left because the object is more to the right of the person. Finally, if the condition is not met, the function returns 2 meaning that the user should go straight because the algorithm has concluded that there is no object in the user's path at the moment.

```python
# get midpoint of image
midpoint = image_w/2
print(midpoint)
height = image_h
img_width = image_w


def list_location(boxes):
  in_center = []
  # access bounding boxes
  for i in boxes:
    # get the dimensions
    dimensions = get_x(i)
    if (dimensions[0] <= midpoint and dimensions[1] >= midpoint):
      # 1 is inside
      in_center.append(1)
    else:
      # 0 is outside
      in_center.append(0)

  return in_center

def left_or_right(positions, boxes):
  # counter to access positions
  counter = 0;
  for i in boxes:
    # get the dimensions to get the width of bounding box
    x = get_x(i)
    y = get_y(i)
    width = x[1] - x[0]
    length = y[1] - y[0]

    bottom = get_center(i)[1] + length / 2
    range1 = height * 0.65

    # run the statement if the box is in the center and if the box's width is greater than 1/4 of the total width of the image
    if (positions[counter] == 1 and bottom > range1 and width > 0.25 * img_width):
      center_x = get_center(i)[0]
      if (center_x < midpoint):
        # 0 means move right
        return 0
      else:
        # 1 means move left
        return 1
    counter += 1
  # 2 means no need to move
  return 2

bounding_boxes = bound_box(boxes)

def final_result(bounding_boxes):
  positions = list_location(bounding_boxes)
  print(positions)
  answer = left_or_right(positions, bounding_boxes)
  return answer

print(final_result(bounding_boxes))

```

The final step is to use the computer vision model and positioning algorithm on a test video. The model and algorithm is built to work on a single image. So, in order to make it work for a video, each individual frame of the video needs to be iterated through so the model and algorithm can run it. After each frame has been iterated through, then a new video can be created with all the new frames generated. Also, for each frame the positioning algorithm returns a value of 0, 1, or 2. These values are interpreted as 0 representing right, 1 representing left, and 2 representing straight. An if condition is used to make this conversion and then the values are added to a list storing positioning data. The video function that contains this code takes in the test input video as a parameter and returns a new video and the positioning list as outputs. Additionally, from the

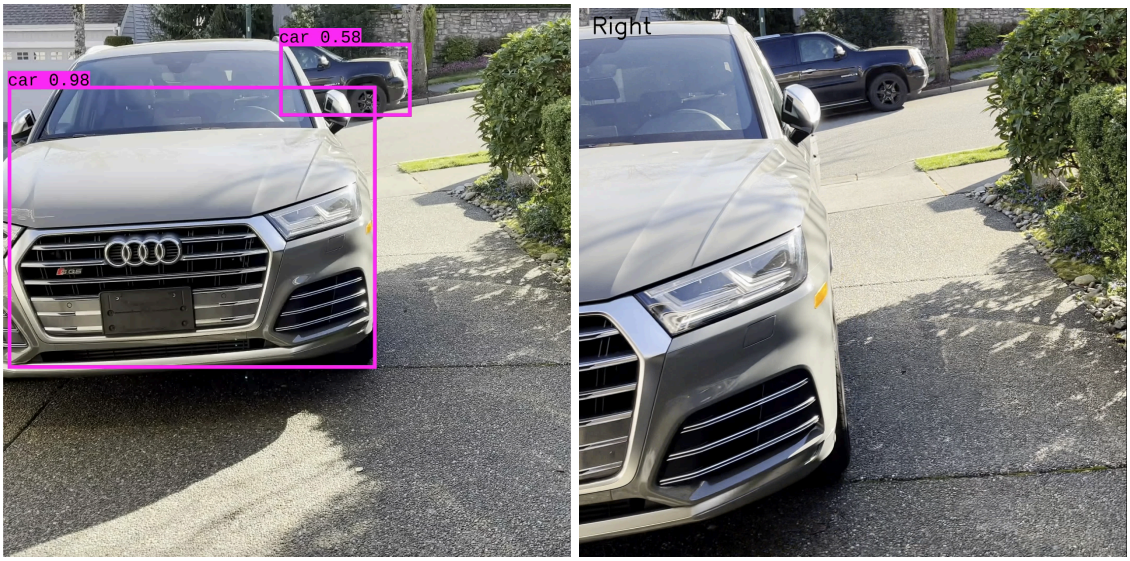
positioning list, code is created to create a text overlay on top of the original test video. This code iterates through the positioning list (consisting of strings only) and then adds text on the video of the string that the list has. To implement this code, a function called add_text is created. This function uses multiple variables for font, size, and text position, and then uses CV2 to add the text onto the input video. Similarly to the last video, the output is a new video file as well.

```python
import cv2

def detect_video(video_path, output_path, obj_thresh = 0.4, nms_thresh = 0.45, darknet=darknet, net_h=416, net_w=416, anchors=anchors, labels=labels):
    vid = cv2.VideoCapture(video_path)
    if not vid.isOpened():
        raise IOError("Couldn't open webcam or video")
    video_FourCC    = int(vid.get(cv2.CAP_PROP_FOURCC))
    video_FourCC    = cv2.VideoWriter_fourcc(*'mp4v')
    video_fps       = vid.get(cv2.CAP_PROP_FPS)
    video_size      = (int(vid.get(cv2.CAP_PROP_FRAME_WIDTH)),
                       int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT)))

    out = cv2.VideoWriter(output_path, video_FourCC, video_fps, video_size)
    #print("here")
    direction_outputs = []

    num_frame = 0
    while vid.isOpened():
      ret, frame = vid.read()
      num_frame += 1
      #print("=== Frame {} ===".format(num_frame))
      if ret:
          ### YOUR CODE HERE
          frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
          image = Image.fromarray(frame)

          n_image = detect_image(image)[0]
          bounding = detect_image(image)[1]
          bounding_boxes_vid = bound_box(bounding)
          left_or_right = final_result(bounding_boxes_vid)

          if (left_or_right == 0):
            direction_outputs.append("Right ")
          elif (left_or_right == 1):
            direction_outputs.append("Left ")
          else:
            direction_outputs.append("Straight ")

          new_frame = np.asarray(n_image)
          new_frame = cv2.cvtColor(new_frame, cv2.COLOR_RGB2BGR)
          ### END CODE
          out.write(new_frame)
      else:
          break
    vid.release()
    out.release()
    print("New video saved!")
    return direction_outputs

vid = cv2.VideoCapture('/content/data/test_vid.mp4')
h = int(vid.get(cv2.CAP_PROP_FOURCC))
codec = chr(h&0xff) + chr((h>>8)&0xff) + chr((h>>16)&0xff) + chr((h>>24)&0xff)
print(codec)

video_path = '/content/data/test_vid.mp4'
output_path = '/content/data/video1_detected.mp4'
outputs = detect_video(video_path, output_path)
```

Results and Discussion

The final product of the project is an end to end system that takes a video input, and returns a step by step navigation through various obstacles and overlaying text specifying what direction the user should walk. This is seen through example videos that are used to test the efficiency of the model. The example video's first output integrates the computer vision model by taking the initial input video and creating a new video with bounding boxes overlaid on objects detected by the model. Items such as cars, trucks, and bikes are detected in the video. The second output of the example video integrates the positioning algorithm that was developed. First the positioning algorithm was run on the input video to get a direction of left, right, or straight for each frame. After that an output video overlaying text specifying the direction in that particular frame was added to the top left of the input video and saved as a new output. The initial goals of the project was to use computer vision to identify objects and direct a user away from those objects to help them better move around. The example output video clearly shows this as it initially directs the user to move right of a car in the users path. Then as the road in front is clear, the model specifies straight to the user. Finally, as a bike emerges in the path of the user (a little to the right of the

user), the program tells the user to move left to avoid the final obstacle in the user's path. This clearly demonstrates success in the models ability to guide a user to navigate outside objects in the COCO dataset.



While the results demonstrate success, there are some limitations that can still be improved on in the future. The first limitations emerge with the dataset. As mentioned previously, the COCO dataset misses some important outside elements such as trees as part of its data. This makes it so that the model cannot detect trees and other greenery like bushes, so if a user comes in contact with those objects, the model specifies no direction. To improve on this limitation more datasets can be used to train the initial computer vision model, so more bounding boxes are generated as a result. With the addition of more items in the dataset, the concept of computer vision to help the visually impaired can be implemented more properly as it should be able to tackle almost any object. This ties in to the second limitation of the model not being able to detect indoor items. While this is also due to the COCO dataset being limited to some images, it is also due to lighting conditions. Good quality cameras are required to counter lighting issues, and it could also help with nighttime conditions. Another limitation includes the use of a powerful headset. The most modern example of such a device would be Apple's vision pro, which has a powerful cpu that could compute real time computer vision outputs [5]. Currently, the model takes an input of a video, but to implement this for visually impaired people, the input would need to be live time. To solve this a device that a visually impaired person can wear on their head (headsets like the vision pro) would be required and they would need to be powerful enough to take in live data and run the model on it at a real time rate. The final limitation has to do with depth. Currently the model figures if an object is close to a user by comparing how far the bottom of the object's bounding box is to the bottom of the image. While this method works theoretically, to make the model more accurate for depth sensing, devices such as a lidar sensor are required. A lidar sensor uses light to sense depth and range. By implementing a device like that to figure out depth, it would make the model more functional. Overall, the primary limitations have something to do with scope and costs. With the correct funding and a bigger team to find more data, this model can be implemented to work as a full time solution to help visually impaired people get around.

Conclusions

Overall, technology that utilizes computer vision will significantly change the world for people with impaired vision. The models are the basis of what can be a huge thing for the future as artificial intelligence and machine learning develop more. Currently, technology such as the apple vision pro is already made to allow people to interact in augmented and virtual reality in real time. This means that there are devices powerful enough to run this model in real time, which can significantly change the world for people with impaired vision. The wait is over as artificial intelligence is growing by the day and soon will apply to almost everything in the world. So, for people who struggle to see properly, it is time that they see through computer vision.

Acknowledgments

I would like to thank Inspirit AI for giving me the opportunity to explore my passion for computer vision and apply it to a topic that I am passionate about. I have bad vision through my glasses as well as some of my family members, so this topic is something that I have always wanted to dive into. To combine my passion with a topic that interests me a lot is something that I have loved the opportunity to work on.

References

[1] "Vision Impairment and blindness." (n.d.). *World Health Organization*, World Health Organization, <https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment#:~:text=Globally%2C%20at%20least%202.2%20billion,are%20refractive%20errors%20and%20cataracts.> (May 19, 2024a).

[2] "What is Computer Vision?: Microsoft Azure." (n.d.). *What Is Computer Vision? | Microsoft Azure*, <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-computer-vision#object-classification> (May 19, 2024).

[3] *Common objects in context* (no date) *COCO*. Available at: https://cocodataset.org/ (Accessed: 27 May 2024).

[4] Redmon, J. *Yolo: Real-time object detection*. Available at: https://pjreddie.com/darknet/yolo/ (Accessed: 27 May 2024).

[5] Lynn, T. (2024) *Using Apple Vision Pro and visionos with computer vision*, *Roboflow Blog*. Available at: https://blog.roboflow.com/apple-vision-pro-visionos-computer-vision/ (Accessed: 27 May 2024).