HIGH SCHOOL EDITION
Journal of Student Research
Volume 13 Issue 2 (2024)

Machine Learning for Visually Impaired: Benchmarking Object Detection Models

Aditya Patra[1] and Rae Crandall[#]

[1]California High School, USA
[#]Advisor

ABSTRACT

This research paper benchmarks object detection models, a form of machine learning, to determine which algorithm would be most beneficial for the visually impaired to locate objects that are used in everyday life. As most benchmarking experiments test object detection models with a variety of objects, it is essential to test the models using images of more relevant objects to find the most suitable algorithm. The models are tested using still images from the COCO database. Pretrained models employing five of the most popular object detection algorithms are used to process the images and find each model's detection accuracy. To simulate real life scenarios, these objects may be partially hidden or at a distance. For each image, the models return a list of detections providing the names, confidence rating, and location of each object detected. These results will be filtered to remove detections with low confidence ratings as well as detections of irrelevant objects. The remaining results are compared to the dataset of object names and locations provided by the COCO database to calculate the distance between the predicted object locations and the true location. The algorithms will be ranked based on the number of failed detections, the time taken to analyze each image, and the accuracy of each object's determined location.

Background

Machine learning emphasizes a computer's ability to use data to learn how to complete specific tasks based on examples rather than instructions. Instead of requiring specific instructions for millions of possibilities, the program can find patterns in a few sample cases that can be applied to other instances as well. Machine learning has proven its effectiveness time and time again with prime examples being chess bots such as Deep Blue and self-driving technology. Machine learning technology is progressing at a rapid pace, with new, more advanced models being released quite often. This makes it imperative to benchmark different models to determine which is most reliable for different use cases. This paper will benchmark five object detection algorithms to determine which is most effective when developing programming applications to help the blind.

In order to understand why different object detection models produce different outputs, it is important to understand how they function. Most newer object detection models use a convolutional neural network as their backbone. A convolutional neural network is a set of convolution layers and pooling layers (layers to simplify the image) connected in series so that the output of one layer feeds into the next.

Convolution layers treat the input image differently if it is black and white compared to if it contains colors. Pixels in black-and-white images can be represented by a single value, so a black-and-white image can be treated as a 2-D array. As pixels in color images must be represented by three scales – red, blue, and yellow – color images are fed into the neural network as a 3-D array.

When fed into a convolution layer, a 3-D array of weights, known as a kernel or filter, will detect patterns in the image. Weights can be both positive and negative and are usually close to 0. During each application of the filter to the image, it multiplies the image values with the corresponding kernel values and adds

the results to create a single value called a dot product. The filter is intentionally smaller than the image so that it can be applied to the image multiple times. When the filter is applied, it is overlapped so the same portion of the image is multiplied with different portions of the weight set. Each convolution layer can employ multiple filters to check for different patterns in the image. The resulting dot products for each filter are organized into a 3-D array known as a feature map. This map highlights different portions of the image depending on the weights of the filter.

The use of more convolution layers will result in a model being able to detect patterns more accurately by detecting more basic features. As convolution layers process not only raw image input but feature maps as well, input from one layer can be sent directly to the next. Subsequent convolution layers can be used to detect more intricate features. Feature maps where the weights revealed a pattern can be combined and sent to the next convolution layer which can use smaller filters to segment the image further.

The convolutional neural network is used both during model training and prediction. During training, objects in the images are highlighted, providing what are known as ground truth bounding boxes, so the model can learn to associate different patterns with different object classes. The features detected during the training are combined to form a single feature map associated with an object class to be used during prediction. During prediction, different models send the image through the neural network differently. Some send the image in as a whole while others use a method known as a scanning window. In this method, the model creates a 2D array. This array is sent through the neural network as it "slides" across the image allowing the network to view objects with greater clarity.

After images are processed during training, the model assigns an accuracy value to each prediction using bounding box regression and cross-entropy. When comparing the predicted bounding boxes and the ground truth bounding boxes, the easiest way to express the accuracy as a value is by calculating the IoU, which is the area of intersection over the area of union. This method makes it difficult to train models as it cannot determine the accuracy if there is no overlap. Instead, bounding box regression utilizes MSE(mean-squared error). Instead of finding overlap, MSE finds the distance between the bounding box coordinates themselves using the following equation where p is the ground truth bounding box and q is the predicted bounding box:

MSE(P, Q): ((p1-q1)^2+(p2-q2)^2+(p3-q3)^2+(p4-q4)^2)/4

There are also multiple ways to compare confidence ratings in predictions. The easiest method would be to subtract the confidence from 100% or simply use the confidence rating as given, however, this operates on too small of a scale and the change in value is constant with the change in confidence rating making it difficult to use for training. Cross-entropy uses a logarithmic equation to extend the range of values and create an exponential increase in the value as confidence decreases. Cross-entropy uses the following equation where p is the ground truth bounding box and q is the predicted bounding box:

CE(p, q): -p*log(q)

Introduction

This paper benchmarks the following object detection models: Yolov8, EfficientDet, Faster RCNN, SSD, and Centernet. Pretrained models from the Tensorflow Hub are trained and tested using images from the COCO database. The results produced by each model are compared to the true values that the COCO database provides for each image. Prior to benchmarking the models, it is important to understand how they differ.

EfficientDet is a single-shot object detection algorithm meaning that the algorithm only utilizes one convolutional neural network. It employs the EfficientNet convolutional network which consists of a series of convolution layers that grow smaller and smaller. This makes the image and patterns more defined near the end as there is more overlap between the filters. Before processing the image using its neural network, EfficientDet performs an operation known as maxpooling in which a small frame is slid across the image with a stride (shift of frame) the same size as the width of the frame. Each application of the frame separates the pixel with the

highest value, effectively downsizing the image to its most important points. The pre-trained EfficientDet model that will be used contains the d2 backbone variation.

Faster RCNN is slightly different from EfficientDet as it utilizes multiple convolutional neural networks making it a two-stage algorithm. It first passes the image through an RPN (Region Proposal Network) which determines where objects might be located in the image. It creates a set of anchor boxes which are rectangles that show subsequent neural networks where to search. The feature maps and anchor boxes are then passed from the RPN to ROI (Region of Interest) layers. These layers accept two inputs, a feature map of the image as well as a 2-D array containing the object class number and the bounding box top-left and bottom-right coordinates for each possible detection. The ROI layer cuts the feature map into a predetermined number of equal portions and extracts the highest value from each section. This allows the ROI layer to convert every detection to the same format making it easier to use. The ROI output is sent to the model backbone (the main convolutional network), which in this case is the inception_resnet_v2 neural network. This method allows the model to limit the area that the Fast R-CNN model has to search. Faster R-CNN also has several maxpooling layers in the backbone as well as ROI (Region of Interest) layers. In this paper, the pretrained Faster RCNN model employs the faster-rcnn-resnet101-v1-1024x1024 backbone.

SSD is a single shot object detection algorithm. It utilizes anchor boxes that are set at multiple locations throughout the image. As SSD does not contain an RPN, these anchor boxes are preset at various locations. The algorithm then runs different resolutions of the image through the neural network to find which anchor boxes contain which patterns. Each anchor box is given a confidence score for each pattern found within the box and keeps the detections above a certain confidence. The pretrained SSD model used in this paper contains the fpnlite-320x320 backbone.

Yolov8 is another single-shot object detection algorithm that employs anchor boxes and maxpooling. In order to detect objects, Yolov8 first converts the image into grid format. It then runs each cell in the grid once through the convolutional neural network. This allows the model to determine which cells an object exists in and double-check whether the object is a false positive or not. For example, if an object was detected at the corner of a cell with low accuracy and wasn't found in any other cell, the prediction would most likely be discounted. The pretrained Yolov8 model used in this paper contains the I_backbone backbone.

Centernet finds objects using a 2-part detection system. First, it sends the image through a convolutional neural network that predicts object locations based off of the geometric center of the object and a key point found during training. After the image is passed through the neural network, Centernet uses a method known as center pooling. In order to find the center of an image, center pooling finds the maximum value in the row and column of a pixel and adds them together. This process is repeated with every pixel and the pixel with the highest value is classified as the center of the object. Centernet also employs cascade corner pooling which finds key points along the border. First, it searches the border for the pixel with the maximum value. Next, it searches the interior of the image along the maximum border value for the highest internal pixel value. These values help pinpoint key points along the corner of an object. Centernet uses the two points identified through center pooling and cascade corner pooling to accurately calculate bounding box locations for the objects. The pretrained model that will be used contains the 101v1-fpn–512x512 backbone variation.

Dataset

As stated in the introduction, this paper uses the COCO image dataset for both training and benchmarking the algorithms. The COCO dataset provides the images and ground truth bounding boxes for the objects. While most other experiments that benchmark object detection model use randomized images to detect a wide variety of objects, the images that are used in this paper contain objects of specific types that the visually impaired commonly search for such as household items. The images are chosen to simulate real life scenarios where objects may be partially obscured.

Computing Resources

When using an object detection model, the speed of the model is dependent on not only the algorithm, but the processor used as well. The hardware components of the computer used to run the object detection models are listed below.

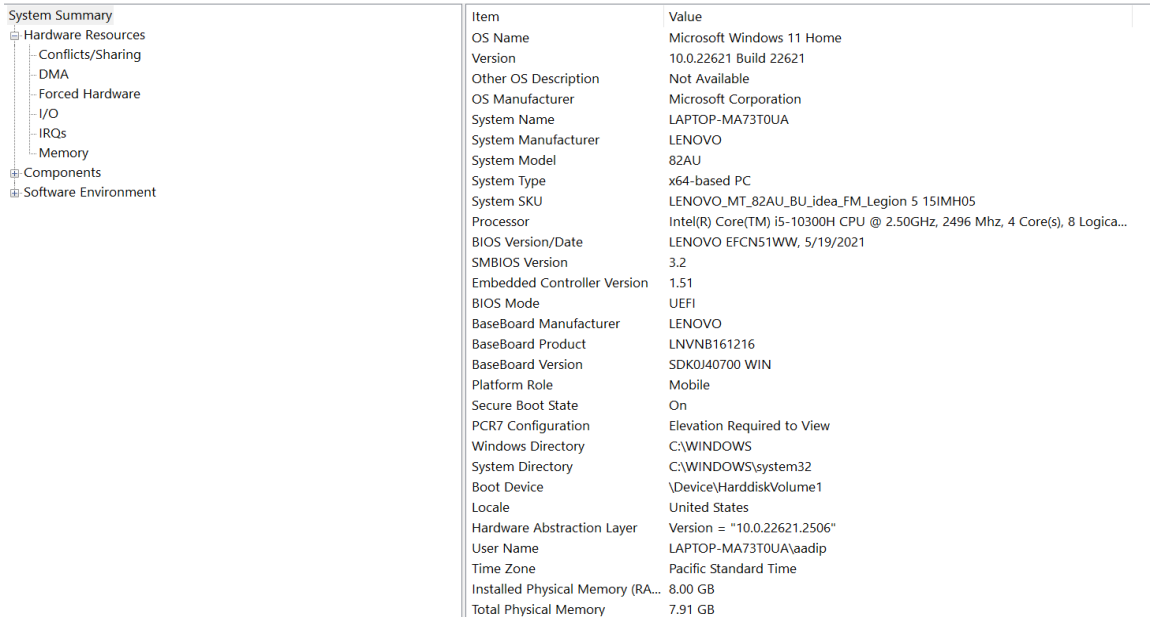

Figure 1. Screenshot showing hardware components of machine used to run object detection model. Processor: Intel Core™ i5-10300H CPU, GPU: NVIDIA GeForce GTX 1650

Predicted Data

For each detection made by the model, the Python script will find the object class, confidence rating, and IoU.

Class

The object class is the type of object the model has detected. The COCO dataset has a list of object classes and corresponding object names ranging from one to eighty.

Confidence

The confidence rating is a percentage of how confident the model is in its prediction.

IoU

IoU is a method of measurement for bounding box accuracy. It calculates the overlap between the predicted bounding box and the ground truth bounding box. This value is then divided by the area of the union of the two boxes. The higher the IoU, the greater the accuracy of the model.

Procedure

The procedure for benchmarking the models is as follows:

- Data Collection
- Model Training
- Object Detection
- Result Compilation
- Benchmarking

Data Collection

As stated previously, the data used is all obtained from the COCO database. This database provides both pictures and object classes/bounding boxes. A python script is used to download the zipped COCO database file. The script then extracts relevant pictures using a list of class ids of relevant objects. In order to make it easier to check for value errors, the script stores the data in two formats in different folders: the raw image and an annotated version of the image. The raw image will be used to test the models while the annotated images are used to check for detection errors as well as errors in IoU calculation.

Model Training

In order to simplify the testing process, this paper uses pretrained models from the tensorflow hub. Training a model using the base algorithm requires a much greater quantity of data as well as more computing power, so it is more efficient to employ pretrained models that were trained on the same dataset.

Object Detection

The tensorflowhub python library is used to access each model. Next, the python file reader is used to obtain the training images. These images are converted to numpy arrays before being passed through the model one-by-one, and the python time library is used to track the amount of time each image takes to be processed. The bounding boxes, confidence scores, and object classes are then extracted from the results. The script uses the confidence scores to separate detections with confidence less than 0.5 - and therefore inaccurate - and the object classes to remove detections of object that are not related to the topic. As the bounding boxes of different algorithms are formatted differently, the script also converts all bounding boxes to the format [ymin, xmin, ymax, xmax].

Result Compilation

For each image tested, the model produces a list of bounding boxes, confidences, and object classes detected. The script iterates through this list to find which result is most closely matched to each of the true values from the COCO database. This is used to calculate IoU. If no bounding boxes with an IoU greater than 0.4 are found, the detection is treated as a fail. For each image, the script stores the average IoU, number of fails, and total time taken to process. This is then stored in a table for later use.

Benchmarking

The results produced by each model will be averaged to produce a single value for each category. These values are used to rank the models in terms of accuracy and speed.

The best model to use when developing an application is usually dependent on the type of application being made. However, when employing object detection to create an application to aid the visually impaired, algorithms are often used to process images rather than videos, therefore, this paper will determine the grading criteria for benchmarking the models using this as the basis. In a scanning app, the ability to detect objects is more important than the time taken to process the objects, therefore the categories from most important to least important are as follows:

- Fails
- IoU
- Speed

Methodology

The best model will be determined by a decision rubric out of 10. A perfect score will be set as 0 failed detections, 0 milliseconds taken to process the image, and an IoU of 1, which is the ideal result. The median of the values will be treated as a score of 5 to prevent outliers from significantly impacting the scores. The scores and values will have a linear relationship.

For the fails category, 1.39 is the median value and will be considered a score of 5. The score will increase by 1 point for every 0.3 decrease in fails. This category will be weighted by a factor of 3 as it is the most important to consider.

For the IoU category, 0.48 will be considered a score of 5 as it is the median value. As the scores and values have a linear relationship, the score will increase by one point for every 0.1 increase in value. This category will be weighted by a factor of 2 as it is more important than speed, but not as important as fails.

For the speed category, 405 milliseconds is the median of the recorded values. The score will increase by 1 point for every 80 millisecond decrease in time taken. This category will be weighted by a factor of 1 as it is the least important to consider.

Model Usage Process

The programs to run each model have the same code for majority of the sections, however, the method to obtain the model and formatting the output is different for each. The SSD model script can be found in the appendix section as an example of how to run and analyze a model's output, and the method for loading each model and formatting the output will be listed below. As the Yolo model was loaded from a different library than the other models, the process of using the model will also be included in the Yolo section.

Yolo

```python
from ultralytics import YOLO
import os
import cv2
import time
#importing pretrained model
model = YOLO('yolov8n.pt')
```

Figure 2. Screenshot of libraries needed and importing the Yolo model.

```python
#start timer
start = time.time()*1000
print(file)
names_list.append(str(file))
img = cv2.imread("images_train/"+file)
img_height, img_width, c = img.shape
img_name = file.split('.')[0]
img_name_YOLO = "image"+str(num)
results = model.predict(source=img, show=True, conf=0.4, save=True, save_txt=True)
#getting result values
prediction_num = "predict"
if num > 0:
    prediction_num = "predict"+str(num+1)
file_name_1 = os.path.join("runs", prediction_num, "labels", img_name_YOLO)+".txt"
#getting prediction data from folder where it is stored
f = open("C:/Users/aadip/PycharmProjects/benchmarking/runs/detect/predict/labels/image0.txt", 'r')
lines = f.readlines()
f.close()
f = open("C:/Users/aadip/PycharmProjects/benchmarking/runs/detect/predict/labels/image0.txt", 'w').close()
boxes_YOLO = []
classes_YOLO = []
for line in lines:
    line_list = line.split(" ")
    classes_YOLO.append(line_list[0])
    line_list.pop(0)
    line_list[3] = line_list[3].split("\n")[0]
    boxes_YOLO.append(line_list)
names = results[0].names
```

Figure 3. Screenshot of code to load image and process image using model. The results appear in a different file, so the script also needs to open the results file and extract the data.

```python
boxes_YOLO[i][0] = float(boxes_YOLO[i][0])
boxes_YOLO[i][1] = float(boxes_YOLO[i][1])
boxes_YOLO[i][2] = float(boxes_YOLO[i][2])
boxes_YOLO[i][3] = float(boxes_YOLO[i][3])
YOLO_x1 = boxes_YOLO[i][0] - boxes_YOLO[i][2]/2
YOLO_x2 = boxes_YOLO[i][0] + boxes_YOLO[i][2]/2
YOLO_y1 = boxes_YOLO[i][1] - boxes_YOLO[i][3]/2
YOLO_y2 = boxes_YOLO[i][1] + boxes_YOLO[i][3]/2
```

Figure 4. Screenshot of Yolo bounding box data being converted to correct form for comparing results to true values.

Efficient Det

```
#model import
detector = hub.load("https://www.kaggle.com/models/tensorflow/efficientdet/frameworks/tensorFlow2/variations/d0/versions/1")
```

Figure 5. Importing pretrained model for Efficient Det(model url: https://www.kaggle.com/models/tensorflow/efficientdet/frameworks/tensorFlow2/variations/d0/versions/1)

```
    #ymin, xmin, ymax, xmax
    xmin = (float(cx)-float(width)/2)
    xmax = (float(cx)+float(width)/2)
    ymin = (float(cy)-float(height)/2)
    ymax = (float(cy)+float(height)/2)
    bboxes_Eff.append([ymin, xmin, ymax, xmax])
```

Figure 6. Formatting bounding boxes of Efficient Det results

SSD

```
#model import
detector = hub.load("https://www.kaggle.com/models/tensorflow/ssd-mobilenet-v2/frameworks/tensorFlow2/variations/fpnlite-320x320/v
```

Figure 7. Importing pretrained model for SSD(model url: https://www.kaggle.com/models/tensorflow/ssd-mobilenet-v2/frameworks/tensorFlow2/variations/fpnlite-320x320/versions/1)

```
    #ymin, xmin, ymax, xmax
    xmin = float(xmin)
    xmax = float(xmax)
    ymin = float(ymin)
    ymax = float(ymax)
    bboxes_SSD.append([ymin, xmin, ymax, xmax])
```

Figure 8. Formatting bounding boxes of SSD results

Faster R-CNN

```
#import model
detector = hub.load("https://www.kaggle.com/models/tensorflow/faster-rcnn-resnet-v1/frameworks/tensorFlow2/variations/faster-rcnn-
```

Figure 9. Importing pretrained model for Faster R-CNN(model url: https://www.kaggle.com/models/tensorflow/faster-rcnn-resnet-v1/frameworks/tensorFlow2/variations/faster-rcnn-resnet50-v1-640x640/versions/1)

```
    #ymin, xmin, ymax, xmax
    xmin = float(xmin)
    xmax = float(xmax)
    ymin = float(ymin)
    ymax = float(ymax)
    bboxes_FRCNN.append([ymin, xmin, ymax, xmax])
```

Figure 10. Formatting bounding boxes of Faster R-CNN results

Centernet

```
#importing model
detector = hub.load("https://www.kaggle.com/models/tensorflow/centernet-resnet/frameworks/tensorFlow2/variations/101v1-fpn-512x5
```

Figure 11. Importing pretrained model for Centernet(model url: https://www.kaggle.com/models/tensorflow/centernet-resnet/frameworks/tensorFlow2/variations/101v1-fpn-512x512/versions/1)

```
    #formating bboxes
    xmin = float(cx) - float(width)/2
    xmax = float(cy) + float(width)/2
    ymin = float(cy) - float(height)/2
    ymax = float(cy) + float(height)/2
    bboxes_Center.append([ymin, xmin, ymax, xmax])
```

Figure 12. Formatting bounding boxes of Centernet

Results

Model Performance

Yolo

Image Number	Average IoU	Fails per Real Object	Time Taken
009	0.51	0	2531
030	0.49	0	148
164	0	1	141
…	…	…	…
581829	0.56	1	161
Average	0.48	1.20	157

The Yolo algorithm is one of the fastest with a mean of 157 milliseconds taken to process one image. However, this increase in speed has led to a drop in accuracy as it averages a 0.48 IoU and has 1.20 failed/false detections for each true object in the images.

Efficient Det

Image Number	Average IoU	Fails per Real Object	Time Taken
009	0.32	0	4204

030	0.32	0	354
164	0	10	382
…	…	…	…
581829	0	4	422
Average	0.36	1.56	405

The Efficient Det pretrained model averaged a 0.36 IoU and 1.56 failed detections per true object. It also took 405 milliseconds to process each image making it subpar in accuracy and average in speed.

SSD

Image Number	Average IoU	Fails per Real Object	Time Taken
009	0.43	0.5	2960
030	0	0.5	128
164	0	9	135
…	…	…	…
581829	0.40	0	139
Average	0.42	1.39	150.26

Like Yolo, SSD is on the faster end of object detection models with a mean speed of 150.26 milliseconds to process an image. This too has suffered a drop in accuracy averaging 1.39 fails per true object and a 0.42 average IoU.

Faster R-CNN

Image Number	Average IoU	Fails per Real Object	Time Taken
009	0.46	1	7918
030	0.46	1	1960
164	0	10	1640
…	…	…	…
581829	0	2	1617

Average	0.5	1.83	1642

Faster R-CNN is the slowest algorithm averaging 1642 milliseconds to process an image. It also has a 0.5 average IoU and 1.83 failed/false detected per real object.

Centernet

Image Number	Average IoU	Fails per Real Object	Time Taken
009	0.59	0.5	2673
030	0.28	0	956
164	0.84	1.44	824
…	…	…	…
581829	0	1	760
Average	0.94	1.26	806

With a 0.94 IoU and 1.27 failed/false detections for each true object in the images, Centernet is the most accurate out of the 5 models. It also takes an average of 806 milliseconds to detect the objects in each image making it an efficient algorithm.

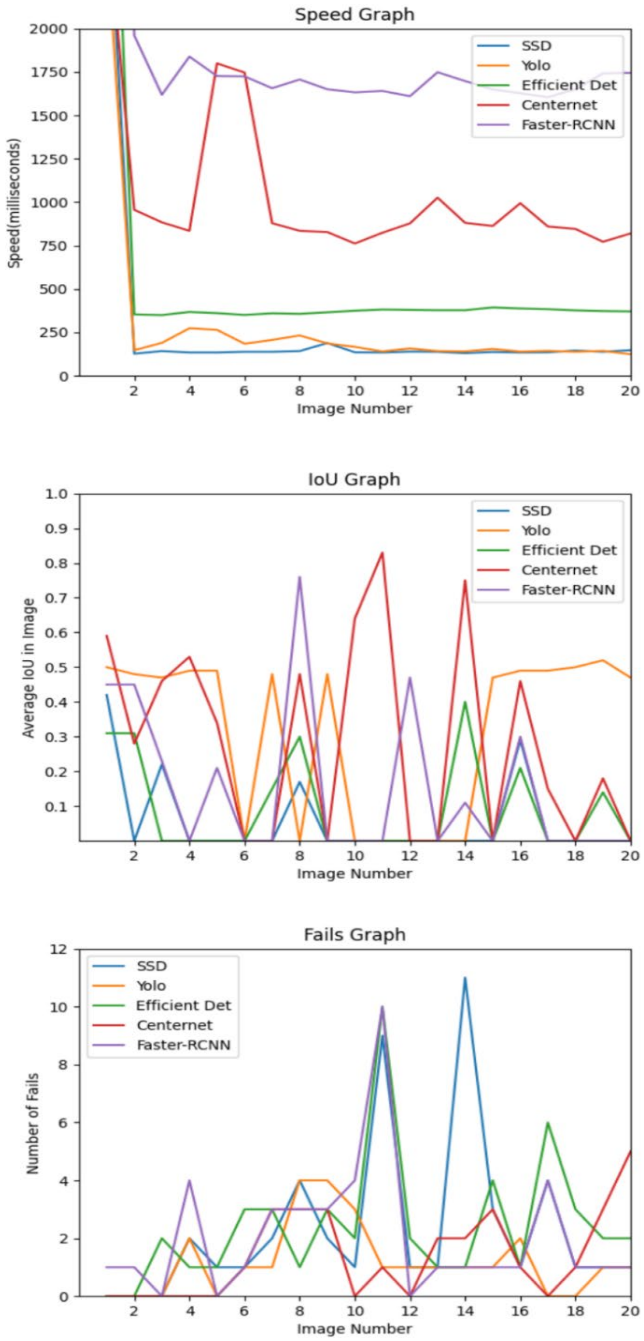

Figure 13. Comparison of failed detections, IoU, and processing speed of each model.

Overall, the majority of the models had around a 0.5 IoU average and 1.5 failed/false detections per true object. This is a drastic difference from the high accuracy commonly seen in these models. The change in accuracy is most likely due to the nature of the objects in the image. During most other benchmarking test cases for the models, the objects that are the subject of detection are clearly visible in the image, however, the objects used in this paper are often partially covered by other items, leading to a decrease in visibility.

	IoU(x2)	Fails(x3)	Speed(x1)	Total
Yolo	5	6	8	36
Centernet	9.5	6	0	37
EfficientDet	3.5	5	5	27
Faster R-CNN	5	4	0	22
SSD	4	5.5	8	32.5

Conclusion

Overall, Centernet is the best model to use when developing an application to aid the visually impaired. It offered high accuracy and moderate speed making it the most ideal model available. Although Yolo was close in score and provided a much better speed for an equal consistency in failed detections, it was unable to accurately locate detected objects. EfficientDet and SSD also have good consistency for failed detections and speed, however they were also had too low of an IoU. Faster R-CNN was an outlier in all categories making it the worst of the five models to use in an application to aid the visually impaired.

Limitations

This paper oversimplifies the process of comparing models as it uses a median value approach to obtain each value in the decision matrix. This method of comparison is able to rank models when there is a large difference between the models, however, it becomes more difficult to rank models as the results become more similar as the weightage of each category becomes more important. There are equations that can be used to find numerical values that accurately represent aspects of models as listed earlier, however, the values produced make it harder to compare models when multiple aspects are taken into consideration. This paper also focuses on one way to use object detection: creating a scanning feature of still images for the visually impaired. This method has specific constraints that were used to grade the models, and these constraints would differ in other use cases.

Future Work

I aim to make use of this research when developing applications to help the visually impaired locate items. I am currently working on a program to count spare change by scanning using a smart phone and this work will enable me choose the best model for that situation.

I also plan on conducting more research to address the limitations of this paper. I want to include real-time object detection in my analysis of the models and find a more concrete method of comparing the data.

Acknowledgments

I would like to thank Mrs. Crandall for acting as my advisor. She provided the encouragement necessary for me to finish this research paper.

References

Brownlee, Jason. “How Do Convolutional Layers Work in Deep Learning Neural Networks?” *MachineLearningMastery.Com*, 16 Apr. 2020, machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

“Centernet/Resnet.” *Kaggle*, www.kaggle.com/models/tensorflow/centernet-resnet. Accessed 20 Feb. 2024.

Cohen, Jeremy. “Finally Understand Anchor Boxes in Object Detection (2D and 3D).” *Welcome to The Library!*, Welcome to The Library!, 2 May 2023, www.thinkautonomous.ai/blog/anchor-boxes/.

Duan, Kaiwen, et al. “CenterNet: Keypoint Triplets for Object Detection.”

“Efficientdet.” *Kaggle*, www.kaggle.com/models/tensorflow/efficientdet/frameworks/tensorFlow2/variations/d2. Accessed 20 Feb. 2024.

“FASTER_RCNN/Resnet_v1.” *Kaggle*, www.kaggle.com/models/tensorflow/faster-rcnn-resnet-v1/frameworks/tensorFlow2/variations/faster-rcnn-resnet101-v1-1024x1024. Accessed 20 Feb. 2024.

Grel, Tomasz. “What Is Region of Interest (ROI) Pooling?” *Deepsense.Ai*, Tomasz Grel https://deepsense.ai/wp-content/uploads/2023/10/Logo_black_blue_CLEAN_rgb.png, 6 Nov. 2023, deepsense.ai/region-of-interest-pooling-explained/.

Keita, Zoumana. “Yolo Object Detection Explained: A Beginner’s Guide.” *DataCamp*, DataCamp, 28 Sept. 2022, www.datacamp.com/blog/yolo-object-detection-explained.

Kundu, Rohit. “Yolo Algorithm for Object Detection Explained [+examples].” *YOLO Algorithm for Object Detection Explained [+Examples]*, 17 Jan. 2023, www.v7labs.com/blog/yolo-object-detection#how-does-yolo-work-yolo-architecture.

“Max Pooling.” *DeepAI*, DeepAI, 17 May 2019, deepai.org/machine-learning-glossary-and-terms/max-pooling#:~:text=Max%20pooling%20is%20a%20downsampling,dimensions%20of%20an%20input%20volume.

“Object Detection Guide - Everything You Need to Know.” *Fritz Ai*, 3 Dec. 2023, fritz.ai/object-detection/#:~:text=for%20object%20detection.-,Basic%20structure,to%20locate%20and%20label%20objects.

Patel, Jagrat. “Top 10 Object Detection Models in 2023!” *LinkedIn*, 27 Aug. 2023, www.linkedin.com/pulse/top-10-object-detection-models-2023-jagrat-patel#:~:text=High%20Accuracy%3A%20Faster%20R%2DCNN,versatile%20for%20different%20use%20cases.

Ren, Shaoqing, et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” *arXiv.Org*, 6 Jan. 2016, arxiv.org/abs/1506.01497v3.

Schumacher, Devin. “Center Pooling.” *SERP AI*, SERP AI, 26 July 2023, serp.ai/center-pooling/.

“Ssd_mobilenet_v2.” *Kaggle*, www.kaggle.com/models/tensorflow/ssd-mobilenet-v2/frameworks/tensorFlow2/variations/fpnlite-320x320. Accessed 20 Feb. 2024.

Thoma, Martin. “How Do Subsequent Convolution Layers Work?” *Data Science Stack Exchange*, 1 Nov. 1961, datascience.stackexchange.com/questions/9175/how-do-subsequent-convolution-layers-work.

Vyas, Kanan. “Efficientdet-A Comprehensive Review.” *Medium*, VisionWizard, 19 May 2020, medium.com/visionwizard/efficientdet-a-paper-review-21918d9a648d.

“Yolov8.” *Kaggle*, www.kaggle.com/models/keras/yolov8/frameworks/keras/variations/yolo_v8_l_backbone. Accessed 20 Feb. 2024.