PROGRESS REPORT: OBJECT DETECTION AND DEPTH PERCEPTION SYSTEM FOR VISUALLY IMPAIRED PEOPLE(AWAREAI)

Rick Lin
Student# 1007977076
rick.lin@mail.utoronto.ca

Ruichen (Spencer) Li
Student# 1007623831
spencer.li@mail.utoronto.ca

Jason Tai
Student# 1008122363
jason.tai@mail.utoronto.ca

Danjie Tang
Student# 1008008941
danjie.tang@mail.utoronto.ca

ABSTRACT

This paper introduces our project, AwareAI, and discusses the progress we have made. Our system integrates deep learning frameworks and a machine learning system with voice output to assist individuals with visual impairments. By analyzing input images, our system generates audio outputs that provide information on nearby objects and obstacles, including their distances. To train the depth perception model, we utilize the NYU Depth V2 dataset, while the MIT Indoor Scenes dataset is utilized for training the object detection model. We compare our proposed deep learning architecture against a baseline model comprising a support vector machine and random forest regression. Our paper presents a comprehensive account of our recent progress, encompassing qualitative and quantitative results. Additionally, it examines the team members' roles and responsibilities, highlighting notable contributions.

—-Total Pages: 9

1 BRIEF PROJECT DESCRIPTION

Our project aims to develop a system that helps individuals with visual impairments utilize their smartphone camera to gain awareness of their surroundings. The system achieves this by providing voice reports on nearby obstacles and their positions relative to the user. For example, the system may announce, "There is a chair approximately 2 meters ahead on your left." By delivering this information audibly, users can enhance their awareness, thereby reducing the risk of collisions, trips, and falls.

Deep learning is an ideal approach for the development of this project due to its high effectiveness in object classification within images. In order to ensure accessibility for as many people as possible, we have chosen to utilize a single camera instead of multiple cameras or LiDAR technology. Despite the limitation of a single camera, deep learning remains the most effective method for accurate depth estimation. Our model takes one picture at a time as input and leverages deep learning frameworks to estimate distances with precision. The architecture of our model is depicted in Figure 1.

2 INDIVIDUAL CONTRIBUTIONS AND RESPONSIBILITIES

Our team has clearly divided the work among team members. Danjie is responsible for data processing, Rick is working on the baseline model, Spencer is designing the depth perception model, and Jason is implementing the object detection and bounding box model. In this section, we will provide an overview of our achievements and outline our future predictions and plans. Detailed information, including deadlines and completion dates, is displayed in later document descriptions. Addition-

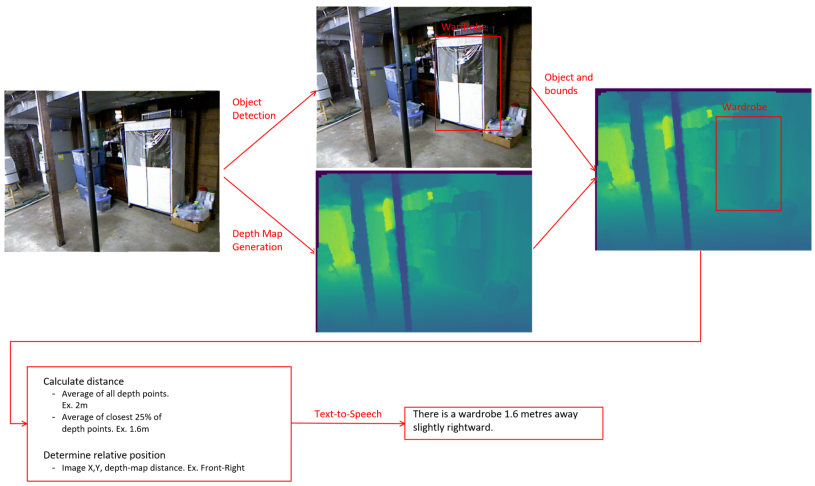

Figure 1: Visual representation of the project architecture. (Nathan Silberman & Fergus (2012))

ally, we provide a short description of individual contributions, highlighting each team member's responsibilities.

2.1 Communication and Project Management

We have maintained regular communication since the formation of our team, holding frequent team meetings on our Discord server at least twice a week. To manage our project effectively, we use a shared to-do list that is updated weekly during these meetings. Additionally, we have created a GitHub repository to store and share our code, and the link to it is provided at the end of this report.

2.2 Data Processing

Danjie is responsible for managing datasets. This includes surveying the field for available datasets, selecting the dataset that best aligns with our project, preprocessing all the data and providing group members with a user-friendly data loader for training. A demonstration of the data loader code can be found later in this paper. Throughout the training phase, we anticipate potential modifications to data requirements. Such modifications may include, but are not limited to introducing or eliminating specific object classes, requests for more data, etc. Danjie is responsible for providing an updated dataset in a timely manner. In the absence of such requests, Danjie's role will shift towards assisting other group members in model training and contribute to the smooth progression of our project workflow.

2.3 Baseline Model

Rick is responsible for conducting background research and implementing a standard machine learning baseline model. The baseline model utilizes a support vector machine (SVM) for obstacle classification and a random forest for depth estimation. We have discussed testing methods and feasibility evaluations for the model. The main framework of our baseline model is nearly complete; however, we require more data and additional training results to further enhance its performance. Additional data training will be conducted starting from July 16th and aims to be completed by July 18th. After completing the training, Rick will help with the implementation of the object detection model and carry out additional testing to ensure the proper functioning of our deep learning model before our next project deadline (Aug 4).

2.4 Object Detection and Bounding Box Model Implementation

Jason is responsible for conducting background research, implementing the object detection and bounding box regression model, writing training/validation/test scripts, and tuning model parameters.

This task proved to be more challenging than initially anticipated, primarily because it required detecting multiple objects within a single image and generating precise bounding boxes for each of them. As a result, this task remains incomplete. However, the background research, architecture, and approach is solidified. We conducted testing on the pre-trained model, fasterrcnn_resnet50_fpn_v2, and determined that it was insufficient for our application. In order to enhance its suitability for our needs, we plan to modify the output layers and retrain the model using the MIT Indoor Scenes dataset Torralba & Sinha (2009). We will also filter out irrelevant classes to optimize its performance.

Although the modified MIT Indoor Scenes dataset required for training the model in this task is not yet ready, our goal is to have this data prepared by July 16. To successfully accomplish this task, we require collaboration and assistance from the rest of the team. Our target is to complete the object detection and bounding box model by July 18. Afterwards, we will tune the hyper-parameters to try to achieve a greater accuracy.

2.5 Depth Perception Model

Spencer is responsible for the background research, implementation, training/validation/test script writing, and parameter tuning of the depth perception model. So far, all tasks have been completed except for parameter tuning. During the background research phase, we studied well-known architectures such as the self-supervised model proposed by Godard et al. (2019) and the DenseMap proposed by Alhashim & Wonka (2019). Ultimately, we chose a classic convolutional neural network with a classifier architecture as the foundation of our model. We implemented two sub-models, namely the coarseNet and the fineNet, inspired by the work of Eigen et al. (2014).

The depth perception model training process consists of two parts: coarseNet training and fineNet training. Initially, our focus is on training the coarseNet, followed by progressing to the fineNet. To train our models, we will utilize the NYU Depth v2 dataset, which contains valuable indoor images with accurate depth information as reported by Nathan Silberman & Fergus (2012). Although we have made some attempts to train the coarseNet recently, we have yet to find the optimal parameters. Once we complete training the coarseNet, we will treat it as a pretrained model, freeze its parameters, and incorporate it into our full model.

2.6 Future Plans and Work Distribution

Upon achieving satisfactory performance with the object recognition, Cbounding box, and depth map models, our next step is to overlay an image's objects and bounding boxes on its depth map. From there, we will compute a reasonable average distance. This could involve methods such as computing the average of the closest 25% points or the average of the closest 25% points to the center of the bounding box. We will further discuss these details in our upcoming report.

Tasks:		Team member:	Deadline:	Done:
Data Preprocessing	Preprocessing scripts	Danjie	Jul 16	
Baseline Model	Implementation	Rick	Jul 10	Jul 10
	Further Testing	Rick	Jul 18	
Object Detection	Architecture Design	Jason	Jul 12	Jul 12
	Model Implementation	Jason / Rick	Jul 18	
	Model Training	Jason	Jul 19	
	Hyper-parameter tuning	Jason	Jul 25	
	Final testing & Visualization	Jason	Jul 27	
Depth Perception	Architecture Design	Spencer	Jul 1	Jul 1
	Model Implementation	Spencer	Jul 6	Jul 6
	Training Scripts	Spencer	Jul 12	Jul 12
	Model training	Spencer	Jul 24	

	Final Testing & Visualization	Spencer	Jul 26	
Model Combining and Audio Output Generation	Model combining	Rick	Jul 24	
	Audio generation	Rick	Jul 24	
	System final test	All	Jul 29	

Table 1: Project timeline

3 Notable Contributions

3.1 Data Processing

Data preprocessing is a prerequisite to all training of deep learning models for practical applications. This holds true for our project as well.

3.1.1 Data Selection

Our project involves training of 2 distinct models, one for depth estimation and another for object detection. Consequently, two separate datasets are required. One specifically tailored for depth estimation and another for object detection purposes. We chose NYU Depth V2 created by Nathan Silberman & Fergus (2012) dataset for depth estimation and MIT Indoor Scenes created by Torralba & Sinha (2009) for object detection and classification.

3.1.2 Primary Objective of Data Preprocessing

Ideally, the 2 data loaders should be highly user friendly, allowing straightforward implementation in actual training processes. The code below are demonstrations of how our data loader can be used in training, they are for depth estimation and object detection respectively.

1. for image, depth_map_pooled, depth_map in dataloader:
2. for image, box, label in dataloader:

Depth estimation As part of data augmentation, we had randomly cropped a 240 by 320 pixels image out of the original image. This cropped image serves as our input to the depth estimation model. For the training of coarse models, the pooled depth map serves as the ground truth label, while the depth_map is utilized as the ground truth label for the refined model. The 3 images below are visualizations of the 3 variables mentioned above. We have generated a total of 2000 image sets to serve as our training dataset.

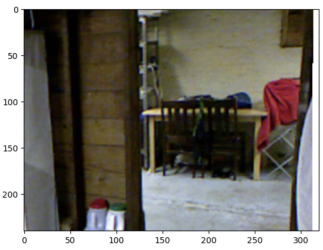

Figure 2: Image

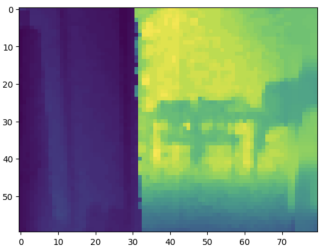

Figure 3: Depth Map Pooled

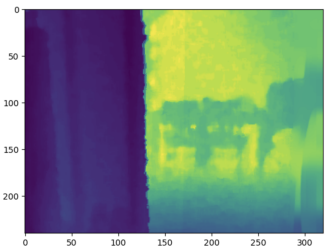

Figure 4: Depth Map

Object detection We have created a training dataset comprising 2736 images sets specifically for object detection. Each image set contains an image, bounding boxes and object labels. The images serve as the input for our object detection model, whereas the bounding boxes and labels serve as the ground truths for object location and object classification.

3.1.3 Data Preprocessing Obstacles

Initially we used the built-in "transforms" module from the torchvision package for image augmentation. This module, however, failed to meet our specific requirements, as the same transformation applied to different images often resulted in different outputs. For example, invoking

"RandomHorizontalFlip" function with p=0.5 sometimes causes the input(image) flipped while the ground truth(depth map) unchanged. Recognizing a need for consistent transformation across different images, we developed our custom data augmentation module. This module guarantees that each image within a pair undergoes the exact same transformation, addressing the issue mentioned.

The object detection dataset was significantly less structured compared to depth estimation dataset. Accordingly, we had taken the following actions to organize data. 1. Remove images missing annotation. 2. Eliminated images containing poorly structured annotations. 3. Generated bounding boxes based on the object polynomial. 4. Create label tensor for each object.

Upon closer inspection, we found that almost half of the images had no annotations. Some other images contain poorly structured annotations with missing closing brackets or keys devoid of values. These unqualified data have been removed from the training set.

In addition, The original dataset contains polynomials with various shapes highlighting the object position which doesn't align with our training requirements. To address this, we create a function that iterates through the polynomial and determines the maximum and minimum x and y coordinates. These coordinates are then used to form the 4 edges of our bounding boxes that represent the location of the object in the image. Also, among 132 object classes of the dataset, most of them are irrelevant to the scope of our project. As we aim to assist visually impaired individuals by providing information about the objects and obstacles in their view, objects such as ceiling, moving people, etc. are deemed nonessential and discarded.

3.1.4 Testing Dataset

We created two testing sets by sampling from NYU Depth v2 and MIT Indoor datasets. Due to the nature of our project, data collection is handy by just capturing indoor images. To augment our testing dataset for final evaluation, each team member will contribute 100 indoor images. This addition ensures a diverse and robust set of data for our final testing phase.

3.2 Baseline Model

To compare the results with our deep learning architecture, we utilized and tested two machine learning algorithms: a support vector machine (SVM) for obstacle classification and a random forest for calculating the depth of a given image. We chose an SVM classification framework as our baseline model for obstacle detection due to its ease of implementation for classification tasks with multidimensional features, such as pixel values. Random Forest Regression was also selected as our baseline model for depth estimation because of its ability to predict non-linear continuous data, such as depth values in an image. Throughout testing, both machine learning algorithms provided reasonable accuracies, which are useful for comparison with our deep learning architecture.

3.2.1 Object Detection Baseline Model

The support vector machine was implemented using scikit-learn and TensorFlow to classify common indoor obstacles. We achieved successful implementation so far by assuming that the image training example datasets contain only common obstacle categories. The images were preprocessed through cropping and augmentation before training the model. In our deep learning architecture for obstacle classification, we implemented it to first take a raw background image for classification, then bound the common obstacles to detect the presence of an object, and finally pass the bounded boxes into our DL classification framework. However, in our ML framework we omit this bounding category due to the difficulty of the SVM's nature and the lack of reasonable choices for other standard machine learning algorithms to detect the existence of an object among common obstacles. Hence, we assumed that it takes image objects that are already classified as common indoor obstacles. The classification baseline model works by first defining the categories we want to classify (Figure 5). In our model, we have classified five categories so far, namely ['bed', 'chair', 'closet', 'couch', 'table']. We will expand our categories to include 30-50 objects for our next project deliverable. For the train-test dataset, we split the data as 80% for training and 20% for testing by randomly selecting the examples. By loading the data categories into our model, we achieved an accuracy of 66.6% for categorizing obstacle images (Figure 6). We also tested a few validation images after training the model. Hyperparameter tuning is also easy to implement. By performing a grid search, we can tune

Figure 5: Support Vector Machine Framework and Hyperparameter Tuning

the hyperparameters to achieve better results for our support vector machine algorithm. We have already tested a few hyperparameters during our model training, which include the following:

- Kernel Function: Linear, polynomial, RBF, or sigmoid.
- Regularization parameter (C): Controls the trade-off between achieving a low training error and maximizing the margin.
- Gamma (for polynomial and RBF kernels): Controls the influence of each training example; a smaller gamma value results in a wider influence, while a larger gamma value makes the model focus more on closer training examples.
- Class weights: Can be adjusted to handle imbalanced datasets by assigning different weights to different classes.

Figure 6: SVM Model Results and a Sample Validation Example

3.2.2 Depth Estimation Baseline Model

To estimate the depth of a given image and detect whether the user will collide an obstacle, we have utilized a Random Forest Regression as our baseline model. Based on the camera's point of view, the depth estimation for common indoor objects should carefully consider the bottom 25% (Figure 7). This is because users are more likely to hit an obstacle in the bottom 25% than in the top 50% of a camera image, such as a ceiling. Therefore, when designing our Random Forest Model, we extracted the maximum depth value of the bottom 25% of an input image. We have achieved a successful implementation by taking background images as inputs, extracting features from the training sets, and predicting the maximum depth values of the bottom 25% to determine whether a user is likely to hit an object. By using preprocessed images before training the model, the random forest machine learning algorithm is able to take raw background image examples and estimate the maximum depth value to avoid user collisions with obstacles. The regression baseline model works by first extracting the features of a raw image using ImageNet-VGG16 proposed by Simonyan & Zisserman (2014). The feature columns (Figure 8) also includes a category that reports the maximum depth pixel value (minimum distance relative to the user) of the bottom 25% of an input image . When working with

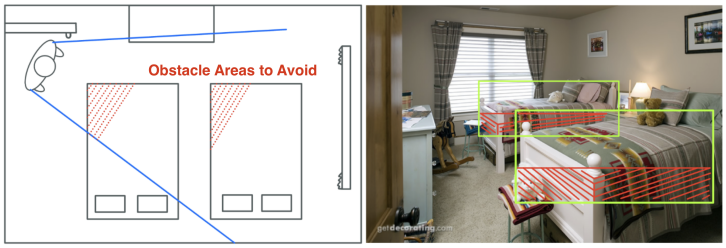

Figure 7: Obstacle Areas to Avoid

small datasets (90 examples), the model reasonably predicts the Root Mean Square Error based on the predicted value and the ground truth. For our next project deliverable, we plan to enhance our model by training it with more than 150+ examples. In our train-test dataset category, we have split the data as 80% training and 20% testing by randomly selecting the examples. By loading the image data features (Depth, Feature1, Feature2, ..., Feature999) into our model, we have achieved an RMS value of 8.947 for the train-test error and 3.782 for the train-validation error. The variation for both errors is relatively high, so we will enhance our training set examples in the next phase of the project to see if the difference between both errors decreases. Hyperparameter tuning is also easy to implement. By performing the regression model, the code shown in Figure 9 determines how we can tune the hyperparameters to achieve better results for our random forest algorithm. We have already tested a few hyperparameters in our model training, including:

- Number of trees (n_estimators): This hyperparameter specifies the number of decision trees to be included in the Random Forest. Increasing the number of trees can improve performance, but it also increases computational complexity.
- Maximum depth of trees (max_depth): It determines the maximum depth allowed for each decision tree in the Random Forest. Limiting the maximum depth helps prevent overfitting.
- Gamma (for polynomial and RBF kernels): Controls the influence of each training example; a smaller gamma value results in a wider influence, while a larger gamma value makes the model focus more on closer training examples.
- Random state (random_state): Setting a random state ensures that the same results can be reproduced when running the code multiple times.

Image	Depth	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8	Feature_9	Feature_10	Feature_11	Feature_12	Feature_13	Feature_14	Feature_15	Feature_16
0	45	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	48	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8988850116729740	0.0	0.0	0.0
2	70	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	12.247492790222200	0.0	0.0	0.0
3	72	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	13.731785774231000
4	61	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.089064598083500	0.0	0.0	0.0
5	58	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	20.687282562259590
6	66	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	9.093318939208980	0.0	0.0	0.0
7	61	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	65	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	72	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	53	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	47	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	75	0.0	0.0	0.0	0.0	0.0	0.0	0.0	15.081829071044900	0.0	0.0	0.0	0.0	0.0	0.0	0.0	15.187735557556200
13	68	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8462308645248410	0.0	0.0	0.0
14	75	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	67	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.994967460632320	0.0	0.0	0.0	0.0	5.679763793945310

Figure 8: Features Extracted using ImageNet-VGG16 (Simonyan & Zisserman (2014))

The model also works somewhat reasonably well when estimating the depth value of the bottom 25% area of a given image, considering the input full-sized image with a set of depth and feature values. From the user's camera point of view, images containing obstacles closer to the camera predict higher depth estimation values than objects that are further away from the user.

Throughout the design and implementation of these ML frameworks, we faced challenges in coming up with reasonable algorithms for both obstacle classification and depth estimation. Bounding boxes for images containing multiple obstacles were difficult to achieve using only machine learning frameworks. Therefore, we decided to use a deep learning architecture to identify the obstacles

```
# Initializing the Random Forest Regression model with X decision trees
model = RandomForestRegressor(n_estimators = 10, random_state = 0)

# Fitting the Random Forest Regression model to the data
model.fit(x_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=10, random_state=0)
```

```
# Splitting the dataset into training and testing set (80/20)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 28)
```

```
print(y_pred)
```

```
[52.8 58.5 58.9]
```

```
print(y_val)
```

```
2    58
3    62
1    57
Name: Depth, dtype: int64
```

Figure 9: Random Forest Hyperparameter Tuning and the Results

and then pass them to our support vector machine. Another issue we encountered while performing regression depth estimations using random forest was the selection of a valid framework. Initially, we planned to implement a support vector machine for depth estimation using continuous value categories, but we realized it wasn't a reasonable approach as it could generate large errors if not classified correctly. Consequently, we switched to designing an architecture that uses random forest for regression models. In the future, we need to be aware of challenges such as the increase in training datasets, which may result in a possible vanishing gradient problem. We also aim to tune hyperparameters to minimize both validation and training errors while selecting the best fit that doesn't overfit or underfit the data.

3.3 Primary Model

Our system consists mainly of two neural networks: one for object detection and bounding box, and another one for depth perception. Jason and Spencer are responsible for the design and implementation, and they have made great progress.

Object detection and bounding boxes: To solve the bounding box problem, we tried the pre-trained model: fasterrcnn_resnet50_fpn_v2. This model was trained on the COCO Lin et al. (2014) dataset to detect objects out of 91 classes and create bounding boxes around them.

When testing this model on an image of the room in figure 10, fasterrcnn_resnet50_fpn_v2 detected many objects that were not useful to our goal such as a cup, a book, and a bird. Not only this, in this test image, a lamp was misclassified as a cup even though the model had $\geq$90% certainty that it was a cup. An entire coffee table in the middle of the room was also not classified with $\geq$90% certainty. (Only objects classified with $\geq$90% certainty are boxed)

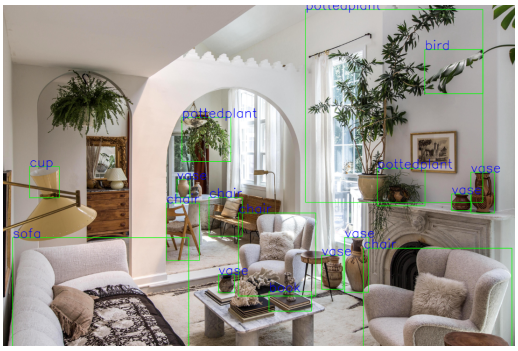

Figure 10: Test Image of room Mendelsohn (2023).

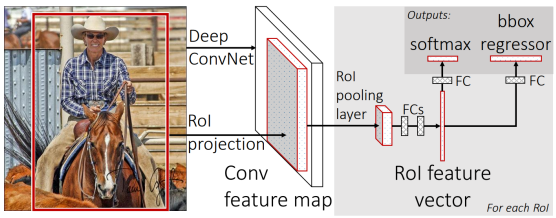

Figure 11: Fast R-CNN architecture Girshick (2015).

In order to retrain with our custom dataset, we will be replacing the 2 output layers of fasterrcnn_resnet50_fpn_v2. There are 2 output layers, one for classes, and one for bounding box predictions. This is shown in figure 11, the softmax for the classes, and the bbox regressor for the bounding boxes. The original dimensions of the pre-trained model are 91 and 364 respectively, shown in Figure 12.

```
(box_predictor): FastRCNNPredictor(
  (cls_score): Linear(in_features=1024, out_features=91, bias=True)
  (bbox_pred): Linear(in_features=1024, out_features=364, bias=True)
)
```

Figure 12: Last 2 layers of Fast R-CNN architecture. Girshick (2015)

We will be changing the size of the cls_score out_features from 91 to the number of classes that are relevant to us, 50, and the bbbox_pred out_features to 200, the number of classes multiplied by 4, creating an output of (x1, y1, x2, y2). We will then start the training and tune the hyper-parameters to try to achieve a high validation accuracy.

Depth perception model: We utilized a Multi-scale Deep Network, modeled after architecture proposed by Eigen et al. (2014), to address the problem at hand. The network comprises two main components: coarseNet and fineNet. The coarseNet takes the input image and generates an initial depth map, while the fineNet takes both the image and the coarseNet's output to produce a full-resolution depth map. The coarseNet consists of 5 convolutional layers and 2 fully-connected layers, while the fineNet comprises three convolutional layers. The depth perception model's architecture is illustrated in Figure 13.

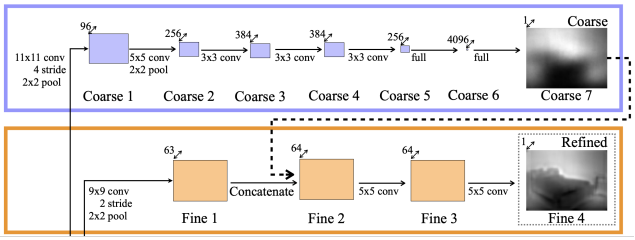

Figure 13: Architecture of our depth perception model

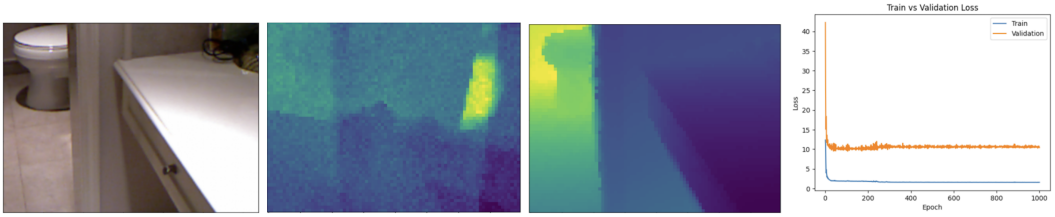

Figure 14: Input image, predicted depth map, ground-truth depth map, and losses (Left to Right)

To enhance training and inference speed, we downscaled the input images from 480x640 to 240x320 and limited the output depth maps' dimensions to 60x80. Our coarseNet has $k = \{11, 5, 3, 3, 3\}$, $s = \{4, 1, 1, 1, 2\}$, and $p = \{0, 2, 1, 1, 0\}$ for conv layers and the fineNet has $k = \{9, 5, 5\}$, $s = \{2, 1, 1\}$, and $p = \{3, 2, 2\}$ for conv layers. We incorporated pooling layers (k = 2) after the first and second convolutional layers of the coarseNet, and after the first convolutional layer of the fineNet. In the complete model, the output of the coarseNet is concatenated with the fineNet's pooling layer output, forming the input for the subsequent stages. The coarseNet encompasses approximately 114 million parameters, with the majority (around 110 million) residing within its fully connected layers. On the other hand, the fineNet comprises roughly 6700 parameters.

Figure 14 depicts the quantitative and qualitative outcomes of the coarseNet. While the coarseNet generates a valid depth map prediction, its performance remains unsatisfactory due to ongoing refinement of the loss function and hyperparameters. Notably, the predicted depth map accurately distinguishes the sink at the front and the toilet at the back. Additionally, the prediction in the bottom right corner is robust. However, it is evident from the displayed loss plot that the model suffers from overfitting. Despite this issue, the model excels in estimating object values in close proximity, as observed in the bottom right corner of the depth map images.

REFERENCES

Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning, 2019.

David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network, 2014.

Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow. Digging into self-supervised monocular depth estimation, 2019.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. *Computer Vision – ECCV 2014*, pp. 740–755, 2014. doi: 10.1007/978-3-319-10602-1_48.

Hadley Mendelsohn. 44 styling tricks that make a small living room feel bigger, Apr 2023. URL `https://www.housebeautiful.com/room-decorating/living-family-rooms/g2310/small-living-room-decorating-ideas`.

Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Antonio Torralba and Pawan Sinha. Recognizing indoor scenes. *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, 06 2009. doi: 10.1109/CVPRW.2009.5206537.

CODE ACCESS

Link to GitHub repository: `https://github.com/Spencer-16/APS360-AwareAI`