

Final Project

# **Mirror Detection from RGBD images**

CM3203 One Semester Individual Project

40 credits

Author: Damjan Dimovski

Supervisor: Dr Jing Wu

## Contents

<b>Introduction</b>	3
<b>Aim of project</b>	3
<b>Background</b>	3
Semantic segmentation	3
Mirror segmentation	4
Mirror Net	4
Progressive Mirror Segmentation	5
Multi-task learning	5
Multi-task Semantic Segmentation and Depth Estimation	5
Datasets	6
MSD Dataset	6
NYUD V2 dataset	6
My dataset	6
<b>My method</b>	7
Single-task learning model	8
Creating the dataset	8
Multi-task learning model	8
<b>Implementation</b>	9
Single-task learning model	9
Visualising the results	12
Multi-task learning	13
Creating the dataset	13
Implementing the model	14
<b>Results and Evaluation</b>	16
Training Loss	16
Visual Evaluation	17
Single-task Model (RGB and Predicted Mirror Mask)	17
Multi-task model (RGB and Predicted Mirror Mask)	18
Single-task and multi-task	19
Discussing the results	20
<b>Future Work</b>	21
<b>Conclusion</b>	21
<b>Reflection</b>	22
<b>References</b>	23

## Introduction

Mirrors appear quite often in image datasets meant for computer vision tasks such as object detectors. They can easily cause ambiguities in computer vision systems that do not consider their existence by considering the objects in the reflection of the mirror to be real. One application where this might be very problematic is in the field of robotics. Imagine a robot navigating through a scene where there is a large mirror. The robot might not be able to differentiate the real scene from the scene reflected in the mirror, resulting in it trying to navigate in the reflection. A way to alleviate this problem is creating a system that will differentiate reflections from real scenes, i.e., a system that can detect mirrors.

## Aim of project

Currently, there exist some methods that perform mirror segmentation in images, and one example is MirrorNet [1]. MirrorNet has been created by utilising only RGB image data.

The aim of this project is to propose a novel model that would integrate depth data, captured using an RGBD camera, alongside RGB data to perform mirror detection, and to see if this would improve the performance over a model that uses only RGB data. Depth data of an RGB image is an image where each pixel relates to distance between the camera and the corresponding object(s) in the RGB image. An example of such data can be seen in Figure 1.

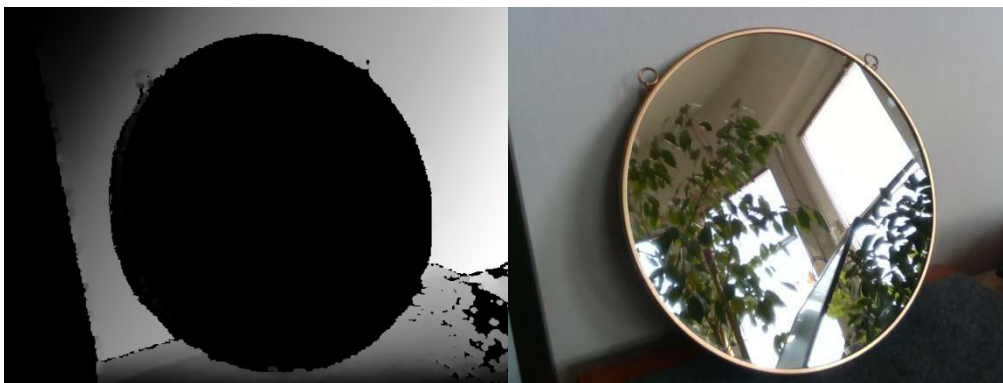


Figure 1. Example depth data for a RGB image

## Background

### Semantic segmentation

Semantic segmentation is a task of per-pixel label classification. It takes an image as an input and converts it into a mask where regions of interest are highlighted [2]. Examples of semantic segmentation can be seen in Figure 2. Semantic segmentation is very useful in computer vision systems as it provides an understanding of the images at a pixel level. It is also considered more advanced when compared to other image-based tasks like object detectors, that identify

objects in the image and label them using a bounding box. Semantic segmentation identifies objects in the image by finding and labelling all pixels that belong to a certain object.



Figure 2. Example of RGB image and its semantic segmentation mask

Source: NYUD V2 Dataset

### Mirror segmentation

Unlike most semantic segmentation model which are used to segment an image into a larger number of segments, i.e., the segmentation mask shown in Figure 2, mirror segmentation masks are constructed by only 2 segments. The white part of the mask is the mirror and everything else is black. An example of such mirror mask can be seen in Figure 3.

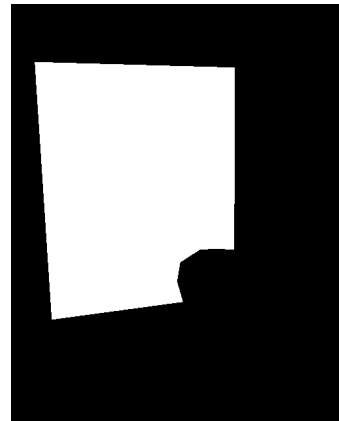
### Mirror Net

One paper that addresses the issue of mirror reflections in object detectors is “Where Is My Mirror” [1]. This paper proposes the model MirrorNet, which like I explained in my initial plan, performs mirror detection by using semantical and low-level colour/texture discontinuities between the contents inside and outside of the mirrors. It uses a fully convolutional encoder-decoder structure, this means that for a given RGB image MirrorNet would be able to output a mirror mask (mirror segmentation) for that image, similar to the mirror mask shown in Figure 3. The difference between my model and this one is that MirrorNet utilises only RGB image data, whereas my model would also utilise depth data from images. The results from the MirrorNet model are very good, which is why I will be using them as a guide of what my model’s output should look like.





RGB image



Mirror Mask

Figure 3. Example of RGB image and its mirror mask from the MSD dataset

#### Progressive Mirror Segmentation

Another paper related to mirror segmentation that also addresses some of the issues with other mirror detection algorithms, including the previously mentioned MirrorNet model, is the paper “Progressive Mirror Detection” [3]. The model proposed in this paper progressively learns the content similarity between the inside and outside of the mirror while explicitly detecting the mirror edges. One issue mentioned with MirrorNet is that it may fail detecting the mirror when the contextual contrasts between the inside and outside of the mirror are not obvious, the paper then goes on to provide an example of such issue and how their model performs better in that scenario. Since the purpose of my method is not to develop the best mirror detection model, rather it is to see whether integrating depth data would improve the performance, I will not need to compare my results with advanced mirror detection models like this one and MirrorNet.

#### Multi-task learning

A multi-task learning model is a machine learning model that is trained to complete two tasks simultaneously. The goal of a multi-task model is to improve generalization performance, in other words to improve prediction accuracy and learning efficiency over a single-task learning model (task-specific model) [4] [5].

#### Multi-task Semantic Segmentation and Depth Estimation

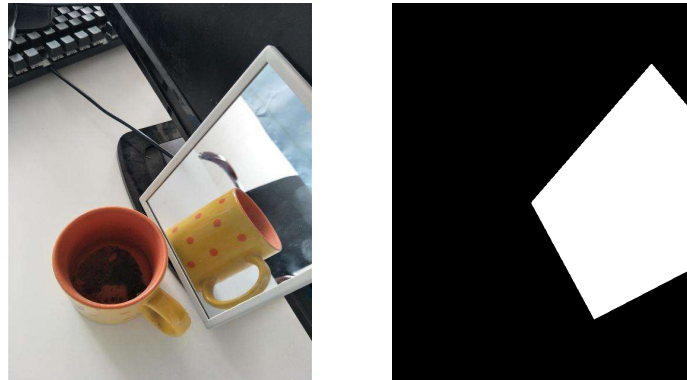
The paper [6] proposes a structure for a multi-task learning model which would perform two tasks simultaneously, these tasks are semantic segmentation and depth estimation. The model would feature an encoder-decoder structure similar to that of MirrorNet, the difference being that this model would have a second decoder which would perform depth estimation. My method would employ the structure proposed in this paper.

The author of the paper [6] provides source code for a *Real-Time Joint Semantic Segmentation and Depth Estimation Using Asymmetric Annotations* model, which can be used for non-commercial purposes.

### Datasets

#### MSD Dataset

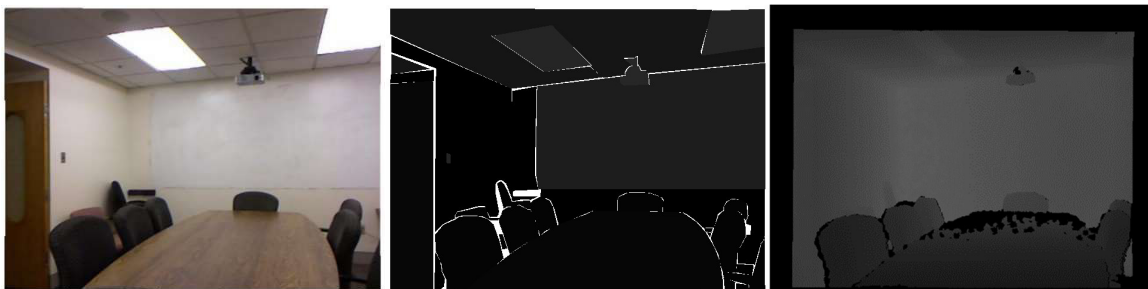
The MSD dataset, which was constructed to train the MirrorNet model in the “Where Is My Mirror” [1] paper, contains 4018 images of mirrors and their corresponding mirror mask. I used this dataset to train my multi-task learning model.



MSD Dataset

#### NYUD V2 dataset

I will also be using the NYUD V2 dataset, which contains 1449 triplets of RGB, segmentation and depth images of different scenes.



NYUD V2 Dataset

#### My dataset

Initially I had the plan to construct another dataset for this project which would contain RGB images and their corresponding depth masks using an Intel RealSense SR300 camera. Due to the novelty of this project and how much time was available I was not able to use this dataset in my project since it was missing mirror masks for each RGB image like the ones in the MSD dataset. I did however construct this dataset and some of the images can be seen in the Figure [4](#).



Figure 4. Example of images from my dataset

## My method

My method involves implementing a multi-task learning model that would integrate the depth data from images alongside RGB data to perform mirror detection. This model would be trained simultaneously to perform mirror segmentation and depth estimation. In my example I am trying to see whether my multi-task learning model for mirror detection would have improved accuracy over a single-task learning model for mirror detection.

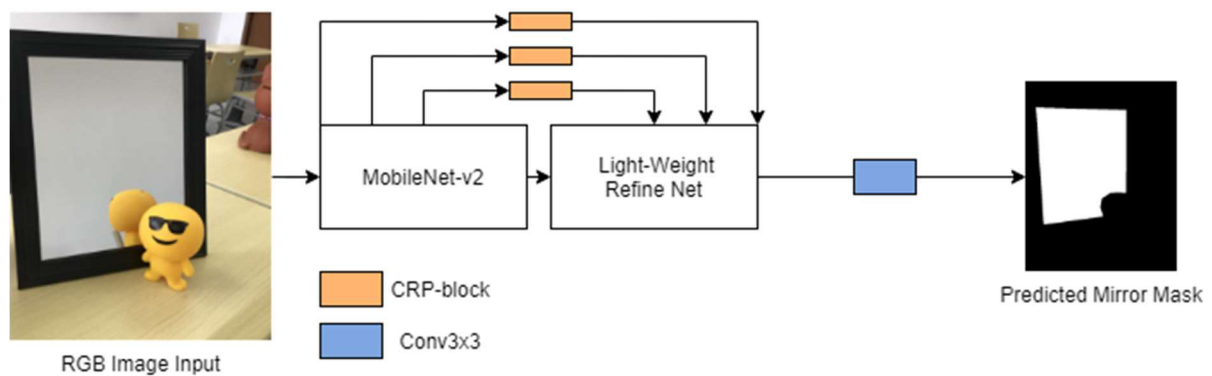


Figure 5. Overview of my single-task learning model structure

### Single-task learning model

To start of I went by constructing a single task learning model that only performs mirror segmentation. I trained this model on the MSD dataset which contains pairs of images and their corresponding mirror mask. The input of this model is an RGB image, and it outputs the predicted mirror mask for the input image. The goal of this part of my project is to see what kind of results I would get from a single-task learning model that preforms mirror segmentation. This is also important because this single-task learning model would be the first branch for my multi-task learning model, and I would be able to compare the results between the two models. For the encoder I am using the MobileNet-v2 [7] network and for the decoder the Light-Weight RefineNet [2] network. The pipeline of the model explained can be seen in Figure 5.

### Creating the dataset

I then moved on to create a dataset that I will use for the training of the multi-task learning model. This dataset contains the RGB images and mirror masks from the MSD dataset in addition to depth images for each RGB image of the MSD dataset. This means that instead of pairs of RGB images and mirror masks like in the MSD dataset, we have triplets of RGB images and their corresponding depth and mirror mask. The depth images are not ground truth depth masks for the RGB images, they are predicted depth masks, since the MSD dataset does not provide depth masks for their RGB images. I gathered these masks by using an existing depth prediction model to get depth prediction masks for the RGB images from the MSD dataset.

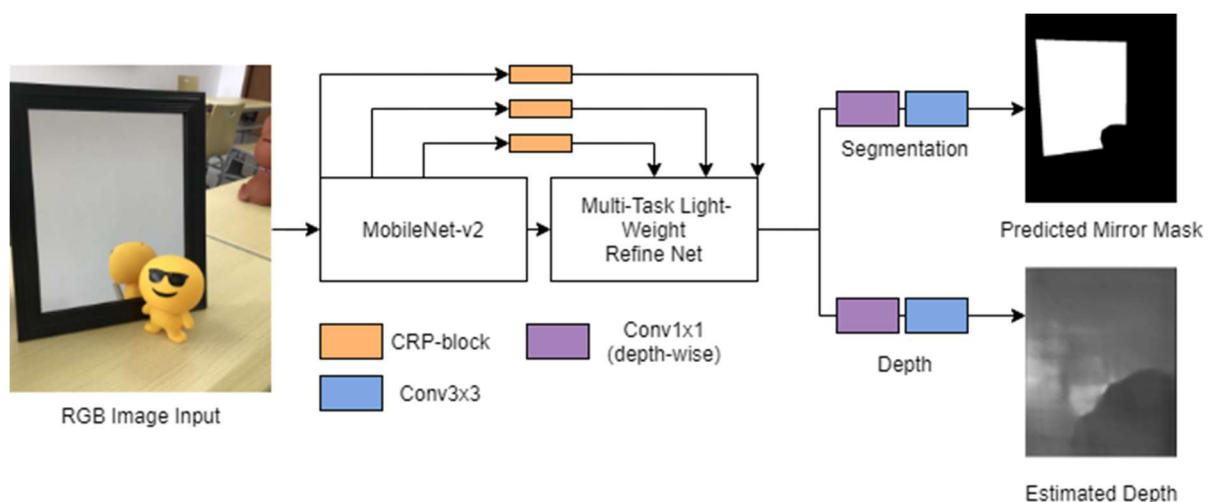


Figure 6. Overview of my multi-task learning model structure

### Multi-task learning model

After creating my dataset, I move on to implementing the final part of the project, the multi-task learning model. For the structure, my model has one encoder and two decoders. For the encoder I am using the MobileNet-v2 [7] network and for the

decoder the Multi-Task Light-Weight RefineNet [2] network, as proposed in the paper mentioned above [6].

I construct this multi-task model by transferring my single-task learning branch for mirror segmentation onto the multi-task learning model provided by paper [6]. I then train this model with the custom dataset I create to get the final results. After getting the results of the multi-task learning model I compare these results to my single task learning model. The pipeline for the multi-task learning model can be seen in Figure [6](#).

## Implementation

### Single-task learning model

The base model for the mirror segmentation was an adoption of the model proposed in the paper [6]. The model given in the paper is for semantic segmentation, so I had to change this architecture of the model to perform mirror detection. The base model before my changes was trained to output a segmentation mask for a given RGB image separated in 40 classes (parts), similarly like the mask shown in Figure [2](#). The mirror mask I am trying to achieve is a separated in two parts, the mirror in the image is a white segment in the mirror mask and everything else is a black segment. An example of the mirror mask I am trying to achieve can be seen in the MSD dataset Figure [3](#). To achieve this, I firstly changed the number of classes in the architecture of the base model from 40 to 2.

```
# Data settings
crop_size = 450
batch_size = 4
val_batch_size = 4
num_classes = 40
n_epochs = 1000
val_every = 5
```

Base config settings

```
# Data settings
crop_size = 450
batch_size = 2
val_batch_size = 2
num_classes = 2
n_epochs = 25
val_every = 3
```

My config setting

I also had to change the dataset that is used for the training of the model. The base dataset provided with the model was the NYUD V2 dataset. I changed the dataset to be the MSD dataset, since I want to use the mirror masks provided in the MSD dataset to train this model for mirror segmentation. Another difference between the dataset I am using for this part (MSD) and the NUYD V2 dataset is that the images from the MSD dataset have one of two resolutions, 640x512 or 512x640, whereas the NUYD V2 hold images that all have the same, 640x480 resolution. This led to some issues for the training part of the model, due to the fact that the base code for the model was expecting an input tensor of the same size for all the images. To

solve this issue, I applied some transformations to the images beforehand. An example of this can be seen in Figure 8. The original code for the model was also using a different encoder and decoder than the ones I planned to use for my model. The code for the encoder (MobileNetv2) and decoder (Light-Weight RefineNet) were also provided alongside the other encoder and decoder that were originally set up to be used for the model structure. I only had to change some parameters, like 'return\_idx' and 'collapse\_ind', in order for the encoder and decoder to work properly. The reason I am switching to the MobileNetv2 and the Light-Weight RefineNet is because I will be using them as well for the multi-task learning model, the only difference is that for the multi-task learning I will be using the Multi-Task Light-Weight RefineNet network instead of the Light-Weight RefineNet. This is important because I want to compare the results between these two models. Changes of the encoder and decoder can be seen in Figure 7.

```
enc = dt.nn.xception65(pretrained=pretrained, return_idx=return_idx)
dec = dt.nn.DLv3plus(enc._out_c, num_classes, rates=enc.rates)
```

Base encoder and decoder

```
enc = dt.nn.mobilenetv2(pretrained=pretrained, return_idx=return_idx)
dec = dt.nn.LWRefineNet(enc._out_c, collapse_ind, num_classes)
```

My changes to the encoder and decoder

```
return_idx = [1, 20]
```

Base parameters used for the  
encoder and decoder

```
return_idx = [1, 2, 3, 4, 5, 6]
collapse_ind = [[0]]
```

My parameters used for the encoder and  
decoder

Figure 7. Changes to the encoder and decoder of the single-task model and their parameters



```
transform_val = transforms.Compose(transform_common)
```

Base transformation for validation

```
transform_val = transforms.Compose(  
    [dt.data.RandomMirror(), dt.data.RandomCrop(crop_size)] + transform_common  
)
```

My changes to the transformation for validation

Figure 8. Example of transformation changes to satisfy the difference of resolution between the images

To start training the model I had to split the MSD dataset into three sections, these are: training, validation, and testing. I followed common dataset ratios and split the dataset such that, 70% is for training, 15% is for validation and 15% is for testing. I split the dataset by following the same method used in the original code. I created three “.txt” files that hold the names of the images that are used for each of the three tasks: training, validation, and testing. The model once the training is initiated, loads the training and validation subsets into a data loader which can be seen in the code in Figure 9.

```
trainloader = DataLoader(  
    dt.data.MMDataset(  
        data_file, data_dir, line_to_paths_fn, masks_names, transform=transform_train,  
    ),  
    batch_size=batch_size,  
    shuffle=True,  
    num_workers=1,  
    pin_memory=True,  
    drop_last=True,  
)  
valloader = DataLoader(  
    dt.data.MMDataset(  
        val_file, data_val_dir, line_to_paths_fn, masks_names, transform=transform_val,  
    ),  
    batch_size=val_batch_size,  
    shuffle=False,  
    num_workers=1,  
    pin_memory=True,  
    drop_last=False,  
)
```

Figure 9. Data loaders for test and validation subsets

I also implemented a function that would store the average training loss of each epoch into a list, which I plot after the training finishes, so I can visualise if the model was trained properly. The training loss should be decreasing after each epoch and finally it should come to a point when it converges. Once it the training loss converges it means that the model is not improving, and the training should finish. This helped me a lot at the beginning of the project since I did not know when I

should stop with the training of the model, i.e., how many epochs to train the model on. The function for plotting the training loss is shown in Figure [10](#).

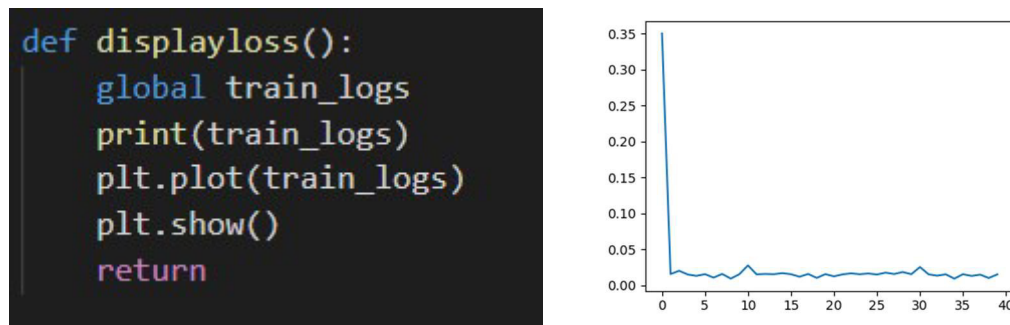


Figure 10. Function for visualising average training loss for each epoch and an example plot

Once I was sure that the model architecture as well as the training were setup correctly, I started with the final training of the model. My single-task model was trained on 25 epochs and validated after every third epoch. I found this to be the optimal setting because after 25/26<sup>th</sup> epoch the training loss tended to converge. Having 25 epoch for the training also meant that the time spent for the model to train and validate was not too long, coming up to roughly 7-8 hours.

### Visualising the results

After I knew that my final model was trained properly, based on the training loss, I moved on to get the mirror mask predictions for the testing subset of the MSD dataset. There was not a code for testing provided alongside the code for the training of the model, so I had to implement this part on my own. From the publicly available code for the MirrorNet [1] model, I got a base structure for the code that I am going to use to get the results from my model. I had to change the architecture of the model to be the same as the one I have for my single-task learning model. This is where I noticed that one part of my single-task learning model is not working correctly. The part that was not working was for loading an already trained model, and in my single task-learning model it was used to continue training from a saved checkpoint, based on the previously saved best epoch, in the case that you need to stop the training of the model and continue it after. Since I trained my model in one go, I did not realise until now that this part was not functioning correctly, and when I tried to use this same code to load my model for testing it was not displaying an error. I was getting results, but they were not the results from my previously trained model, rather they were just results that you would get from the base model with the same architecture, but without any training. This was one of the first setbacks I had in the project, due to the fact that I had to spend a lot of time trying to find problem. I went back to retrain my single-task learning model again because I thought that it was not trained properly and that was the reason for me not getting good results. After finally figuring out what was causing the problem, I implemented another way of



loading a trained model, and the results I got were the correct ones. The changes I made for the loader can be seen in Figure [11](#).

```
start_epoch, __, state_dict = saver.maybe_load(
    ckpt_path=ckpt_path, keys_to_load=["epoch", "best_val", "state_dict"],
)
```

Not functioning loader

```
if resume_from_file:
    if os.path.isfile(resume_file):
        print("=> loading checkpoint '{}'.format(resume_file))
        checkpoint = torch.load(resume_file)
        start_epoch = checkpoint['epoch']
        model1.load_state_dict(checkpoint['state_dict'])
        print("=> loaded checkpoint '{}' (epoch {})"
              .format(resume_file, checkpoint['epoch']))
    else:
        print("=> no checkpoint found at '{}'.format(resume_file))
```

My alternative

Figure 11. Changes I made to solve loading a previously trained model

### Multi-task learning

The next step now was to implement my idea of a multi-task learning model for mirror segmentation and depth estimation. Before implementing this model, I had to construct the dataset which I would use for the training.

### Creating the dataset

Since the idea of this multi-task learning model is to simultaneously perform mirror segmentation and depth estimation, I have to train it using a dataset that would contain RGB images of mirrors and two corresponding masks for each RGB image, a mirror mask, and a depth mask. The closest dataset that has this type of data is the MSD dataset, the only problem is that it is missing the depth masks for the mirror images. To solve this issue, I used an already constructed depth estimation model, proposed from the paper “Deeper Depth Prediction with Fully Convolutional Residual Networks” [8]. I found a publicly available PyTorch implementation of this model which I used to get the depth prediction for all the images in the MSD dataset. The only issue I had was that there was not a pretrained model available, so I had to train this model on my own using the NYUD V2 dataset. I chose to train it on this dataset since the NYUD V2 dataset comes with depth masks for its RGB images. Once I trained this model, I moved on to get the predictions for the depth masks for the MSD dataset. Since the model was trained on the NYUD V2 dataset it was expecting

an input of a 640x480 tensor for the testing, and as I mentioned before the images in the MSD dataset have a different resolution than this one. This mean that I had to some transformations before giving the images as an input into the trained depth prediction model. These changes can be seen in Figure [12](#).

[illegible]

## Base transformations for the testing

```
img = np.asarray(img)
original = img.shape
img = cv2.resize(img, dsize=(228, 304), interpolation=cv2.INTER_CUBIC)
img = img.transpose(2, 1, 0)
input_var = Variable(torch.tensor(img).type(dtype))
input_var = input_var.reshape((1, 3, 228, 304))
```

```
img = cv2.resize(pred_depth_image, dsize=(original[1], original[0]), interpolation=cv2.INTER_CUBIC)
```

### My changes to the testing

Figure 12. Changes I made to solve the issue with the resolution of the images

Firstly, I convert the loaded image into an array using the NumPy library in Python and save the original image resolution into another variable. I then resize the array to match the resolution of the expected input of the trained model. For this I use the cv2 Python library and the function `resize` to make sure that I do not lose a lot of information while resizing the image. I then transpose the image and reshape the tensor that is given as an input to the model to match the expected input from the model. At the end I resize the image back to its original size, so it matches its corresponding RGB image and mirror mask.

## Implementing the model

The final part of the project included combining all the work I had done until now into a multi-task learning model. This is the part that took the most time and was the most difficult to complete. Like with the single-task learning model I started by building upon the base multi-task model provided in the paper “Real-Time Joint Semantic Segmentation and Depth Estimation Using Asymmetric Annotations” [6]. Building upon the base model I swapped the semantic segmentation branch with my mirror segmentation I created for the single-task learning model. I then changed the dataset used for the training to be the MSD dataset combined with the depth prediction masks I gathered in the previous part and changed the image transformations to match those from the single-task model. In this part I encountered the second bigger setback of the project. The multi-task model had error when reading the images from

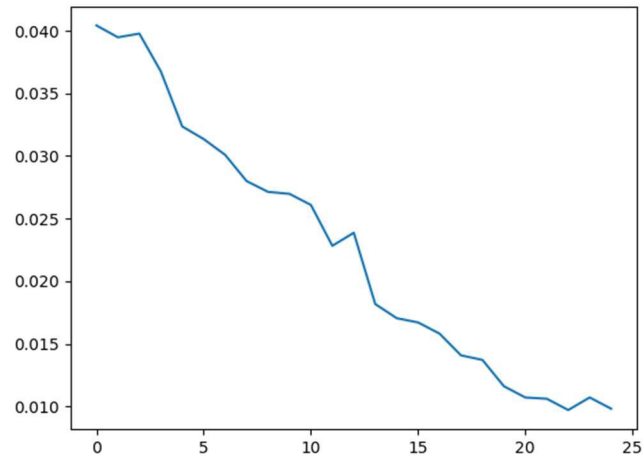
the dataset. At the beginning this was confusing to me since there was not a clear error as to why the model had this problem. The architecture and the loading were the same as the one I was using for the single-task model, which led me to believe that there was some kind of issue with my previous model or the dataset I created, since those were the only two things I altered from the base code provided. To make sure that the problem was not caused by my changes to the model I wanted to try to reproduce the results from the base model without altering anything. This is when I discovered that the same problem was present even without changing anything in the original code. Finally, after some online research I figured out the cause of this problem. The way that the original code was splitting the dataset into three subsets for: training, validation and testing was with having three different ".txt" files which held the names of the images that would be used for the training, validation and testing separately. The model then reads these lists when loading the images into the data loaders that I previously mentioned. I followed this way of splitting the dataset both for the single-task and multi-task learning models from the start to avoid any errors. The error occurred in the multi-task learning model when the original code was incorrectly reading the ".txt" files as it was saving the character "\r" (carriage return) at the end of each line in the file. The code was not showing this error, rather it was not loading some images and crashing at a different point, which was the reason why this error was difficult to notice. After fixing this problem I trained the final multi-task model for my project.

Using the same method for the testing as before I gathered the mirror segmentation predictions for the multi-task learning dataset and moved on to compare the results.

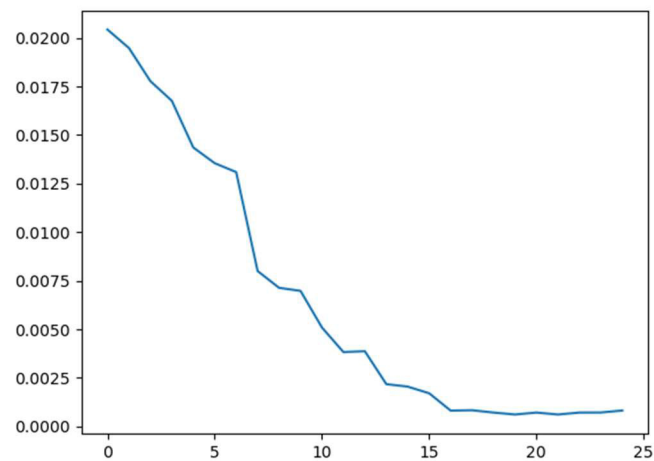
## Results and Evaluation

In this section I will be comparing the results I gathered from both my models on the MSD dataset. Before discussing the results, I want to show the plots of the training loss to show that both my multi-task and single-task models were trained correctly.

### Training Loss



Single-task learning model

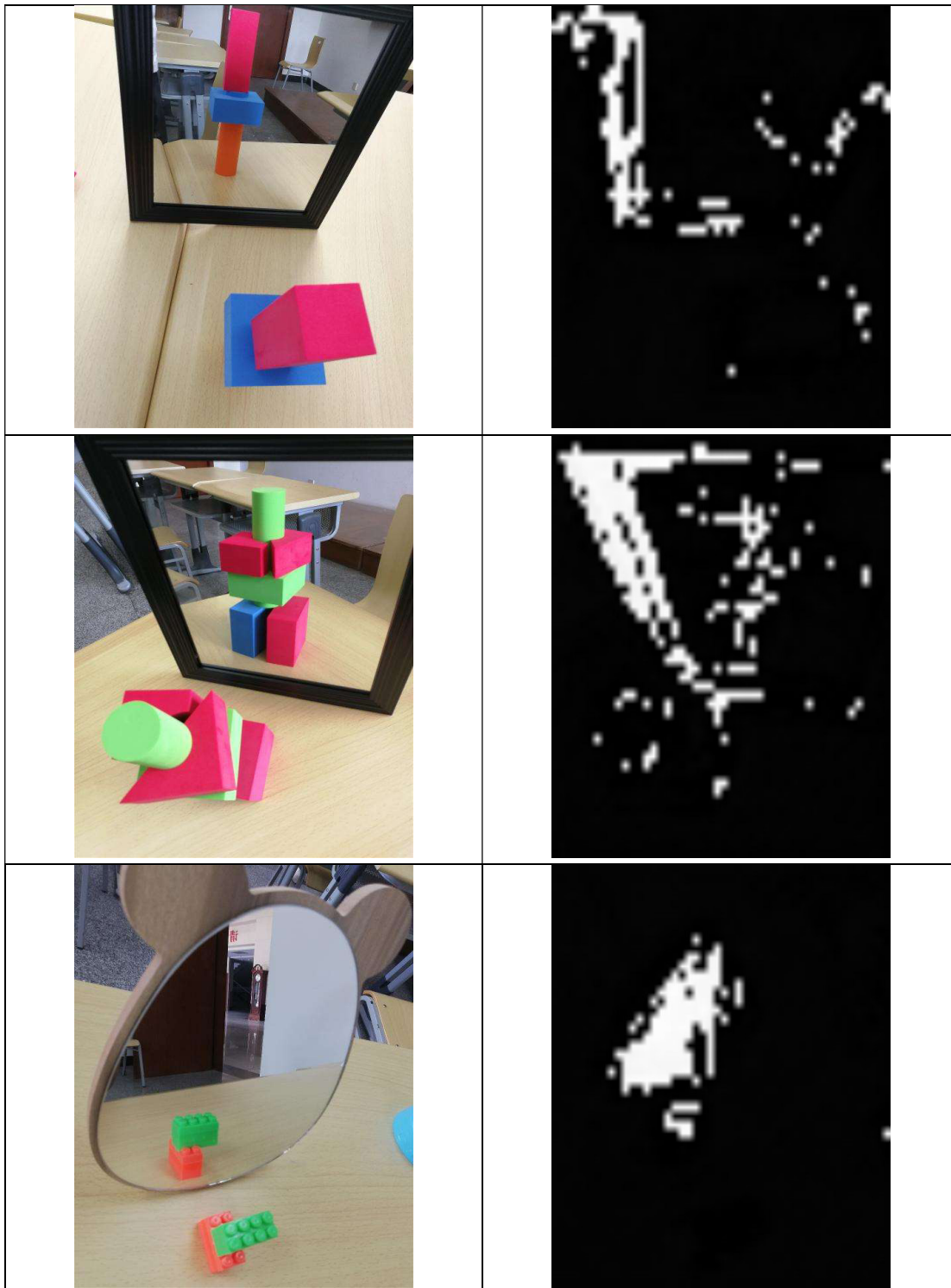




Multi-task learning model

Figures 13 and 14. The training loss for both models, x-axis is the training loss and y-axis is the epoch number





## Visual Evaluation

Single-task Model (RGB and Predicted Mirror Mask)

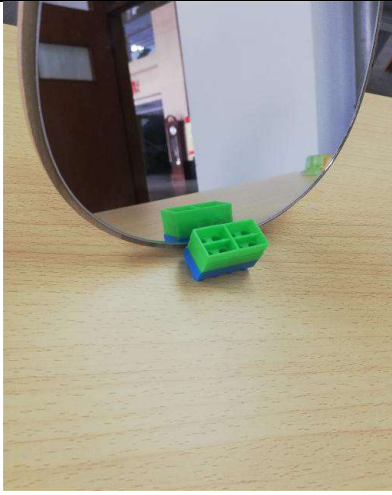





	
RGB Image	Mirror Prediction

Multi-task model (RGB and Predicted Mirror Mask)







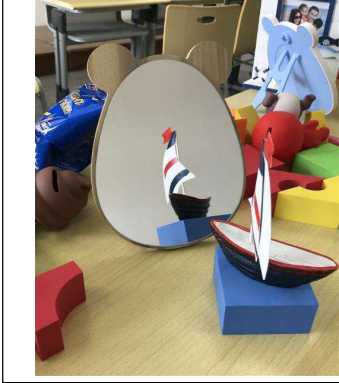


	
	



	
	
<p>RGB Image</p>	<p>Mirror Prediction</p>

Singe-task and multi-task



		
		
		
RGB Image	Single-Task	Multi-Task

### Discussing the results

From all the results shown in the visual evaluation we can see that there is definitely a difference between the predicted mirror masks from the single-task and multi-task learning models.

The results from the single-task model are very poor and struggle to locate the mirror. There is also the problem that the edges of the mirror are labelled with white, and the insides tend to be labelled with black, where it should be the other way around. I can also see that this model tends to be more accurate with the images that contain the rectangle shaped mirror, and it struggles more with oval/round shape mirrors. Although these results are poor, I think this model did better in terms of locating and labelling **only** the mirror in the image, whereas the multi-task model although more accurate tends to also pick up on other objects that are in the scene.



The accuracy in the multi-task learning model is definitely improved over the single-task one in terms of correctly recognizing the edges of objects in the scene. It captures the shape of the mirrors better than the single task one, but it also captures other object outside the mirror and inside of its reflection.

The poor quality of the results is most likely due to the simplicity of the segmentation algorithm and the models altogether.

## Future Work

I think there is definitely lots of unexplored things when it comes to whether depth data would improve accuracy when implemented alongside RGB data in a mirror detection model. In my opinion this hypothesis should be tested with other more advanced segmentation and depth estimation algorithms. This way the results would be clearer, and a better conclusion could be made.

I also think that if I had more time to use my own dataset for the training of the multi-task model (Figure 4), it could be possible that I would have gotten better results. The point of using that dataset was to have the mirror in the depth masks labelled with one colour, the same colour that the edge of the mirror has. This way when the multi-task model performs the depth estimation it would not take into account the objects in the reflection since the mirror would be a plain colour. An example can be seen in Figure 15.

Finally, like I mentioned in the initial plan, another possibility could be to construct a multi-task learning model for segmentation and depth estimation and replace the segmentation branch with an already advanced mirror segmentation model like MirrorNet.



Figure 15. Example of depth before and after hand label

## Conclusion

To conclude I think that the hypothesis I set out to test in the project, “Integration of depth information alongside RGB information in a mirror detection model can improve performance over current mirror detection models that use RGB information only” is true. Based on the results I got from the testing of my single-task and multi-

task learning models, even though they fail to correctly segment the mirrors in images, I can still see an accuracy improvement in the multi-task model over the single-task one. I think that if we test this hypothesis with a more advanced segmentation and depth reconstruction model, we will get much better results.

## Reflection

I can definitely say that I have learned lots of important things, that I can reflect on, while doing this project.

Firstly, I learned that reusing code that is already available and trying to alter it to complete something else can be very difficult. A lot of time goes into actually understanding what the code does in the beginning and this step is necessary if you want to modify the code. Any potential errors that you might encounter whether they are caused from your modifications or were like that at the start cannot be solved if you don't fully understand how the system functions. I spent a lot of time on understanding the code and at the start I did not realise how much time this was actually taking from me being able to complete other tasks that I have set out.

Secondly, since I was not familiar with the topic of deep learning before starting the project, I also had to spend time learning how libraries like PyTorch worked so I can fully utilise them.

Most importantly I think I improved my time management skills. I think this was a very good project for that since I had to always balance between when I am training the models and when I am doing another task. It took some time to get used to planning correctly so I have the model training whilst I am doing other tasks that does not depend on me waiting for the model to finish. I think once I planned my tasks better, I was having much better progress for the project.

Although I had lots of setback, I found them very useful for the development of me and all the development of all the things I mentioned before.

Finally, I would say that I am happy with what I accomplished for the time available, and I think that in the future everything I have did for this project would help me achieve better results.

## References

- [1] X. Yang, H. Mei, K. Xu, X. Wei, B. Yin and R. W. H. Lau, *Where Is My Mirror?*, 2019.
- [2] V. Nekrasov, C. Shen and I. Reid, *Light-Weight RefineNet for Real-Time Semantic Segmentation*, 2018.
- [3] J. Lin, G. Wang and R. W. H. Lau, "Progressive Mirror Detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [4] R. Caruana, "Multitask Learning," *Machine Learning*, vol. 28, pp. 41-75, 01 7 1997.
- [5] M. Crawshaw, *Multi-Task Learning with Deep Neural Networks: A Survey*, 2020.
- [6] V. Nekrasov, T. Dharmasiri, A. Spek, T. Drummond, C. Shen and I. Reid, *Real-Time Joint Semantic Segmentation and Depth Estimation Using Asymmetric Annotations*, 2019.
- [7] A. Howard, A. Zhmoginov, L.-C. Chen, M. Sandler and M. Zhu, "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation," in *CVPR*, 2018.
- [8] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari and N. Navab, *Deeper Depth Prediction with Fully Convolutional Residual Networks*, 2016.