**1. Abstract**

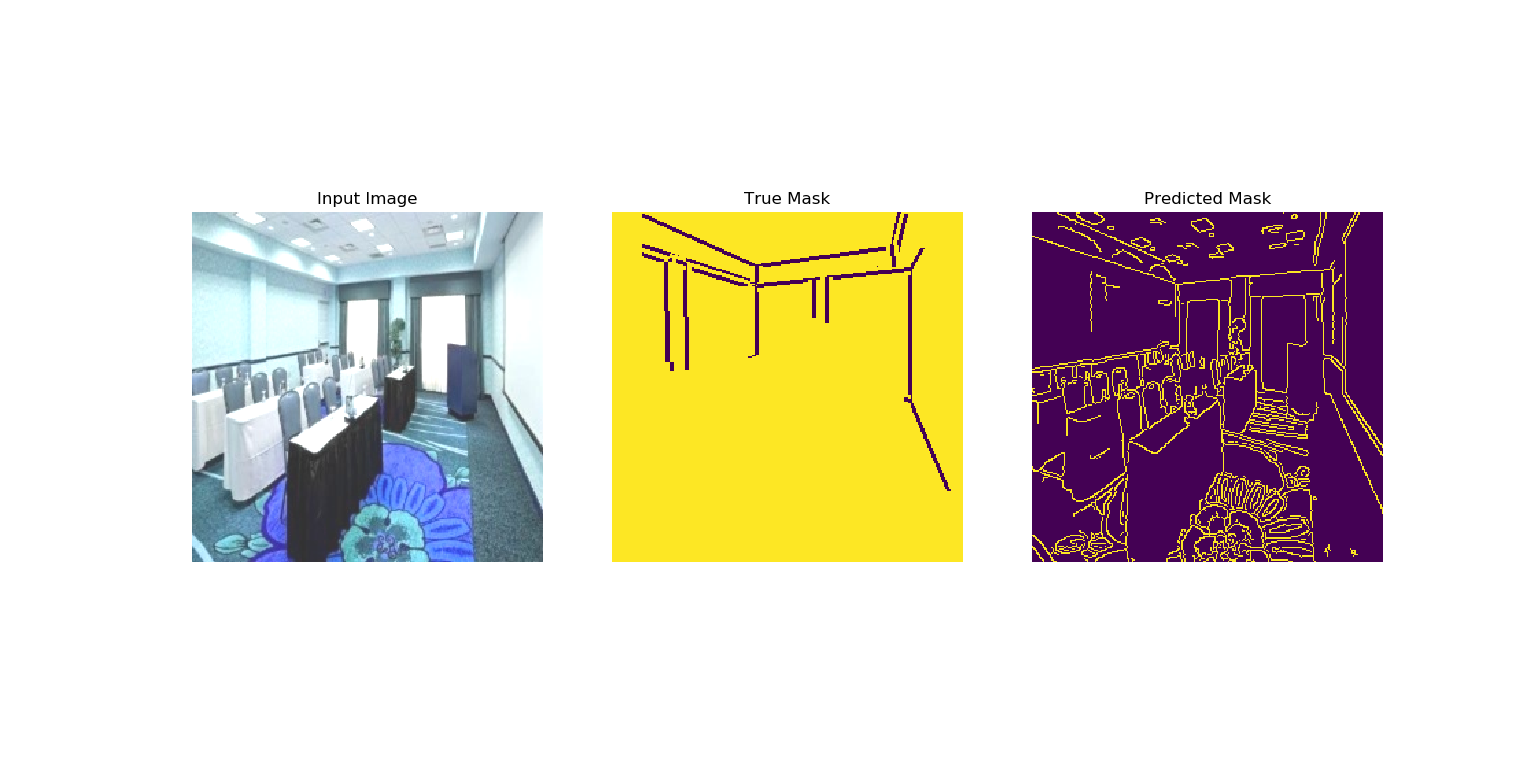
**2. Introduction**

**1.1 Background and Motivation**

This project aims to reconstruct the 3D model of room architecture from photographic images. This has many applications, such as measuring room dimensions and interior design. The process of 3D reconstruction from photographs can be broken down into 4 steps:

1. Extracting the relevant structural edges
2. Determining the line equation of each edge
3. Using the line equations to form a wireframe model
4. Creating the 3D model from the wireframe model

The greatest challenge in this process is the first step – extracting the relevant edges in photographs. Due to the wide variety of noise present in photographs, a deterministic approach to the problem, such as the Canny edge finder (Canny, 1986), proved to be inadequate based on past projects (Figure 1). This is because classic algorithms fail to distinguish between different types of edges and is therefore unable to filter out the irrelevant edges.



*Figure 1 – Results from Canny Edge Detector (OpenCV)*

**1.2 Objectives and Scope**

Due to the limitations of classic algorithms, machine learning is proposed as a possible alternative for detecting structural edges in photographic images. It is hypothesised that a neural network can be trained to extract relevant edges from the noise in photographic images.

Therefore, the objective of this project is to establish whether machine learning is a viable solution to the challenge of extracting structural edges in photographs and, if so, determine the neural network that is most suitable for performing this task.

**1.3 Layout of Report**

This report shall start with a literature review to briefly introduce the concept of Fully Convolutional Networks, as well as some of the existing technologies that were referenced and used for this project. In the following section, this report will discuss the methodology used in every stage of this project, from the data labelling, data augmentation and data processing to the training of each neural network model. The motivation for each choosing the methodology in each step is also discussed. Next, this report will present the results obtained from the neural network training and analyse the significance of the results. The final model architecture is justified based on these results and the final desired output from this model is presented. The report will end off with a summary of the entire project and the future work that can be done to build upon this project.

**3. Literature Review**

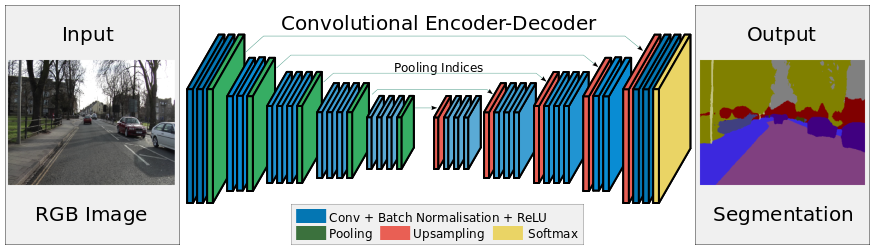
**3.1 Overview**

Recent developments in machine learning have mainly revolved around deep learning and it represents an opportunity for machines to perform tasks that have traditionally been impossible (Marr, 2018). In particular, Convolutional Neural Networks (CNN) and Fully Convolutional Networks (FCN) has progressed significantly, leading to huge improvements in image processing capabilities. FCNs are a natural progression from CNN and they differ from CNNs in their last layer. While CNNs generally have a linear output layer for classifying an entire image, FCNs have a convolutional output layer that attempts to classify each pixel in an image (Jonathan Long, 2014). This has caused FCNs to rapidly gain popularity as it can potentially identify multiple objects in a single image as well as the location, size and shape of each object. This process is known as image segmentation and has applications in many fields, including smart cars.

**3.2 Fully Convolutional Networks**

There are a few notable FCN architectures that are readily available. The more successful ones include SegNet, U-Net, DenseNet, and many more (Shah, 2017). More details regarding some of the architecture is provided below.

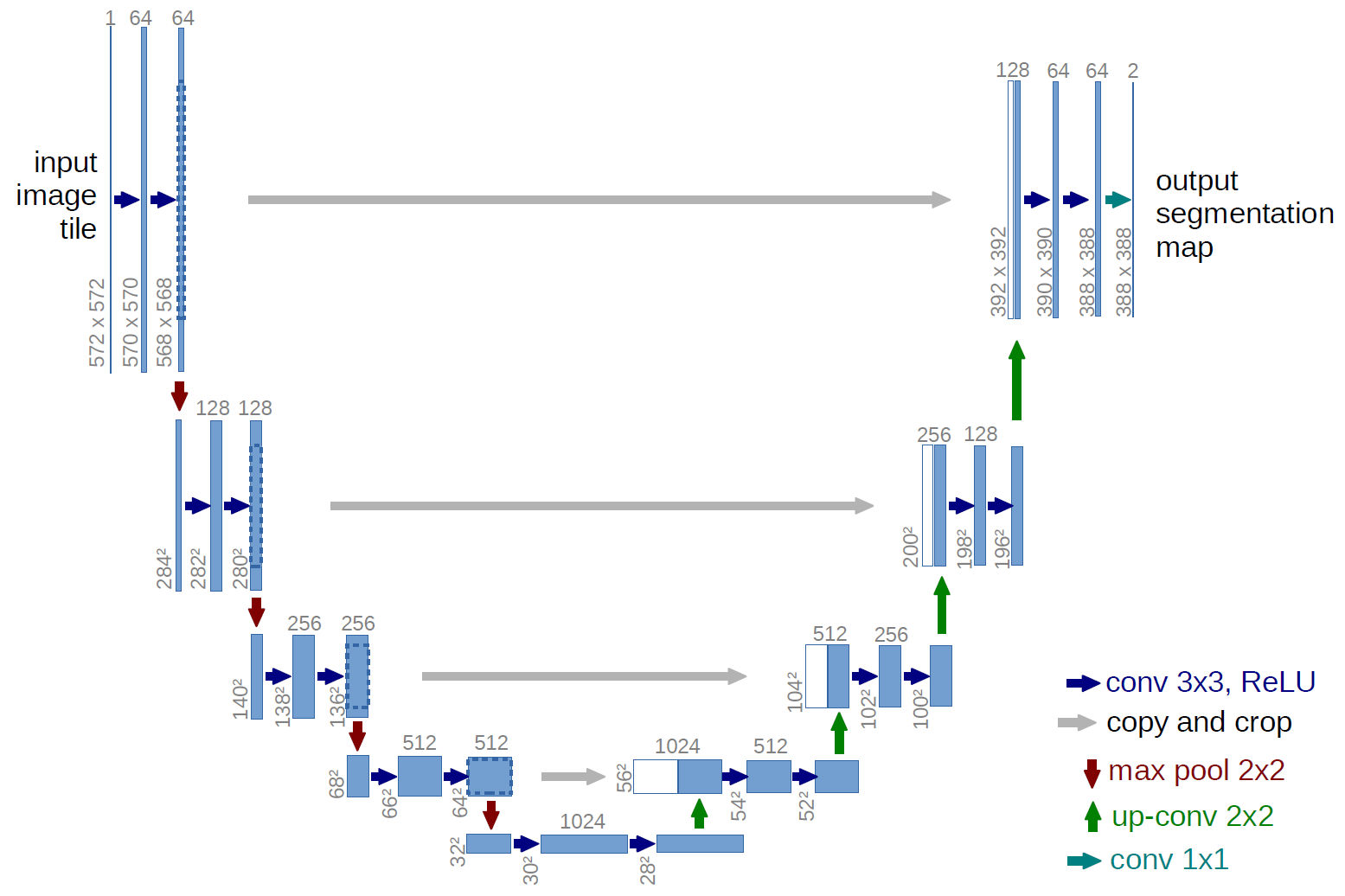
1. SegNet architecture (Vijay Badrinarayanan, 2015)



*Figure 2 – SegNet architecture*

SegNet uses cascading modules of encoders followed by cascading modules of decoders to extract key features from an image. Encoding modules consist of Convolution, Batch Normalization, ReLU (activation function) and Max Pool, while decoding modules consist of Max Unpool, Convolution, Batch Normalization and ReLU.

1. U-Net architecture (Olaf Ronneberger, 2015)

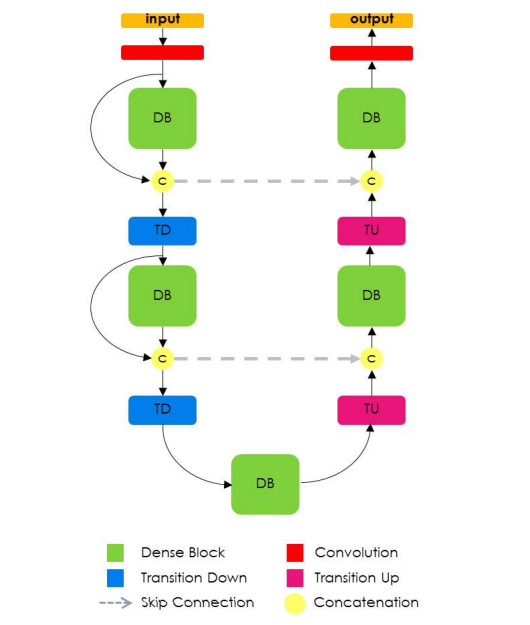


*Figure 3 – U-Net architecture*

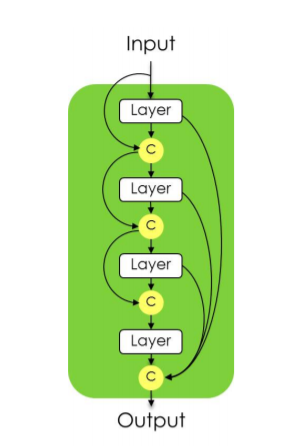
U-Net is similar to SegNet in many ways except 2 key differences. First, it uses Transposed Convolution instead of Max Unpool for upsampling. This creates additional parameters for the model to train on. Second, the output from each encoder block is cloned and concatenated onto the input of each respective decoder block. This allows some features that might otherwise have been lost to be captured by the decoder blocks.

1. DenseNet architecture (Simon Jégou, 2016)

DenseNet is similar to U-net in almost every aspect except for its building block, the Dense Block (Figure 5). The Dense Block is made by concatenating the input with the output of each layer and concatenating the outputs of all 4 layers together at the end.



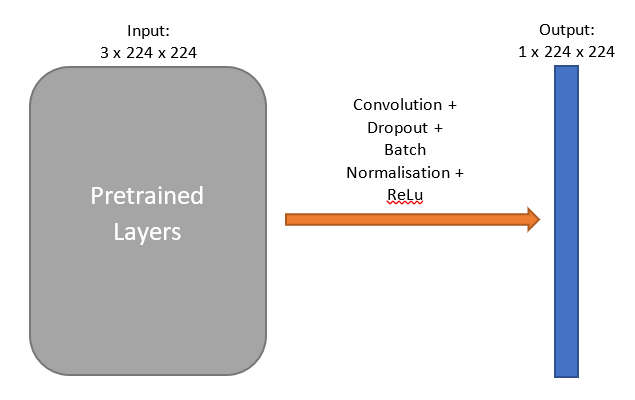
*Figure 4 – DenseNet architecture*



*Figure 5 – A Dense Block in the DenseNet architecture*

1. Transfer Learning using pre-trained FCN-RESNET101

FCN-RESNET101 is a Fully Convolutional Network model with a ResNet-101 backbone. The model has been pretrained on a subset of the Common Objects in Context (COCO) 2017 dataset. The pretrained weights were used, and only the last few layers of the model were modified and retrained for the purposes of this project.



*Figure 6 – Transfer Learning model using FCN-RESNET101* *architecture*

Testing is required to determine which architecture would work best for the purposes of this project since it was difficult to determine at first glance. For practical purposes, only SegNet, U-Net, DenseNet and FCN-RESNET101 were chosen as candidates. In order to implement these architectures, a suitable language and library must first be chosen.

**3.3 Machine Learning Libraries**

Most machine learning programs are written in Python due to its simplicity and the tremendous amount of open source libraries and resources supporting the language. Therefore, Python is the programming language of choice for this project. In recent years, 2 machine learning libraries that are built upon Python have become widely popular:

1. Tensorflow – developed by Google
2. PyTorch – developed by Facebook

While each library has its strengths and weaknesses, Tensorflow was initially chosen for this project as it was the preferred library in the industry and was also well supported with comprehensive documentation and forums (Jain, 2018). However, PyTorch was eventually chosen for the purposes of this project as it was a lot more memory-efficient in implementing the neural network, a condition that was necessary due to hardware constraints. In addition, it was more intuitive than Tensorflow when it comes to implementing non-sequential neural networks, such as U-Net and DenseNet. This will be further elaborated on in the Methodology section.

**3.4 Other Libraries**

This project deals with photographic images and it is inevitable that image processing capabilities are required at various stages of this project. For these purposes, this project tapped on the Open Source Computer Vision Library (OpenCV) for its vast selection of image processing algorithms. OpenCV is BSD-licensed, therefore it is free for businesses and individuals to use and modify its code (OpenCV, 2020).

In addition, this project requires extensive image visualisation as well. The visualisation tool should be capable of interpreting a wide variety of data such as the standard RGB 8-bit unsigned integer format, or a greyscale 32-bit floating point format that is more commonly used in machine learning. As such, Matplotlib, an open source library in Python, was used for this project (Hunter, 2007).

Another tool that was used extensively for this project was Pygame, a library built using Python for users to write video games and is compatible on most systems (Pygame, 2020). It was used to create a custom labelling tool and develop an intuitive Graphic User Interface to simplify the process of data labelling.

**4. Methodology**

**4.1 Overview**

The machine learning model chosen for this project is the Convolutional Neural Network (CNN). CNNs are widely used for image processing due to its ability to extract features from images. The input to the model is a 256 x 256 RGB image and the expected output is a 256 x 256 grayscale image containing the structural edges found in the image.

The process of training the model is divided into three parts:

1. Labelling of data
2. Data Augmentation
3. Data Processing
4. Training the Model

The images that were used were sourced from a scene-centric database used in the LSUN Scene Classification Challenge (Yu, et al., 2015). Approximately 5,000 images from categories “bedroom”, “classroom”, “conference room” and “living room” were selected as the dataset for this project (Figure 7). The rationale for choosing images across multiple categories was to increase the variety of objects in and the layout of the room depicted in each image. This reduces the likelihood of overfitting and improves the model’s ability to recognise rooms that are not covered in the dataset, such as storerooms.

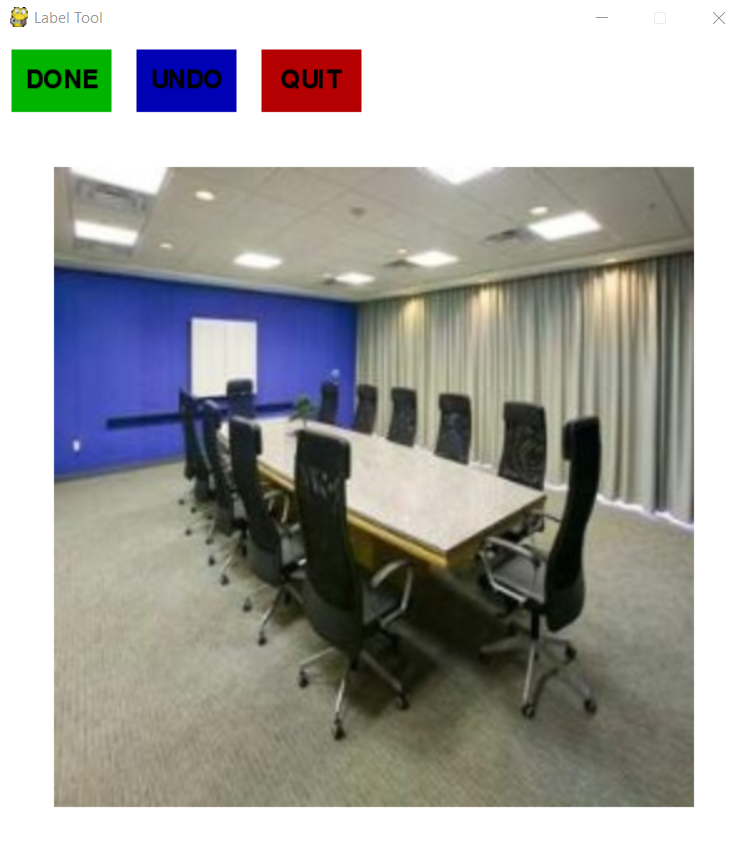
**4.2 Data Labelling**



*Figure 7 – Clockwise from top left: classroom, bedroom, conference room, living room*

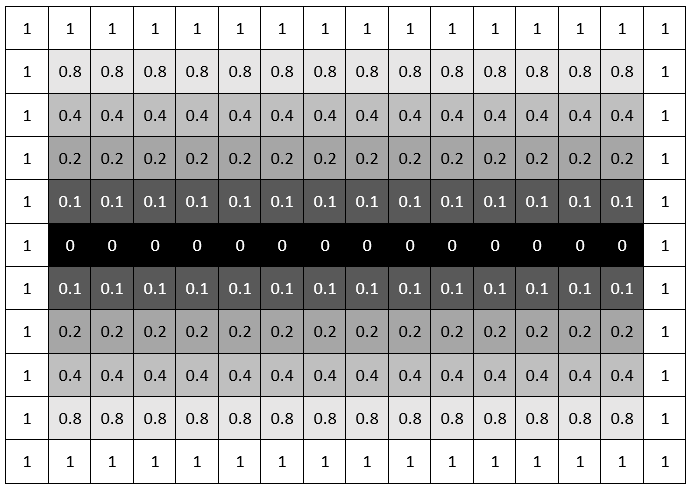
Data labelling was done by tracing structural edges in each image. However, popular online tools such as MIT LabelMe (Russell, Torralba, Murphy, & Freeman, 2008) and Amazon Mechanical Turk (Amazon, 2018) did not have the capabilities suitable for this task. For example, LabelMe only contains polygon tools and is suitable for creating masks over objects (Figure 8). This project, on the other hand, requires masks over edges, hence a line tool would be more suitable. With these considerations, a custom labelling tool was built using Python and the Pygame library (Figure 9).

Essential to every machine learning model is its loss function, which is a measure of the error between the model output and the target value. For this project, the loss function should measure the error of each pixel. Given that edges make up very few pixels in any given image, one concern is that the optimised model might get trapped in a local minimum and generate blank images. In order to prevent this, it is proposed that a larger error can be artificially introduced. One way of doing this is to create a heatmap around each edge (Figure 10). Evidently, the error of a blank model output (an array of ‘1’s) will produce a much larger error, hence reducing the likelihood of such an output. Another proposed idea was to simply thicken the edge from a width of 1 pixel to 3 pixels.



*Figure 9 – Custom Label Tool*

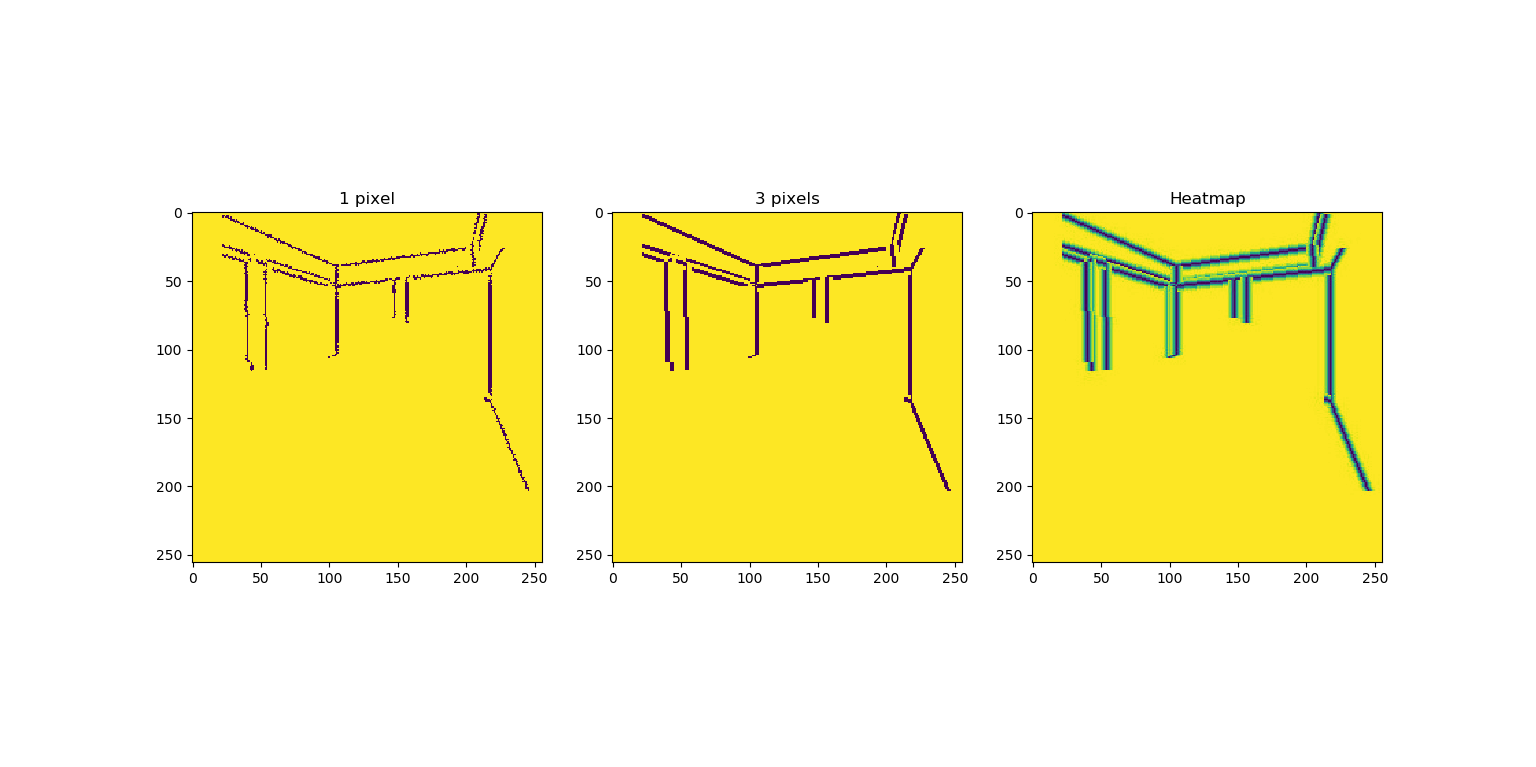
*Figure 8 – Image labelling using MIT LabelMe tool*



*Figure 10 – Example of Heatmap Feature Around an Edge*

Since the labelling tool was custom-built, these modifications to the edges could be easily built into the tool, hence automating this process. Based on the considerations discussed above, a total of 3 different types of edges were used – 1 pixel edge, 3 pixel edge and heatmap edge (Figure 11).

**4.3 Data Augmentation**



*Figure 11 – Various edge types*

While there is no ‘one size fits all’ answer for the recommended dataset size for machine learning, there are some general guidelines to follow. For image-related (Koiran & Sontag, 1998). Research has shown that for image-related neural networks, the size of the dataset should be at least on the order of 104. In order to significantly increase the labelled dataset of 5,000 images, common data augmentation techniques for images were used. Using the Open Computer Vision (OpenCV) library, all images are subjected to three augmentations (Figure 11):

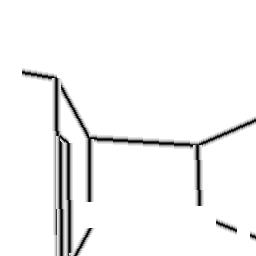
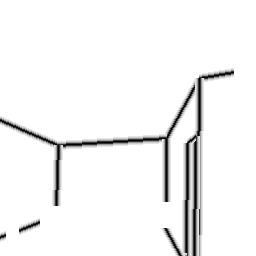
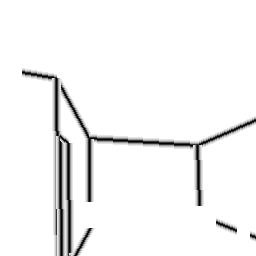
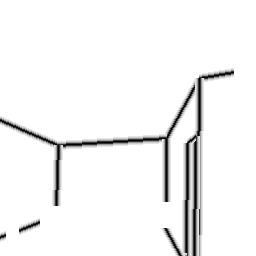
1. A horizontal flip
2. An increase in brightness
3. A horizontal flip plus an increase in brightness

The corresponding labels to the augmented image must also be altered (Figure 13):

1. A horizontal flip to match the flipped image
2. No change since brightness does not affect the location of structural edges
3. A horizontal flip to match the flipped image



*Figure 12 – Clockwise from top left: Original, Flipped, Brightened, Flipped + Brightened*



*Figure 13 – Clockwise from top left: Original, Flipped, Brightened, Flipped + Brightened*

Using these augmentation techniques, a total of 20,000 labelled images is contained within the dataset. The new dataset is then further split in a 7:3 ratio to obtain a train set and a test set. The 14,000 labelled images in the train set is used to train the various neural networks, while the 6,000 labelled images in the test set is used to validate the neural network and prevent overfitting.

**4.4 Data Processing**

Each RGB input image to the neural network is then further normalized and standardized. This is common practice in training neural networks as it is beneficial to ensure that the entire dataset has a standard normal distribution (B, 2017). The detailed normalisation for each channel is as such:

Each channel’s mean and standard deviation is calculated based on the training dataset. Each image in the training dataset is assigned a value based on the mean value of its pixels. These values are then used to calculate mean and standard deviation of the entire dataset as shown below:

The final values obtained for each channel is given below:

**4.5 Training the Model**

For each neural network, a suitable loss function must be chosen to measure the accuracy of the model. It was initially assumed that Mean Squared Error Loss (MSE) could be used as a loss function. MSE measures the deviance of the predicted value of each pixel from the true value. However, the training failed to yield any meaningful results. Upon further investigation, it was determined that MSE was only suitable for linear regression models, whereas this project uses image segmentation and is therefore considered a classification model.

As such, a loss function that measures the accuracy of the classification of each pixel would be suitable. To that end, 2 loss functions may be relevant – Cross Entropy Loss and Binary Cross Entropy Loss. Cross Entropy Loss measures the performance of a classification model whose output is a probability value between 0 and 1 (Machine Learning Glossary, 2017). It is frequently used in multi-class classification problems. In PyTorch, this is implemented using the function CrossEntropyLoss and has the following equation (PyTorch, 2019):

Binary Cross Entropy Loss is a special case of Cross Entropy Loss where the classification problem is strictly binary. In the case of edge detection, this is relevant since each pixel is either classified as an edge or not an edge. In PyTorch, this is implemented using the function BCEWithLogitsLoss and has the following equation (PyTorch, 2019):

While both loss functions could theoretically work well for this classification problem, it was discovered that BCEWithLogitsLoss worked much better. The reason for this difference in performance is unknown.

The optimisation function that was chosen for this project was Adam. While there has been some concerns regarding its convergence (Bushaev, 2018), Adam and SGD remain the two most popular optimisers for deep neural networks. The models are trained at a learning rate of 0.001.

With these considerations, the three models based on SegNet, U-Net and DenseNet are trained on the train set for 50 epochs each. Each epoch attempts to fit each data in the train set exactly once. The model with the highest accuracy is then selected for further finetuning. The detailed results are highlighted in the next section.

**5. Results**

**5.1 Training results**

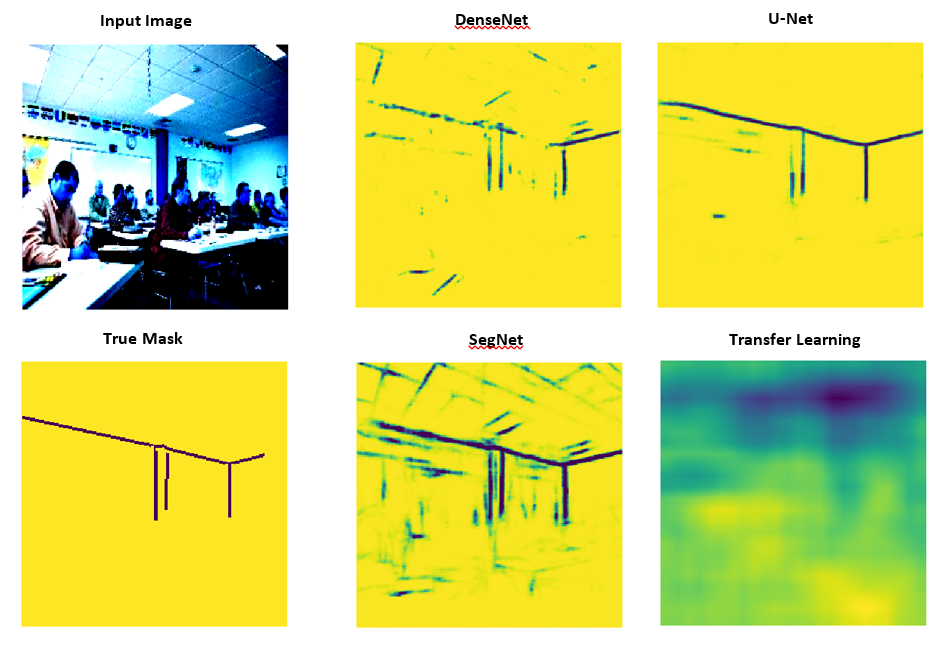
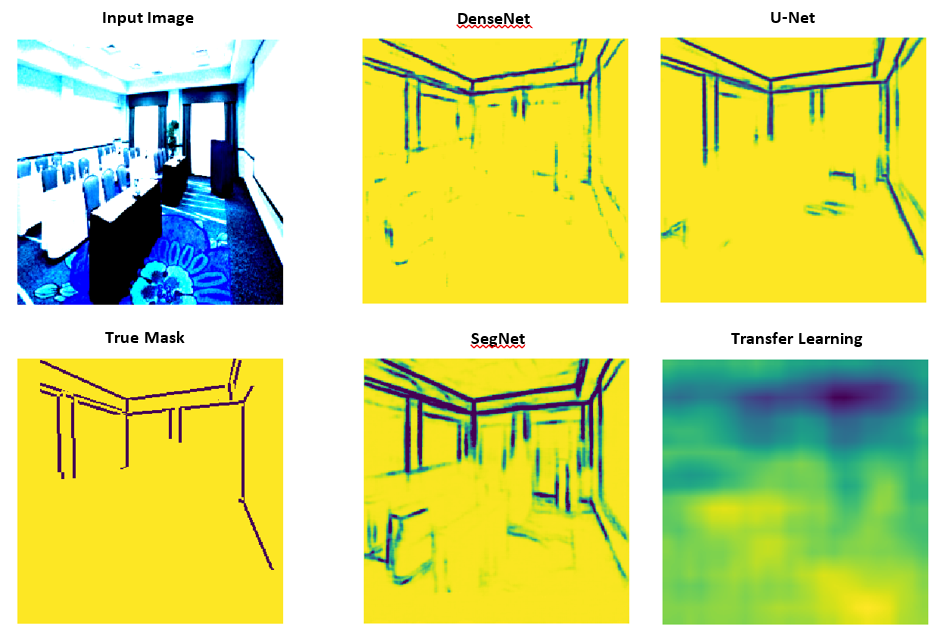
The three models based on SegNet, U-Net and DenseNet were each trained on 3 pixel wide edges over 50 epochs. The accuracy of the model is cross-validated with the test dataset and is summarised in Table 1. The loss is calculated using Binary Cross Entropy, while the accuracy is calculated using absolute error as shown in the following equation:

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Loss** | **Accuracy** | **Parameters Size (MB)** |
| SegNet | 0.0562 | 0.9616 | 8.75 |
| U-Net | 0.0506 | 0.9683 | 29.38 |
| DenseNet | 0.0436 | 0.9740 | 20.14 |
| Transfer Learning | 0.0252 | 0.9998 | NIL |

*Table 1 – Summary of performance of various models*

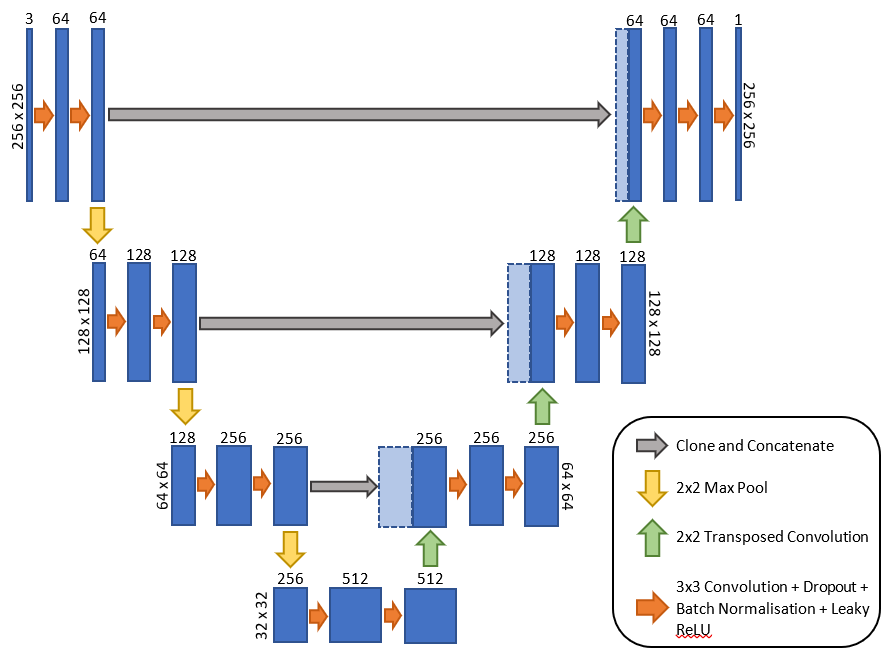
Evidently, SegNet is the most lightweight model, but it is also the least accurate. On the other hand, the transfer learning model based on FCN-RESNET101 scores the highest in terms of both loss and accuracy, while U-Net is the most memory-intensive. Based on these results alone, it would seem the transfer learning model has the best performance. However, closer inspection at the visual output from the various models indicates that models that work well on paper may not be the best. As shown in Figure 14, the output from the transfer learning Model does not contain any recognisable edges despite its stellar performance. In fact, U-Net has the best performance in identifying relevant structural edges.

*Figure 14 – Sample predicted masks from various models*



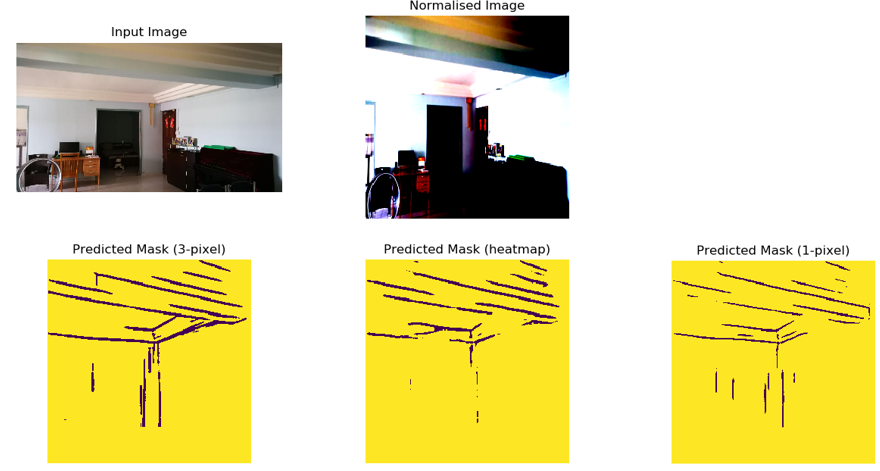
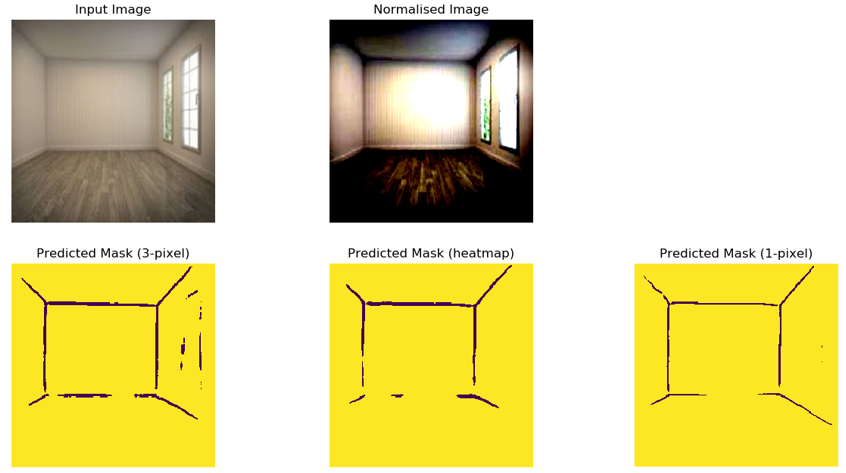
As such, the final model based on U-Net that was used is summarised in Figure 15:

*Figure 15 – Summary of model architecture*



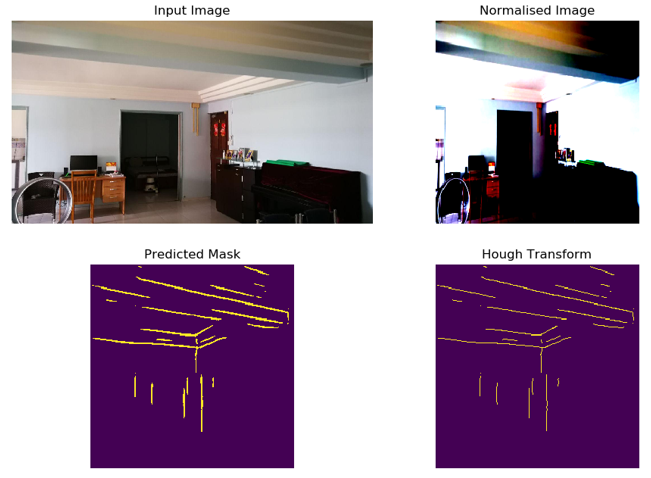
Further investigation was made into the effect of the type of edges used in the quality of the trained model. After training, the output of the model is processed using a threshold value of 0.5 to classify each pixel as either 0 or 1. Some sample results from testing the 1-pixel, 3-pixel and heatmap edges on the U-Net model is shown in Figure 16. From these results, it is determined that the model trained on 1-pixel edges has the best performance.

Using this information, the U-Net model is then retrained on the 1-pixel edge dataset over 50 epochs to attain the ideal number of epochs for the model. The training and testing losses are plotted against epochs to identify the epoch number where testing losses are minimal (Figure 17). This is to prevent any overfitting of the model on the trainset. From the results, the optimal number of epochs is 30.



*Figure 16 – Effect of edge type on model performance*

As shown in Figure 15, the output masks from the U-Net model that is trained on the 1-pixel edge dataset still had significantly thick edges. As such, the edges are first thinned using the Zhang-Suen thinning algorithm and is implemented using the OpenCV library. From there, the line equations of each edge are then obtained using a Probabilistic Hough Transform. The final output is shown in Figure 18.



*Figure 18 – Mask after thinning and Hough Transform*

**6. Conclusion**

**6.1 Summary**

The results derived from this project shows that machine learning, in particular Fully Convolutional Networks, can be used for detecting structural edges in photographic images and it is shown that it performs better than classical algorithms. The difference in performance is especially pronounced in photographs with large amounts of noise.

**6.2 Future Work**

Future work can be done to improve the mode by finetuning some of the hyperparameters such as the type of optimiser and the learning rate. In addition, the current model is not excellent at identifying floor edges since the dataset mostly contains images where the floor edges are hidden by objects. As such, there is a natural bias in the dataset which cannot be corrected easily. To rectify this, the dataset can be expanded in the future to include more empty rooms so that the model can be trained on a more balanced dataset. One way of doing it is to use a CAD software such as Solidworks to create digital renders of empty rooms.

Lastly, some edges in the photographs are partially hidden behind noise such as furniture and people. These edges would therefore be broken into smaller segments. As such, further work needs to be done in developing a robust algorithm that can extrapolate broken line segments to reconstruct the original line such that a wireframe model can be obtained.

# Bibliography

Amazon. (2018). *Overview: Amazon Mechanical Turk*. Retrieved from Amazon Mechanical Turk: https://www.mturk.com/

B, N. (2017, Sep 11). *Image Data Pre-Processing for Neural Networks*. Retrieved from Becoming Human: https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258

Bushaev, V. (2018, Oct 22). *Adam — latest trends in deep learning optimization.* Retrieved from Towards Data Science: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

Canny, J. (1986, Nov). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8*(6), 679-698.

Fung, V. (2017, June 16). *An Overview of ResNet and its Variants*. Retrieved from Towards Data Science: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering, 9*(3), 90-95.

Jain, Y. (2018, Apr 24). *Tensorflow or PyTorch : The force is strong with which one?* Retrieved from Medium: https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4

Jonathan Long, E. S. (2014). *Fully Convolutional Networks for Semantic Segmentation.*

Koiran, P., & Sontag, E. D. (1998, August 18). Vapnik-Chervonenkis Dimension of Neural Nets. *Discrete Applied Mathematics, 86*(1), 63-79.

Machine Learning Glossary. (2017). *Loss Functions*. Retrieved from Machine Learning Glossary: https://ml-cheatsheet.readthedocs.io/en/latest/loss\_functions.html

Marr, B. (2018, October 1). *Forbes*. Retrieved from What Is Deep Learning AI? A Simple Guide With 8 Practical Examples: https://www.forbes.com/sites/bernardmarr/2018/10/01/what-is-deep-learning-ai-a-simple-guide-with-8-practical-examples/#5881c2a08d4b

Olaf Ronneberger, P. F. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation.*

OpenCV. (2020). *About OpenCV*. Retrieved from OpenCV: https://opencv.org/about/

Pygame. (2020, Mar). *About Pygame*. Retrieved from Pygame: https://www.pygame.org/wiki/about

PyTorch. (2019). *Neural Network Docs*. Retrieved from PyTorch: https://pytorch.org/docs/stable/nn.html

Russell, B. C., Torralba, A., Murphy, K. P., & Freeman, W. T. (2008, May). LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision, 77*, 157-173.

Shah, M. (2017, June 1). *Semantic Segmentation using Fully Convolutional Networks over the years*. Retrieved from Github: https://meetshah1995.github.io/semantic-segmentation/deep-learning/pytorch/visdom/2017/06/01/semantic-segmentation-over-the-years.html

Simon Jégou, M. D. (2016). *The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation.*

Vijay Badrinarayanan, A. K. (2015). *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation.*

Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., & Xiao, J. (2015). *LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop.* Princeton University.