**Introduction**

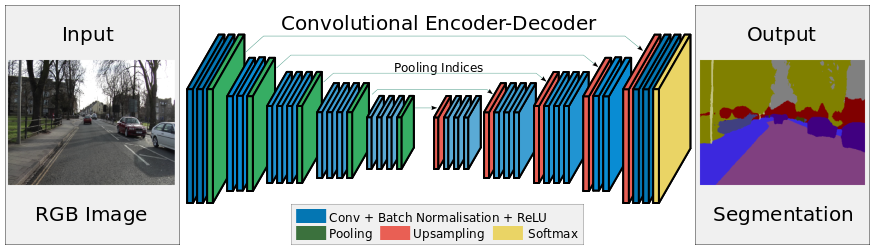
This project aims to reconstruct the 3D model of room architecture from photographic images. This has many applications, such as measuring room dimensions and interior design. The greatest challenge in this process is extracting the relevant edges in photographs. Due to the wide variety of noise present in photographs, a deterministic approach to the problem, such as the Canny edge finder, proved to be inadequate based on past projects. As such, machine learning is proposed as a possible alternative for detecting architectural edges in photographic images. This project therefore aims to establish whether machine learning can have a higher accuracy than traditional algorithms.

**Literature Review**

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.

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.

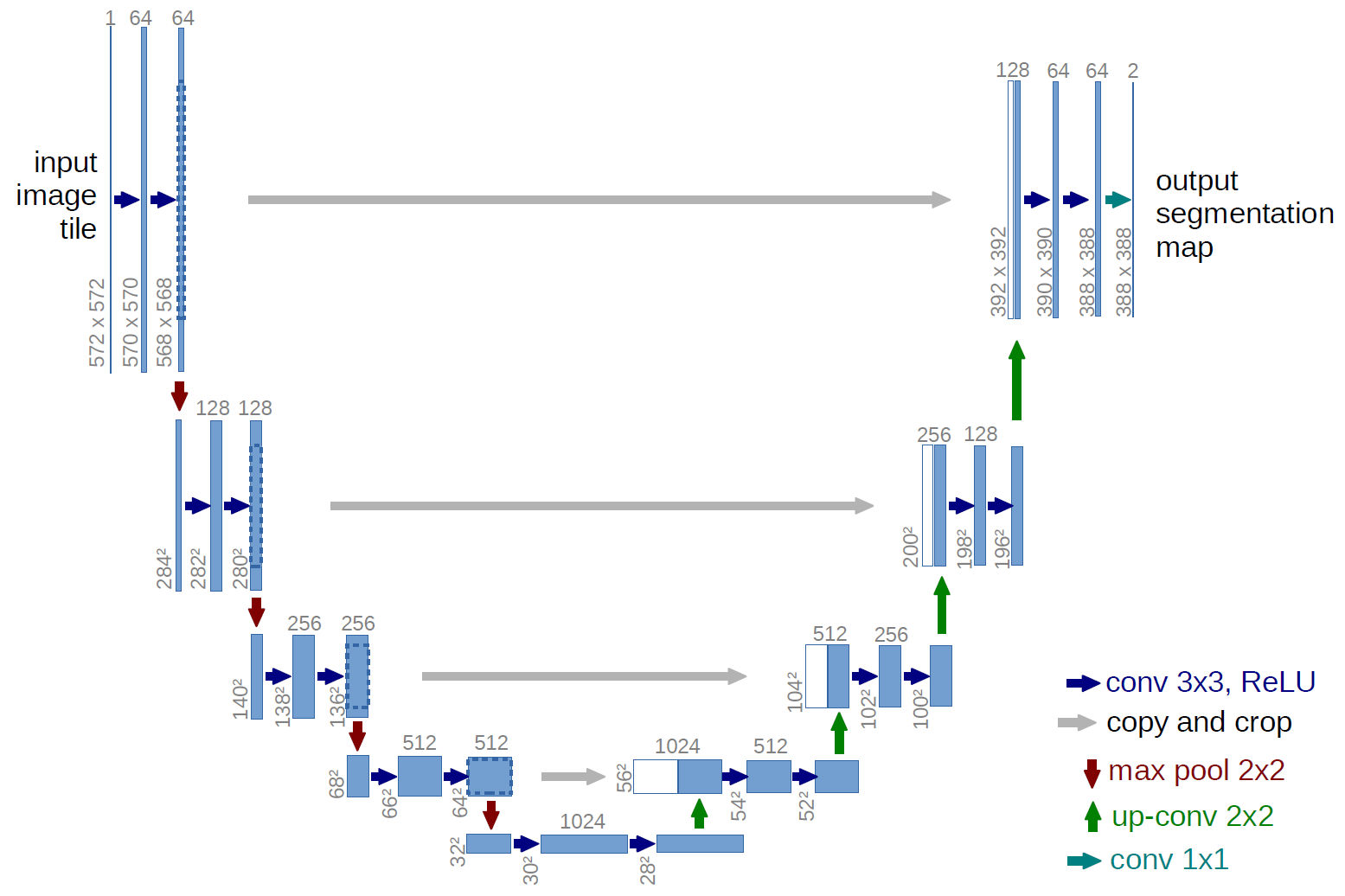
1. SegNet architecture (Vijay Badrinarayanan, 2015)



*Figure 1 – SegNet architecture*

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

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

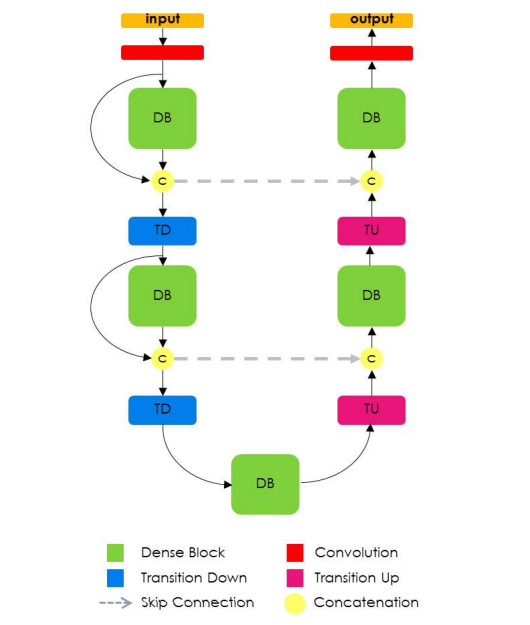


*Figure 2 – U-Net architecture*

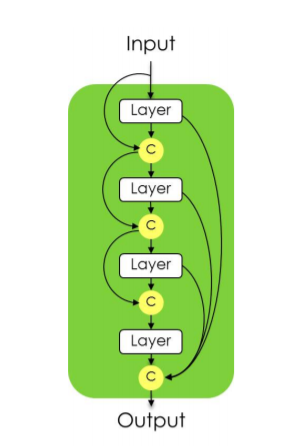
U-Net is similar to SegNet in many ways except 2 key differences. First, it uses Transposed Convolution instead of Max Unpool for upsampling. This creates additional parameters for the model to train on. Second, the output from each encoder block is cloned and concatenated onto the input of each respective 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 (Fig. 4). 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.



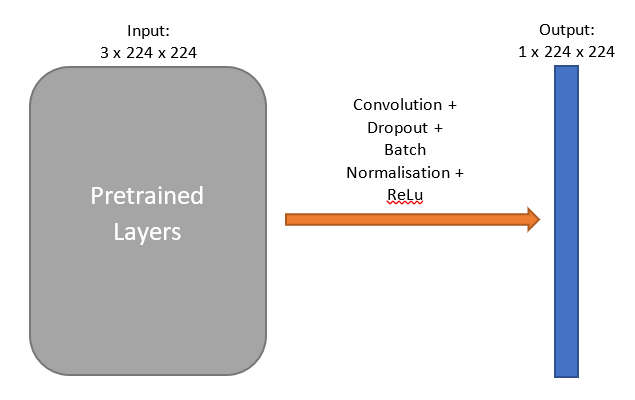
*Figure 3 – DenseNet architecture*



*Figure 4 – 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 5 – 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.

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. In recent years, 2 machine learning libraries have become widely popular:

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

While both libraries have its strengths and weaknesses, PyTorch was chosen 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.

**Methodology**

2.1 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. Model Optimisation

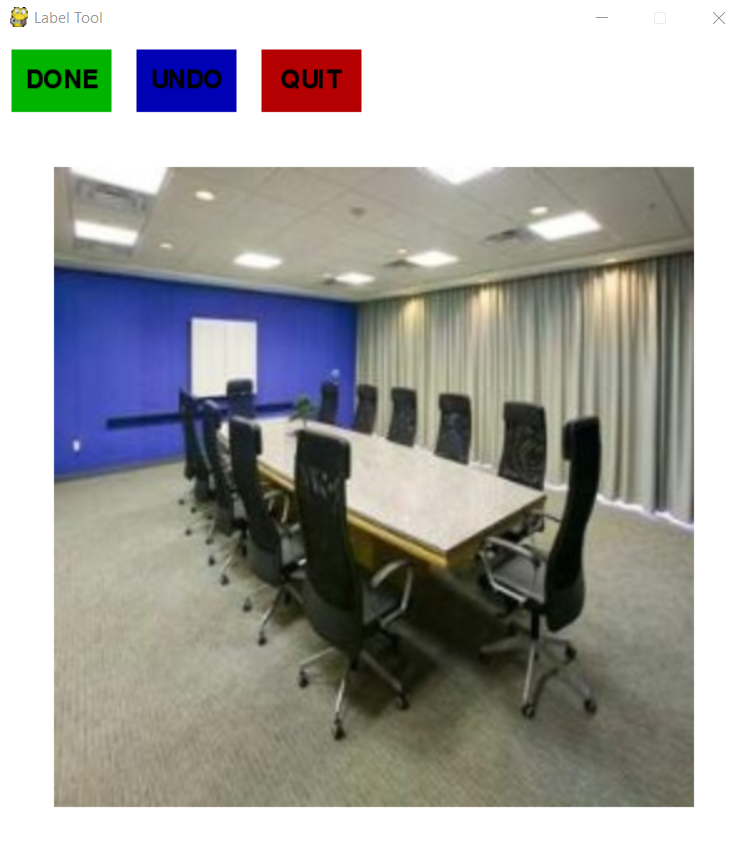
The images that were used were sourced from a scene-centric database used in the LSUN Scene Classification Challenge. Approximately 5,000 images from categories “bedroom”, “classroom”, “conference room” and “living room” were selected as the dataset for this project (Figure 6). The rationale for choosing images across multiple categories was to increase the variety of objects in and the layout of the room depicted in each image. This



*Figure 6 – Clockwise from top left: classroom, bedroom, conference room, living room*

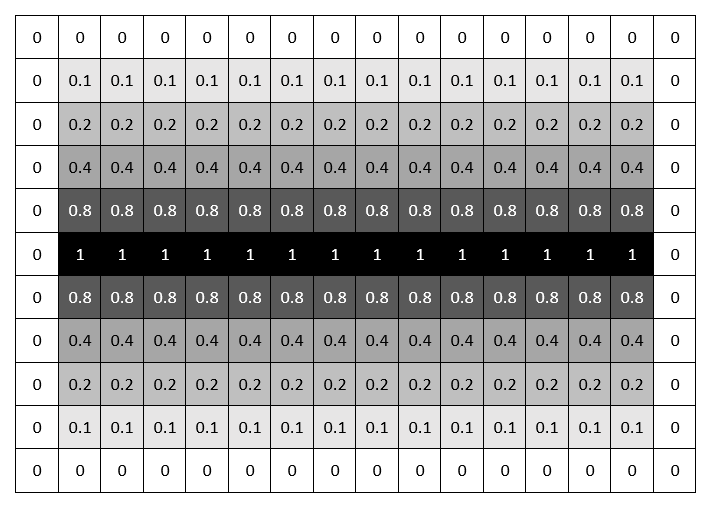
reduces the likelihood of overfitting and improves the model’s ability to recognise rooms that are not covered in the dataset, such as storerooms.

Data labelling was done by tracing structural edges in each image. However, popular online tools such as MIT LabelMe and Amazon Mechanical Turk did not have the capabilities suitable for this task, hence a custom labelling tool must be built. Using Python and the Pygame library, a simple labelling tool was designed (Figure 7).

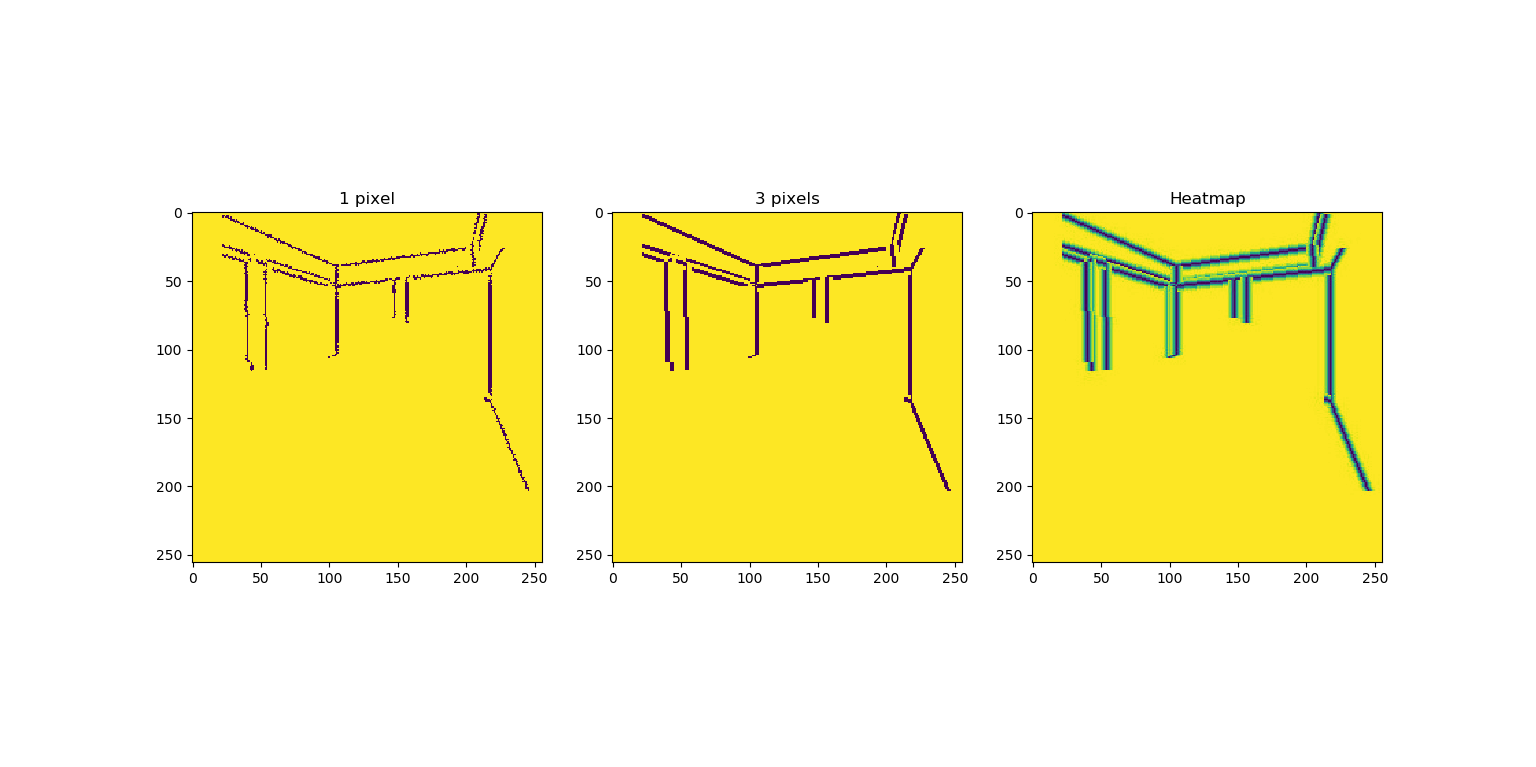


*Figure 7 – Custom Label Tool*

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, a larger penalty must be created. By creating a heatmap around each edge (Figure 8), it not only increases the penalty of generating a blank image, it also allows the model to learn when it is near the coordinates of an edge.

*Figure 8 – Example of Heatmap Feature Around an Edge*

Since the labelling tool was custom-built, this “heatmap” feature could be easily built into the tool, hence automating this process. A total of 3 different types of edges were used – 1 pixel edge, 3 pixel edge and heatmap edge (Figure 9).



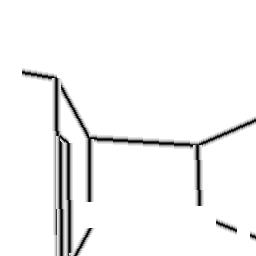
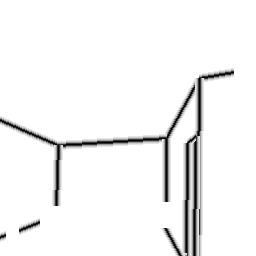
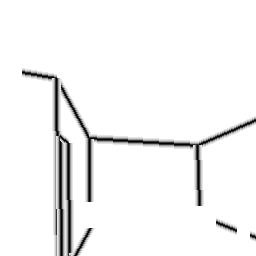
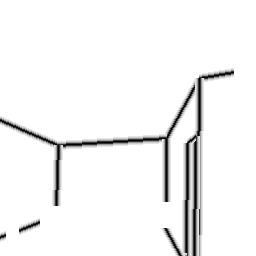
*Figure 9 – Various edge types*

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 library, all images are subjected to three augmentations (Figure 10):

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 11):

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



*Figure 10 – Clockwise from top left: Original, Flipped, Brightened, Flipped + Brightened*

Using the above 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.

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 (Loss Functions, 2017). It is frequently used in multi-class classification problems. In PyTorch, this is implemented using the function CrossEntropyLoss and has the following equation (Docs: torch.nn, 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 (Docs: torch.nn, 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.

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.

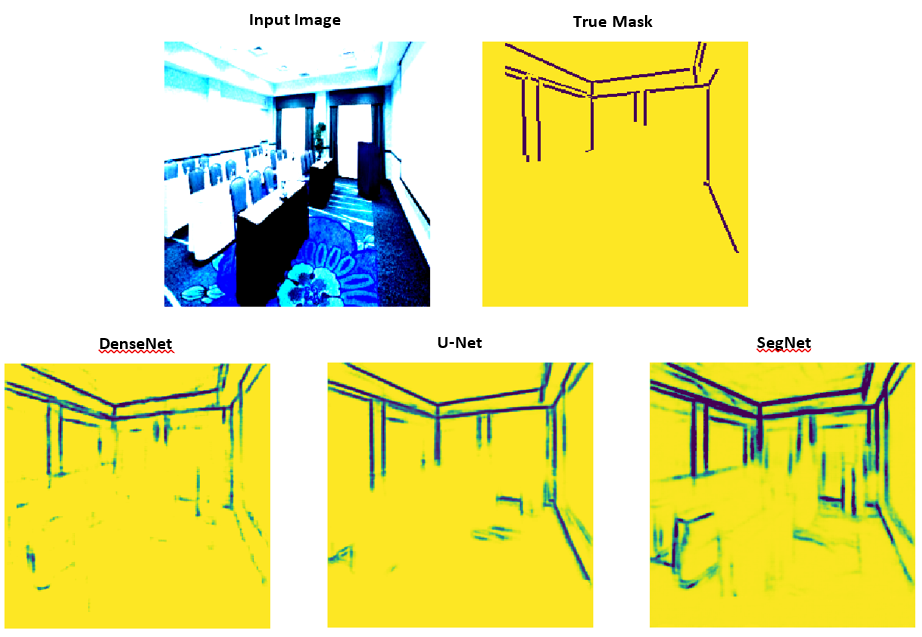
**Results**

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:

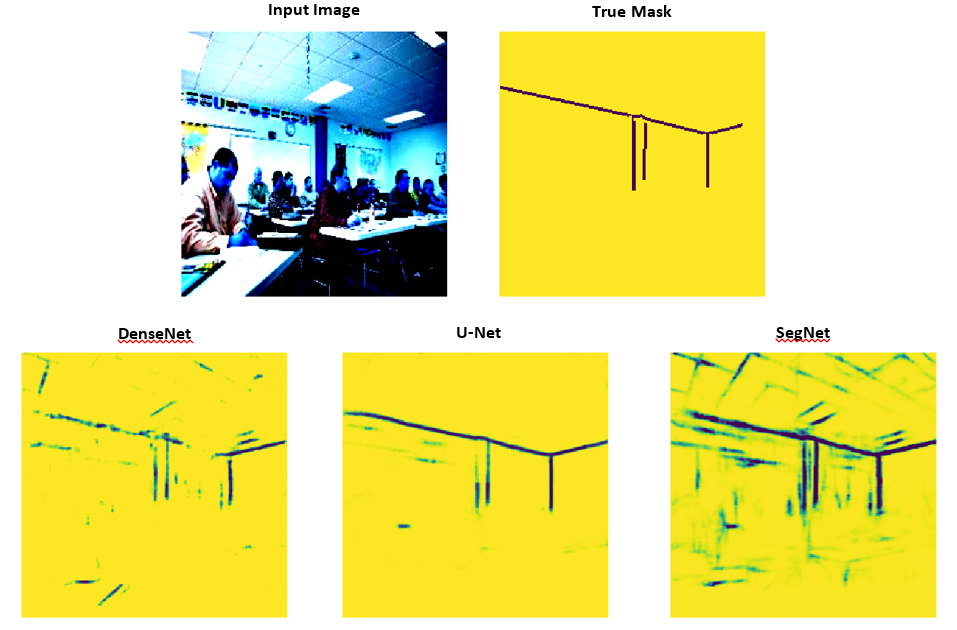
|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Loss** | **Accuracy** | **Parameters Size (MB)** |
| SegNet | 0.0562 | 0.9616 | 8.75 |
| U-Net | 0.0506 | 0.9683 | 29.38 |
| DenseNet | 0.0436 | 0.9740 | 20.14 |
| Transfer Learning |  |  |  |

*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, DenseNet 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 DenseNet is the most suitable model. However, closer inspection at the visual output from the various models indicates that U-Net is in fact better than DenseNet at identifying relevant structural edges (Figure 12). As such,

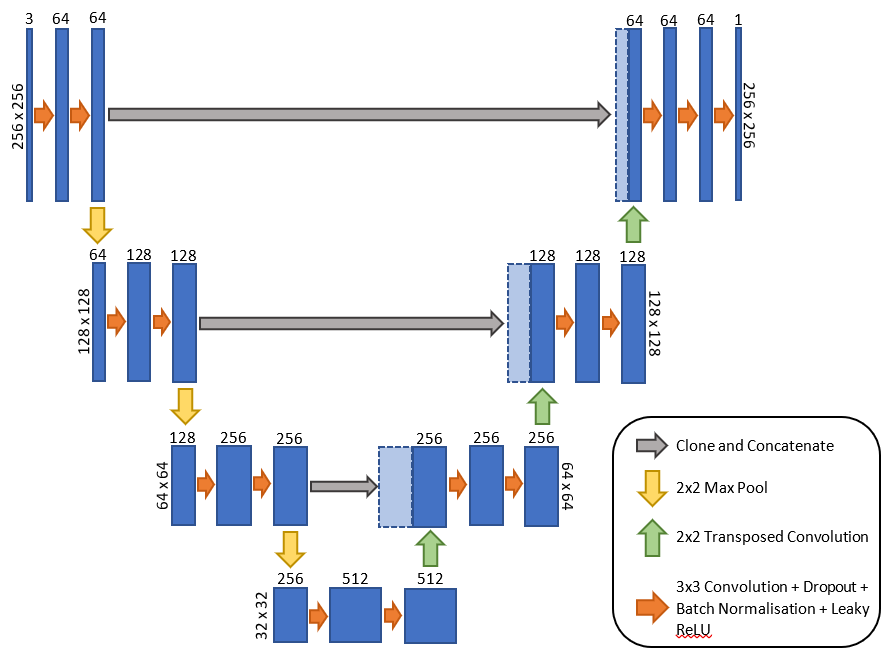


*Figure 12 – Sample predicted masks from various models*



The final model based on U-Net that was used is summarised in Figure 13:

*Figure 13 – Summary of model architecture*



Further investigation was made into the effect of the type of edges used in the quality of the trained model. The results from testing it on U-Net is summarized in Figure 14.

**Conclusion**

The results derived from

# Bibliography

*Docs: torch.nn*. (2019). Retrieved from Pytorch: https://pytorch.org/docs/stable/nn.html

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

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

*Loss Functions*. (2017). 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.*

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.*