Image Segmentation Using Deep Learning

Himanshu Gupta
hg387@drexel.edu

***Abstract*—Image Segmentation has been a very difficult problem for a long time. Traditional methods for image segmentation consisting K-means clustering, Thresholding, Edge Detection, and Histogram based Image segmentation have not been very effective in other environments except from their training environments. Recently, Deep learning Models and Techniques have not only been effective than the traditional approaches, but also produced a general solution covering wide-spectrum of environments. In this paper, this problem of image segmentation is solved using the Unet architecture model generating a very high accuracy over the images of urban environments. Paper also discusses the Unet architecture in detail explaining layer-by-layer construction. With the architecture hyper-parameters tuning is also important, and paper discusses the steps to correctly tune them. Also, the long training time limitation of these sophisticated models has been eradicated using cloud computing and Machine Learning Libraries.**

I. Background

Over the years, The problem of Image Segmentation has emerged in various fields ranging from finding types of cells in Neuro-Science to identifying different objects in geographical satellite applications. With the advancement in Deep Learning models and techniques, many fields have already used these powerful tools. However, We have still faced the problem of how we can train our machine to understand our urban surroundings. As we are surrounded by urban environments, there are unlimited applications facing the same problem of Image Segmentation of Urban Environment.

With the evolution of Convolutional Neural Networks, this task has become very efficient. Image Segmentation could typically be divided into two sub-categories, Semantic Segmentation and Instance Segmentation. Semantic Segmentation involves pixel level classification where we are grouping pixels which belong to the same object class, whereas Instance Segmentation involves a process of detecting each and every unique instance of all the classes present in an image. Figure 1 describes the difference between both the segmentations with Image 1 is semantic and Image 2 is instance segmentation.

In this paper, Semantic Segmentation is used and implemented using U-Net Architecture which is based on a residual deep learning model. The results produced are astonishing with very high accuracy in the multiple classes model.

Given the limitations of local hardware, following tools are used to increase the efficiency of our model:

Google Colab: A cloud computing service for Deep learning providing free GPU integration for your Python-based code, making the training and testing of your model very easy. It also has a functionality to mount your google drive and store all the results virtually. Most of the popular Machine Learning Libraries such as Pytorch, TensorFlow, Keras, etc can easily be integrated with this tool and take advantage of powerful Google hardware.

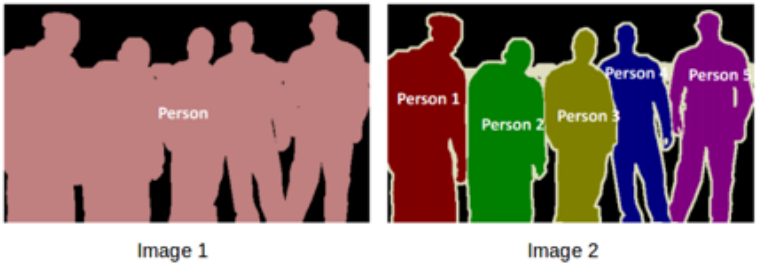

Fig. 1. Semantic VS Instance Segmentation

FastAI: FastAI is the software library which is built upon popular Python Deep Learning and Machine Learning libraries, and specifically catered towards the applications of Machine and Deep Learning. This library is different from any other Machine Learning library in a way that users have functionalities to easily build and transform the dataset, customize the existing models, and compare results. Together with Google Colab, this combination provides users with an immense number of resources to build the Image segmentation.

The dataset used in this paper is known as "CamSeq01 Dataset".This dataset consists of high resolution images from a video sequence build from a moving driving scene in the city of Cambridge. Within the dataset, we are provided with the hand-labeled ground truth images and the validation set images. This dataset has 101 960X720 pixels images in which each pixel is manually assigned to one of 32 object classes shown in the Figure 2.

Fig. 2. CamSeq01 Dataset classes

Section 2 of related work covers the similar techniques used for image segmentation in different applications.

Section 3 describes Model Architecture of U-Net layer-by-layer, also spanning the steps to tune various Hyper-parameters correctly.

Section 4 describes the Evaluation of our model with the Accuracy calculation and plotting of our results.

Section 5 consists of Future Work and potential extension of our model.

II. Related Work

In the Convolution Neural Networks, the input of images gets down-sized as they go from left to right which helps in image classification. Similarly, Regional Convolution Neural Network (R-CNN) is predecessor to U-Net model where it would classify different regions of the input image using the class specific linear SVMs. However, for the task of pixel-by-pixel image classification, output and input should be of same size. That's why U-Net comes into the play which uses the concept of Decoders, downsizing the samples and Encoders, upsizing the samples.

A good example of R-CNN is shown in the Figure 3. In Figure 3, different parts of the brain are identified using R-CNN.

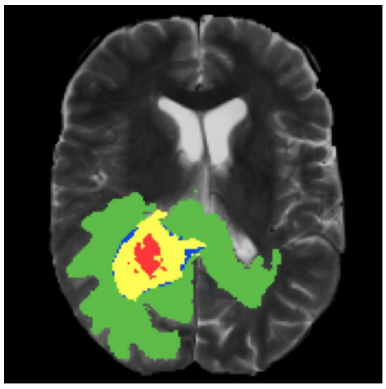

Fig. 3. R-CNN Example

AlexNet is Toronto's pioneering deep CNN which won the 2012 Image Net competition with a test accuracy of 84.6%. The Architecture consists of 5 convolutional layers, max-pooling ones, ReLUs as non-linearities, 3 fully-convolutional layers, and dropout. This was the first time when using multiple CNN layers, the accuracy results were pretty respectable.

Google LeNet: In the 2014 Image Net competition, Googles network won the competition with an accuracy of 92.3%. The Architecture consists of 22 layers, and a newly introduced building block called the inception module. This module has a Network-in-Network layer, a pooling operation, a large-sized convolution layer, and a small-sized convolution layer.
ResNet: This is architecture on which the U-Net is based upon. ResNet was created by Microsoft and won the 2016 Image Net competition with the 96.4% accuracy. It involves the concept of residual learning which is further discussed in the next section. Particularly, this ResNet had 152 deep layers, and results were very effective compared to the other approaches.

III. U-Net Model and Hyper-parameters Tuning

A. Architecture

The most important reason for using U-Net is to solve the problem of vanishing gradients. This problem becomes very apparent in networks having too many deep layers. In the deep layers, the gradients easily shrink to zero after forward-backward propagation. Due to this, weights got stuck at zero, not changing their value. However, in the case of U-Net, we are using Residual blocks where gradients skip connections to flow directly between different layers.

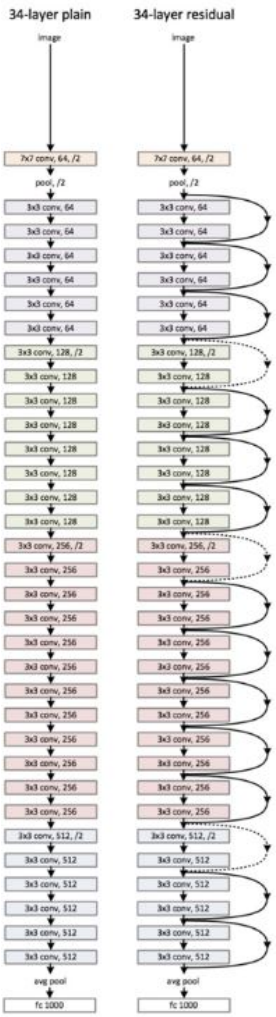

Fig. 4. 34 Layers Residual Blocks

Figure 4 shows the difference between the traditional 34 layers CNN and the 34 layers residual blocks. In the residual learning based model, weights from different layers are passed between the layers. This essentially ensures that we are not losing any high level information as network goes into deep layers.

Moreover, This transfer of weights are further refined to transition into Encoders-Decoders concept and form a U-Net Model.

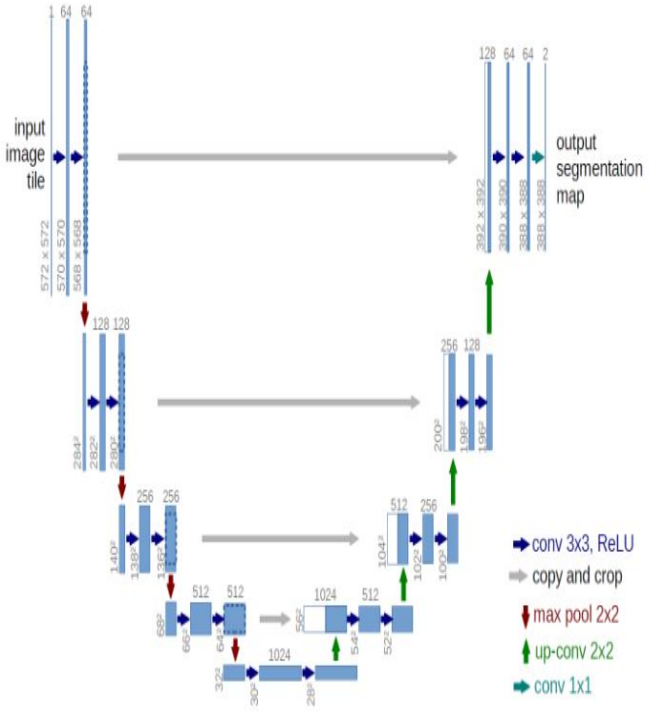

Fig. 5. U-Net Architecture

Figure 5 shows the Architecture of U-Net. As it it shown, there are two steps in this model. In the first step, inputs are passed into the downsizing layers called decoding whereas in the second step, upsampling is done which is called as encoding. Compared to the traditional CNN, the output and the input are of same size making the pixel-by-pixel image classification possible.

While Upscaling we are adding the outputs of the respective decoding layer to the encoding layer which is also called weight transfer. Also, Transposed Convolution layers are used after this step. In Transposed layers, the width and height of the images remains the same while depth is halved. This is done simply by breaking our tensor in two equal parts and combining the outputs in one-to-one form. After this transposed layers, outputs passed into convolutional layers with the padding = ”same” size, then loop continues in the decoding.

We can think of Encoding as Forward Propagation and Decoding as Backward Propagation in a sense that final outputs and inputs images are of same sizes.In the final layer, average pooling layer of 1 X 1 is used with the softmax activation function and cross-entropy loss function.

This paper utilized the architecture of ResNet34 consisting of 34 layers, the size and dimensions of these layers are given in the Figure 6.

Layers	34-Layer
conv1	7 X 7, 64, stride 2 3 X 3 max pool, stride 2
conv2_x	[3 X 3, 64] [3 X 3, 64] X 3
conv3_x	[3 X 3, 128] [3 X 3, 128] X 4
conv4_x	[3 X 3, 256] [3X 3. 256] X 6
conv5_x	[3 X3, 512] [3 X 3, 512] X 3
Average pooling Layer with Softmax Activation Layer and Cross Entropy Loss Function	

Fig. 6. 34 Layers Dimension

B. Hyper-Parameters Tuning

FastAI plays a very important role in configuring our model very efficiently. To begin with, very first concern is to how to effectively divide our data into training, testing, and validation sets. For this process, FastAI has provided a Data Block API, using this users can create a databunch for their model. Now for problem of Image Segmentation and with the given dataset,we can’t just randomize the data and divide into respective training, testing, and validation sets. Our dataset is video sequence and if we randomize our data for training, this would become too easy of a problem as we would not be testing it on unseen data. That’s why the dataset provider has given us ”valid.txt” file for mapping our data into the validation set easily.

Now we have our respective sets, we have to decide on learning rate used in our model. For this problem, we are using variable learning rate, and this process is called as ”one cycle policy”. In Figure 7, it is shown how the learning rate varies in the model with the momentum.

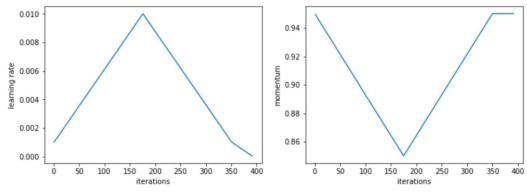

Fig. 7. Variable learning rate

The reasoning behind the variable learning rate is that our objective function could have many local maximas/minimas throughout our model and our values could just start oscillate between one local maxima/minima if we kept our learning rate constant. That's why in this policy, we start increasing our learning rate to reach the global maxima/minima faster, then decreases to settle down on a good solution. This tune has been very effective in decreasing the training time of the model significantly. Fortunately, FastAI has already built-in method to fit a model with this tune. Specifically for our model, learning rate varies according to Figure 8.

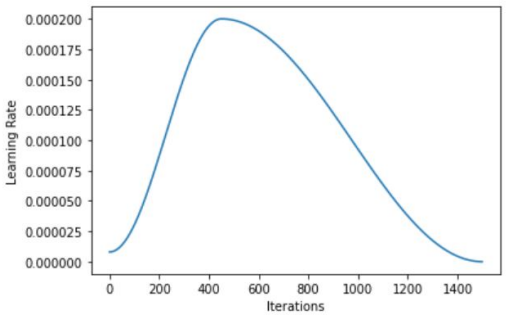

Fig. 8. Variable Learning Rate For Our Model

IV. Training Process and Evaluation

Considering our GPU limitations, we have started with sub-sizing our images by 2 to train our model. Using the one cycle fit policy, one concern left is how can we initialize the value of learning rate. TO solve this problem, we would let our model train initially for some time, then decide on learning rate for our training. In our model, initial loss vs learning rate is shown in Figure 9.

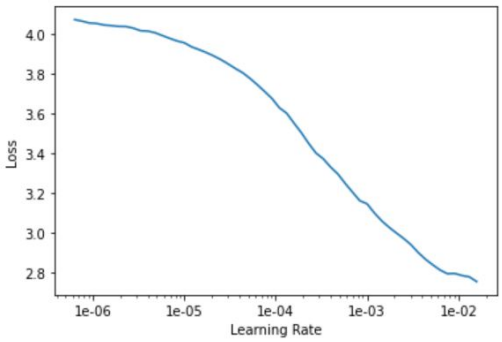

Fig. 9. Loss VS Learning Rate

On unfreezing our model, from the Figure 9, a good learning rate would 1e-3. FastAI method for one cycle cycle is: $learn.fitonecycle(10, slice(1e-3))$ This method is doing the training process 10 times with each time starting from the 1/10th of learning rate left in the previous iteration.For the first iteration, we have set the max learning rate to be 1e-3. After selecting this, we would let our model train again by unfreezing for some time and plot our loss VS learning rate again in Figure 10. This time a good estimate for maximum learing rate for one cycle fit policy is 1e-4.

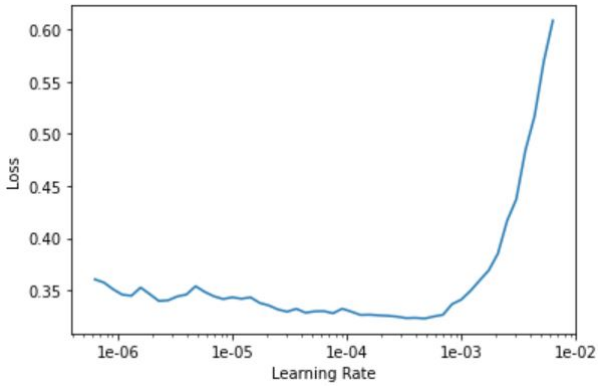

Fig. 10. Loss VS Learning Rate

After training our model using the process described above, we have plotted Plot VS Iterations with the validation set loss depicted by orange line in Figure 11.

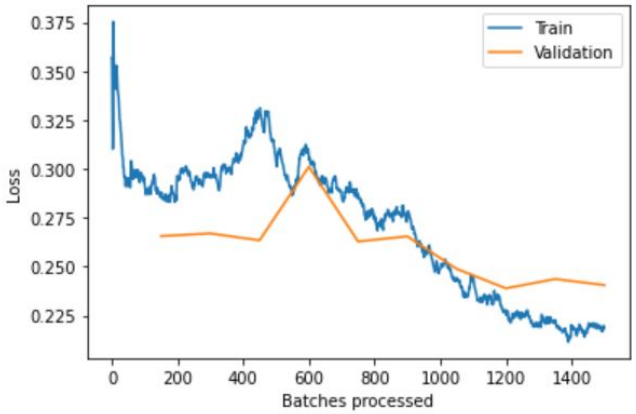

Fig. 11. Loss VS Iterations

From the Figure 11, we can see our model over-fits a bit so we tried tuning our hyper-parameters a bit more. However, the results were pretty much the same so one strategy is to increase the size of our sample by not halving them in the start, but due to this increase in size, our GPU couldn't handle this computation. To make this fit in our GPU limitations, we have decreased our batch size and got better results. However, while doing this whole process, Weights initialized to the left over weights value from our last model. This decreased our training time even further, now outputs are same size as our raw images. New Loss VS Iterations

plot is shown in the figure 12.

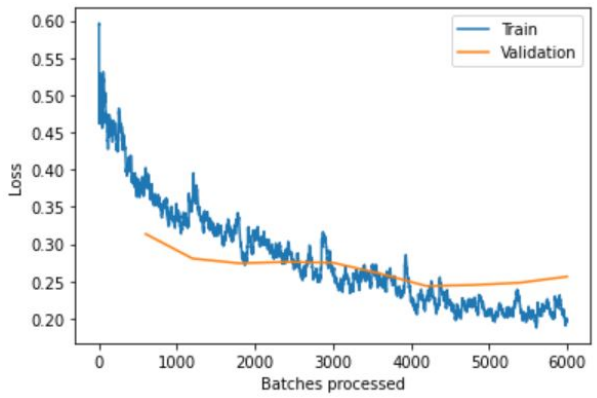

Fig. 12. Loss VS Iterations

For Accuracy Calculation, pixel-wise classification used with one-hot-encoding for our 32 object classes. Optimum values of this classes are given in the "codes.txt" inside the dataset. However, dataset provider has suggested to not count pixels classified as "void" class in our accuracy calculation. This is because "void" class is supposed to be used for the areas that don't belong to any other urban classes. When extending our model to take more classes, our void entries woud get reduced. Our Accuracy calculation would just be total number of pixels classified correctly divided by total number of pixels. The results are shown in the Figure 13.

epoch	train_loss	valid_loss	acc_camvid	time
0	0.402269	0.313491	0.910634	13:35
1	0.352384	0.280509	0.920474	13:36
2	0.320919	0.274105	0.919582	13:35
3	0.289653	0.275932	0.923566	13:33
4	0.259343	0.274893	0.922939	13:32
5	0.257682	0.260441	0.924452	13:33
6	0.228676	0.243556	0.930656	13:34
7	0.226691	0.245000	0.930444	13:33
8	0.216126	0.248078	0.930113	13:34
9	0.195613	0.256272	0.927409	13:33

Fig. 13. Evaluation

From Figure 13, we can see the Accuracy found is around 92% which is very good. Considering this model has been around for only 4 years, generating significantly better results from the traditional techniques is an achievement. The difference between the ground truth and prediction images is shown in Figure 14. From the Figure 14, they looks very similar given these ground truth images are hand-labeled and would already have some human-errors.

Fig. 14. Ground Truth VS Prediction

V. Future Work and Extensions

In Future, there is a lot of scope of hyper-parameters tuning using techniques like data augmentation, thresholding, etc so Overfitting can be reduced further by tuning these Hyper-parameters.

Instead of using only 34 layers, more deep networks could be build as our object classes and feature sizes increases. In order to support this extension, optimization of compiler and CPU FLOPS is needed.

To optimize our performance, images can shrunken down to do regional hybrid pixel wise classification.

On top of Image Segmentation, Edge detection can be added to make our machine further better understand the differences in our urban settings.

Moreover, the model should be tested in completely different environment such as in different country urban setting. This would help us in building more generalized solution as similar objects looks very different in various urban settings. One solution is train our model in batches of different urban settings, and test it out on completely unseen environment.

Given the unlimited hardware resources, one ambitious extension is to train our data using a real-time video of the urban setting, and produce Image Segmentation in the real-time. This can be applied to various commercial applications like Autonomous driving, Tracking persons in crowded areas, Google Street 3D segmentation, etc.

Producing datasets are still very laborous work and on top to it, hand-label requires significant time investment. However, these models could be used to generate synthetic images on which other models can be trained to automate this dataset collection problem.

VI. Conclusion

Overall, U-Net Architecture has produced great results, and with the help of tools such as Google Colab and FastAI, models are trained in a faster manner. However, a lot is still pending to be tested given dataset is video sequence of just one city. With the promising results generated by ResNet developed by Microsoft, more and more fields are getting attracted towards the commercial application of these models especially Bio-Medial and Health Science fields. Generating used dataset is not easy, it requires significant time and money investment so using these models further to generate synthetic images could solve this dataset collection problem. With the further advancements in Cloud computing and hardware, Sky is the limit for the application to these techniques

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *ECCV*, 2008.

[3] Harshall Lamba. Understanding semantic segmentation with unet, Feb 2019.

[4] Heet Sankesara. U-net, Jan 2019.

[5] Divam. A beginner's guide to deep learning based semantic segmentation using keras, Jun 2019.

[6] James Le. How to do semantic segmentation using deep learning, Jul 2018.

[7] H. Tao, W. Li, X. Qin, and D. Jia. Image semantic segmentation based on convolutional neural network and conditional random field. In *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*, pages 568–572, 2018.

[8] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018.

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.