Semantic Segmentation to Spade Pipeline

https://github.com/normankarr/CS194-Final-Project
https://inst.eecs.berkeley.edu/~cs194-26/fa21/upload/files/projFinalProposed/cs194-26-abe

Norman Karr

December 10, 2021

1 Introduction

The deep learning space of computer vision is rich with different problems, applications, and models. The primary goal of this final project was to be an exploratory endeavor into two of computer vision's deep learning problems: semantic segmentation and generative vision models. The end product is a semantic segmentation and SPADE pipeline such that both topics could be explored at an introductory level. The high level understanding is that the segmentation model takes an image as an input and outputs a segmentation map; this segmentation map is then fed into a conditional generative adversarial network, specifically SPADE.

2 Literature/Related Work

2.1 Semantic Segmentation Models

A common problem across different fields of machine learning are classification problems. Problems such as ImageNet challenged models to classify photographs in their entirety [1]. Semantic segmentation proposes a different sort of classification problem: pixel-wise classification. In short: given an image, can we classify each object by its exact pixels?

As of 2021, there exists many different semantic segmentation models such as U-net or Google's DeeplabV3 however I primarily wanted to train my own model instead of using a plain pre-trained model. In favor of simplicity, I opted to follow a fully-convolutional network as described in Long, et al's paper [2]. Described in this paper is to replace the final fully connected layers of existing models such as AlexNet and VGG with a 1x1 kernel convolutional layer with the output channel number being equal to the number of classes to classify. These outputs are then upsampled through a transpose convolution to bring each channel's image to the same size as the input image. The individual activation of each pixel in a channel represents the probability of that pixel being part of that channel's corresponding class.

2.2 Conditional GANs

The first conditional GAN I researched was Pix2Pix [3]. Pix2Pix demonstrated a method to use GANs for image-to-image translation; this model could take an input image of a segmentation label map or edge activations and then generate realistic images conditioned on the input. However, with inputs such as segmentation maps, it was observed that much of the conditioned information was being diluted or lost within the many convolutional and

normalization layer of pix2pix.

Since one of the pieces of my project was to pass a segmentation map into a GAN, I then researched a second model: the SPADE generator [4]. Spade improves upon Pix2Pix by introducing a new layer, spatially adaptive normalization, to attempt to prevent loss of information throughout the network. The high-level understanding of the layer is that it performs a channel-wise normalization with a with a learned scale and bias modulation.

2.3 Datasets

Through readings of papers and brief explorations through github repositories, I noticed some common datasets used for both semantic segmentation and GANs were Coco, ADE20K, and Cityscapes. Since I wanted to train my own semantic segmentation model, I opted to use Cityscapes due to its moderate size [5]. The split I chose to use was the finely annotated Cityscapes dataset which consists of 5000 finely annotated images of streets across Germany. Another benefit of cityscapes was that the SPADE repository provides a SPADE model pre-trained on Cityscapes' label ids.

3 Models and Methods

3.1 Fully Convolutional ResNet-34

For semantic segmentation, I decided to follow a transfer-learning approach with a pre-trained model rather than training a newly initialized fully convolutional network. I opted to use a ResNet-34 pre-trained on ImageNet in the hopes that many of the ImageNet features and kernels are transferable to Cityscapes segmentation. I had performed experiments with ResNet-50 and ResNet-18 but found that performance vs. training speed were best with ResNet-34. The details of the model is that I replace the average pool and fully connected layer of ResNet-34 with a 1x1 kernel convolutional layer followed by a transpose convolutional layer. This model was then trained end-to-end with an Adam optimizer for 25 epochs with a batch size of 32 and a learning rate of 1e-4.

3.2 SPADE:

For the conditional GAN portion of the project, I simply used an unaltered SPADE model pre-trained on cityscapes from NVIDIA's SPADE repository. The model takes a label id map and then generates a realistic image from that map. Given that the input to this pre-trained GAN is a label id map, I made sure that my segmentation model was trained to produce a label id map and not instance maps or polygons ids.

3.3 Data Augmentations

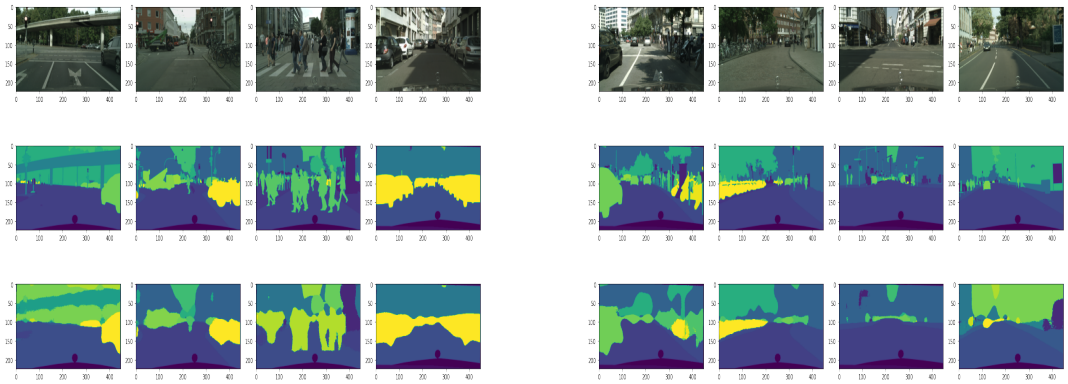
Generally, data augmentations are necessary because they help make networks more robust to transformations and noise and pseudo-increase the size of the dataset. In the case of training the segmentation network, they were necessary because the cityscape image sizes were 1024x2048 pixels while the expected image size for the pre-trained ResNet34 is 224x224 pixels. Although the change to a fully connected network actually removes the necessity for the images to be 224x224, the filters of the ResNet34 are all optimized for 224x224 images. Therefore, the first transformation applied was to resize the smaller dimension to 224 while maintaining the correct aspect ratio. I chose not to crop the image to 224 in the hopes that any sub-optimal filters could be processed through the fine-tuning process. Further transformations included a normalization in accordance to the normalization used for the

pre-trained model. Lastly, for training data only, I added a randomized color jitter to develop robustness to different colors as weather and lighting can vary heavily especially for street photography.

4 Results

4.1 Semantic Segmentation Model Results

I was able to train a successful segmentation network on Google Colab for 25 epochs in roughly 2.5 hours. The bulk of this runtime, roughly 70%, was unfortunately due to reading and loading the training data. The actual model had pretty quick forward and backward passes. Fortunately, while watching the training and validation data, both appeared to decrease with each epoch suggesting that my model was not over-fitting. Pictured in Figure 1 are some example results of my finalized model.



(a) Training Data Results

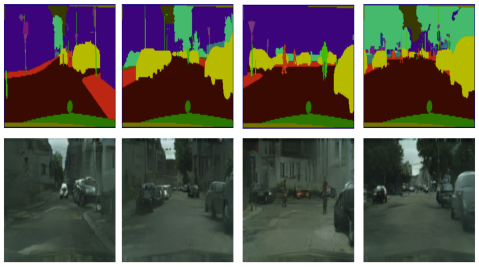
(b) Validation Data Results

Figure 1: The figures above display the comparisons between my model's predictions and the true labels across training and validation data. In both figures, the top row of images are example training images. The middle row is the training images' true segmentation. The third row is the output maps of my segmentation model.

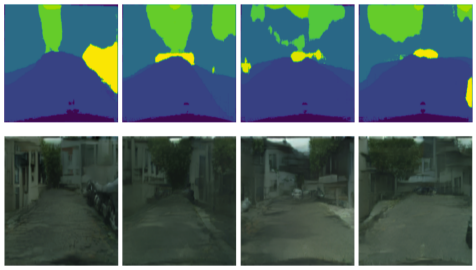
As is evident from figure 1, the model is relatively imprecise with it's pixel-wise-classifications but is able to capture the general structure of each image. There are a few examples where misclassifications appear to happen such as in the final column of the validation data example. However, this is likely because Cityscapes contains specific classes for car, caravan, truck, and trailer, all of which are fairly similar objects that my model appears to just classify as car. A similar misclassification also appears to happen with person and rider. Likely, these specific classifications would be improved with more training or a larger model.

4.2 SPADE Results

I first tested the pre-trained SPADE model on true labels to visualize how well the pre-trained model performs. I passed in the label id maps of every validation image as the input to the SPADE model and then visualized each generated image pair. Secondly, after I was confident that the model was working with the true labels, I then passed in my prediction maps as the input to the SPADE model. Displayed in Figure 2 are example results of both experiments.



(a) True Label maps to generated Image



(b) Prediction label maps to Generated Image

Figure 2: The figures above display the results of SPADE conditioned first on the true label maps and then on my prediction label maps. Note that the segmentation map images appear different because their displays use 2 different color maps

The results of the pre-trained SPADE model are not quite as good as I had hoped. The direct input of true labels produced photos that made sense but fell short of realistic. As a result, the output images of my models less precision predictions produced even less realistic images. Nevertheless, all the generated images show promise in terms of color, object differentiation, and general image structure. Likely, the SPADE model could have benefited from a certain degree of fine-tuning or a few additional epochs of training.

4.3 Additional Results

One interesting idea I had was what would happen if the generated image was passed back into the segmentation model. In theory, the generated image should not add significantly more structure in the images meaning that the generated image should produce the same segmentation map. As a result, I not only pipelined the segmentation map into SPADE, but pipelined the output of SPADE back into the segmentation model.

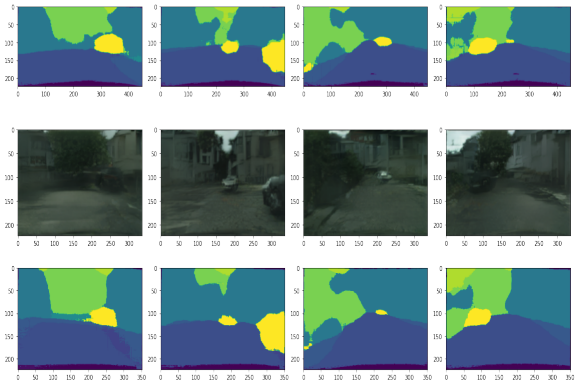


Figure 3: Four examples of feeding generated image into segmentation model. First row is original segmentation maps. Second row is the generated image from the map, third row is the new segmentation map

The results of this experiment are roughly what I expected. The new segmentation map is almost the exact same segmentation map as the original but with softer edges. This is actually a testament to the capabilities of the SPADE generator. It is able to generate objects with enough accuracy that not only is its discriminator fooled, but other models such as my segmentation model also see the correct object.

5 Conclusion

In regards to the project, I managed to get all my ideas and models functioning but I personally wish my results were a bit more interesting. For the semantic segmentation model, I would have liked to train for longer to get more accurate maps or worked with more complex models such as U-net. For the generator, I think had a idealized idea of what a pre-trained SPADE generator would do. If I had more time and could train my own GAN, I would likely spend some time researching how to add more specificity to the GAN such as generating an image in a specific city or at a specific time-of-day instead of just generating a general Cityscapes image.

Overall, I learned a lot throughout this project due to the open-ended style of the project. I gained experience in how to decide which data sets and models to use as well as how to extract valuable information from the many different existing git repositories. Especially with working with the SPADE generator, I had spent a lot of time reading through the code and readme files to understand how to actually use a pre-trained model. Additionally, I feel that I have learned better practices in my code when it comes to deep learning and working with a corpus of images. For example, I spent a lot of time just figuring out how to optimize image loading and saving in Colab.

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," 2015.

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," 2015.

[3] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," 2018.

[4] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu, "Semantic image synthesis with spatially-adaptive normalization," 2019.

[5] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," 2016.