Constructing a Fully Convolutional Network for Semantic Segmentation

Zirong Chen
Department of Computer Science
Georgetown University
Washington, D.C., U.S.A
zc157@georgetown.edu

Haotian Xue
Department of Computer Science
Georgetown University
Washington, D.C., U.S.A
hx82@georgetown.edu

Shuyang Yu
Department of Computer Science
Georgetown University
Washington, D.C., U.S.A
sy614@georgetown.edu

Yuansheng Xie
Department of Computer Science
Georgetown University
Washington, D.C., U.S.A
yx131@georgetown.edu

I. ABSTRACT

Fully convolutional networks have been shown to exceed state-of-the-art models in the task of semantic segmentation. Fully convolutional networks, trained end-to-end and pixel-to-pixel, have been shown to produce outputs with efficient inference and learning. [1][2] In this project, we define and introduce our fully convolutional network for semantic segmentation on the canonical PASCAL VOC Dataset and present the pixel accuracy as well as mean IU of our models as our results.

II. INTRODUCTION AND PROBLEM STATEMENT

Semantic segmentation is a pixel-by-pixel image classification problem. In this project, we construct a fully convolutional network with the architecture described in part V. We then train our fully convolutional network end to end on a subset of images from the PASCAL VOC 2012 Dataset and test our model on the remaining subset.

We evaluate our model using the metric of pixel accuracy, i.e., the percentage of pixels that our model was able to correctly classify in the testing image., as well as mean IU. We then compare the results of our own, trained-from-scratch model with pre-trained models such as DeepLabv3.

Semantic segmentation is of utmost usefulness in today's world. It remains a central aspect in the creation of autonomous vehicles. [3] Semantic segmentation is also used in facial recognition to estimate gender, age, ethnicity and even expressions. [4] In satellite imaging, semantic segmentation is used to segment different types of land and conduct automated landscaping. [6]

In this project, we intend to construct and replicate the results of FCN-8s from [1] and compare our segmentation results with state-of-the-art models like DeepLabv3.

III. RELATED WORK

CNN: When it comes to computer visualization, the first thing that comes to many people's minds is Convolutional Neural Networks, because they perform so well. [7] Under regular circumstances, CNN's, in fact, do perform well in the field of computer visualization. A traditional CNN, such as the one depicted below in Figure 1, only output the probabilities

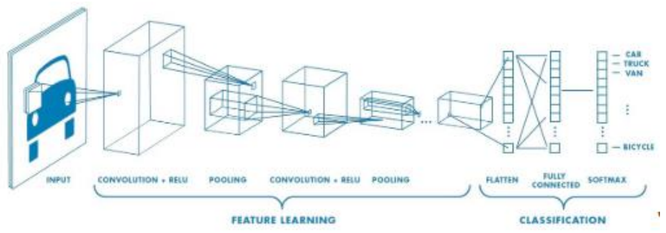


Figure 1: CNN for image classification

of *each image* belonging to each class. In order to conduct semantic segmentation in our project, we actually sought an estimation of the likelihood of *each pixel in each image* belonging to each class. Therefore, we will modify an existing CNN into a fully CNN, or FCN (Figure. 2), the architecture of which will be designed for semantic segmentation.

Encoder-decoder: An encoder-decoder first performs necessary dimension-reduction on an image. The decoder then

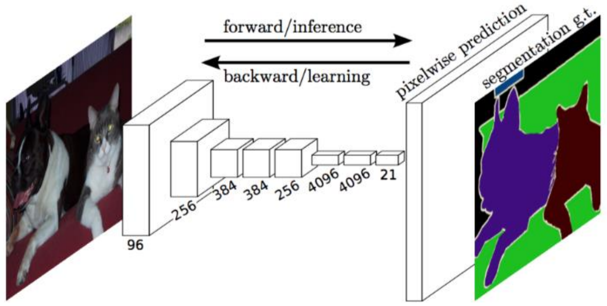


Figure 2: FCN for semantic segmentation [1]

up-samples the reduction that resulted from the encoder and reconstructs the image, which is compared with the original image in order to calculate error and loss during training. (See Figure 3.) [6]

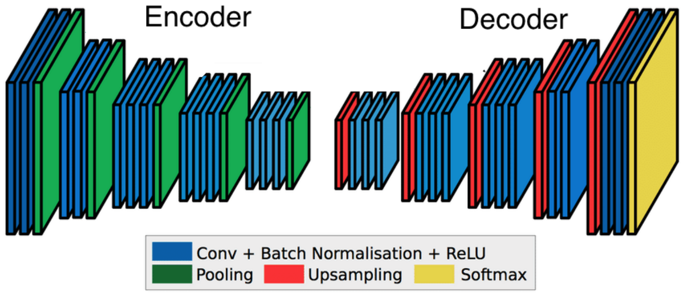


Figure 3: Encoder-decoder

In our project, instead of up sampling in our model, we utilized transposed convolutional layers to perform deconvolution to reconstruct the image back into its original dimensions. When the depth of a CNN gets too large, the model incurs a loss of information due to the vanishing gradient problem. Therefore, to combat this problem, we employed the approach of using skip connections mentioned in [1]. By connecting corresponding layers in the encoder-decoder and transferring information from earlier layers where information loss is relatively low to later layers where information loss is expected to be high, we can retain some of the information that might otherwise be lost in a CNN with large depth.

ResNet: Similar to the encoder-decoder above that utilizes skip connections, perceptrons in ResNet receive and retain information from perceptrons in previous layers. This characteristic is also useful in our model and for our problem because retaining information from previous layers is beneficial to the overall result. Our model is in a sense inspired by the intuition behind ResNet because we, too, intended to preserve information from layer to layer and incur the least amount of error possible in our model with relatively large depth. [5]

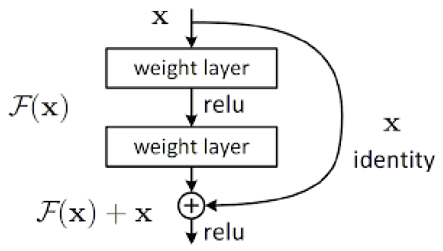

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 4: Synapses and perceptrons in ResNet

IV. DATASET

We utilized the dataset from PASCAL Visual Object Challenge 2012. A part of the challenge was for the challenge's participants to build models that perform semantic segmentation on the challenge's provided dataset. The dataset consisted of 2913 images with their corresponding ground truth – which are segmented images with each one of their pixels labeled to be an object. There are 20 classes for objects in these images and one extra class to represent the background. Each pixel was therefore labeled as one of these 21 classes in the ground truth. The classes are listed below:

1	Background
2	Aeroplane
3	Bicycle
4	Bird
5	Boat
6	Bottle
7	Bus
8	Car
9	Cat
10	Chair

11	Cow
12	Dining Table
13	Dog
14	Horse
15	Motorbike
16	Person
17	Potted Plant
18	Sheep
19	Sofa
20	Train
21	TV/Monitor

Table 1: Classes in PASCAL VOC 2012

V. Methods

Data preprocessing: The images in the dataset provided by the challenge were originally not compatible with our proposed model. We needed to preprocess them first before feeding them into our model. The images were first read in as multi-dimension arrays of size $h * w * c$, where h and w are the height and width of the image. c represents the number of channels, which, in this case, was 3, since all the pictures were of RGB format. The ground truth corresponding to each image was read in a similar manner. Due to the fact that the total amount of classes is numbered in this challenge, the labeled values of each pixel in the ground truth could only take on 21 possible RGB values corresponding to the classes. Our model was designed to output probabilities for each of the 21 classes. Therefore, our first major preprocessing task was to map the RGB values in the ground truth images to the 21 classes and output the resulted array which had dimensions $h * w * 21$. Each image also came in different sizes, which was also counterproductive to our efforts. During the preprocessing step we also had to reshape each image to a size fit for our models.

Model construction: Our general approach was to recreate and train from scratch our own FCN-8s similar to the one in [1]. We utilized several approaches mentioned in the paper as well as made some personal design choices.

- Channel Fusion: Between convolutional layers, in order to reduce the number of feature maps, we used a kernel of size (1,1) to "fuse" the many feature maps from previous layers and make them more compact.
- Up-sample by ConvTranspose2D: Usually, UpSampling2D does not achieve as good results due it repeating the same element in the rows and columns of the new matrix. UpSampling2D ignores the distribution in the original image. It also doesn't ensure that the dimensions would later be compatible with skip connections. ConvTranspose2D is a less naïve way of up-sampling that is more mathematically sophisticated and ensures the output's dimensions would match dimensions in the model's earlier layers.

Convolutional Transposition:

$$Input: (N, H_{in}, W_{in}, C_{in})$$
$$Output: (N, H_{out}, W_{out}, C_{out})$$

Where

$$H_{out} = (H_{in} - 1) \times stride[0] - 2 \times padding[0] + dilation[0] \times (kernel\ size - 1) + output\ padding[0] + 1$$

and

$$W_{out} = (W_{in} - 1) \times stride[1] - 2 \times padding[1] + dilation[1] \times (kernel\ size[1] - 1) + outputpadding[1] + 1$$

- Skip Connection: After many layers of convolution, the network is due to lose information. We utilized skip connections twice to connect earlier layers during convolution to their corresponding later layers during deconvolution. We utilized transposition and cropping in order to ensure that the dimensions in the network are compatible for skip connection.

Output & Visualization: Our predictions on the segmentation of each image are multi-dimensional arrays of size $h * w * 21$. For each pixel in the image, we took the class with the highest probability and labeled that pixel with this class. This label was then color mapped to a corresponding RGB color provided in PASCAL VOC 2012 and from there we were able to output our segmented image.
A complete depiction of our model is in Figure 5.

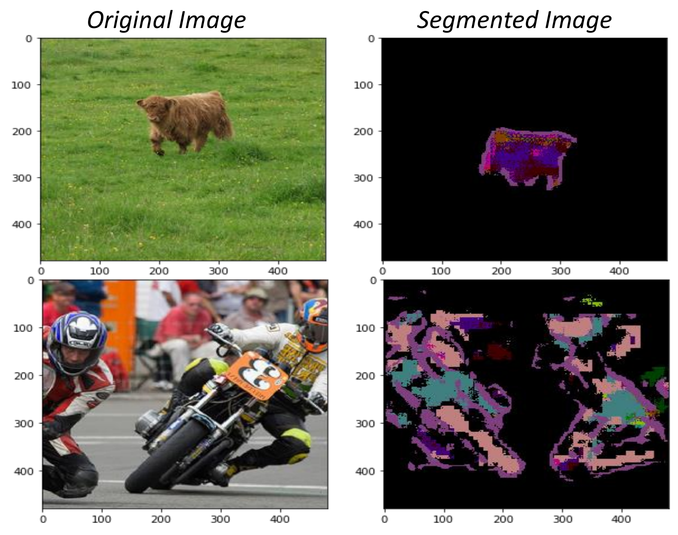
VI. RESULTS

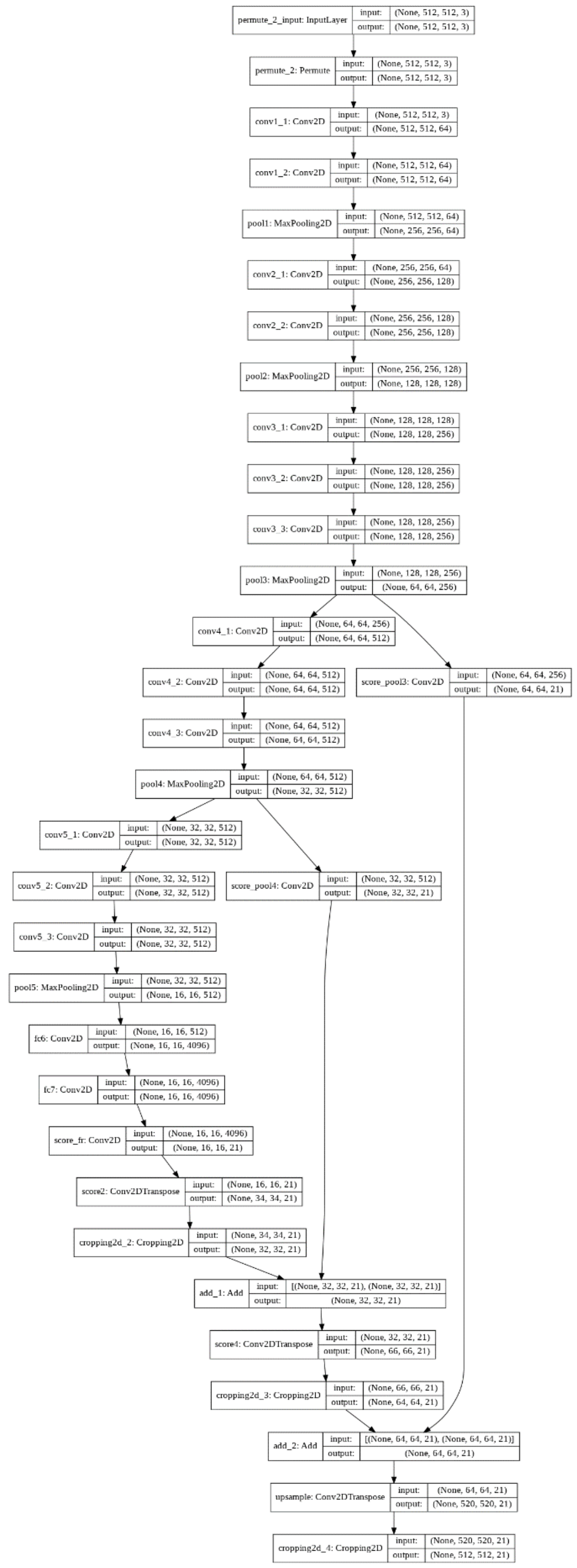
In the table below we summarize the performance of our model in comparison with some other models.

	Pixel accuracy	Mean IU
Our FCN	77.01	50.04
FCN-32 from [1]	89.1	59.4
FCN-16 from [1]	90.0	62.4
FCN-8 from [1]	90.3	62.7
DeepLabV3 [8]	*Not reported*	85.7

Table 2: Results from models
***N.B.*, a discussion of the results will be in VII.**

We also present here in this section several result segmentation images from our model:





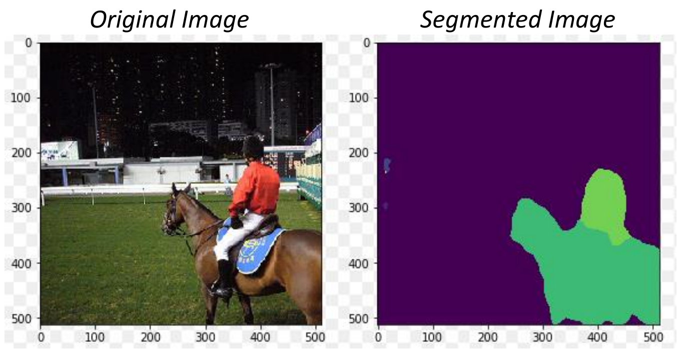
permute_2_input: InputLayer | input: (None, 512, 512, 3) | output: (None, 512, 512, 3)
permute_2: Permute | input: (None, 512, 512, 3) | output: (None, 512, 512, 3)
conv1_1: Conv2D | input: (None, 512, 512, 3) | output: (None, 512, 512, 64)
conv1_2: Conv2D | input: (None, 512, 512, 64) | output: (None, 512, 512, 64)
pool1: MaxPooling2D | input: (None, 512, 512, 64) | output: (None, 256, 256, 64)
conv2_1: Conv2D | input: (None, 256, 256, 64) | output: (None, 256, 256, 128)
conv2_2: Conv2D | input: (None, 256, 256, 128) | output: (None, 256, 256, 128)
pool2: MaxPooling2D | input: (None, 256, 256, 128) | output: (None, 128, 128, 128)
conv3_1: Conv2D | input: (None, 128, 128, 128) | output: (None, 128, 128, 256)
conv3_2: Conv2D | input: (None, 128, 128, 256) | output: (None, 128, 128, 256)
conv3_3: Conv2D | input: (None, 128, 128, 256) | output: (None, 128, 128, 256)
pool3: MaxPooling2D | input: (None, 128, 128, 256) | output: (None, 64, 64, 256)
conv4_1: Conv2D | input: (None, 64, 64, 256) | output: (None, 64, 64, 512)
conv4_2: Conv2D | input: (None, 64, 64, 512) | output: (None, 64, 64, 512)
score_pool3: Conv2D | input: (None, 64, 64, 256) | output: (None, 64, 64, 21)
conv4_3: Conv2D | input: (None, 64, 64, 512) | output: (None, 64, 64, 512)
pool4: MaxPooling2D | input: (None, 64, 64, 512) | output: (None, 32, 32, 512)
conv5_1: Conv2D | input: (None, 32, 32, 512) | output: (None, 32, 32, 512)
conv5_2: Conv2D | input: (None, 32, 32, 512) | output: (None, 32, 32, 512)
score_pool4: Conv2D | input: (None, 32, 32, 512) | output: (None, 32, 32, 21)
conv5_3: Conv2D | input: (None, 32, 32, 512) | output: (None, 32, 32, 512)
pool5: MaxPooling2D | input: (None, 32, 32, 512) | output: (None, 16, 16, 512)
fc6: Conv2D | input: (None, 16, 16, 512) | output: (None, 16, 16, 4096)
fc7: Conv2D | input: (None, 16, 16, 4096) | output: (None, 16, 16, 4096)
score_fr: Conv2D | input: (None, 16, 16, 4096) | output: (None, 16, 16, 21)
score2: Conv2DTranspose | input: (None, 16, 16, 21) | output: (None, 34, 34, 21)
cropping2d_2: Cropping2D | input: (None, 34, 34, 21) | output: (None, 32, 32, 21)
add_1: Add | input: [(None, 32, 32, 21), (None, 32, 32, 21)] | output: (None, 32, 32, 21)
score4: Conv2DTranspose | input: (None, 32, 32, 21) | output: (None, 66, 66, 21)
cropping2d_3: Cropping2D | input: (None, 66, 66, 21) | output: (None, 64, 64, 21)
add_2: Add | input: [(None, 64, 64, 21), (None, 64, 64, 21)] | output: (None, 64, 64, 21)
upsample: Conv2DTranspose | input: (None, 64, 64, 21) | output: (None, 520, 520, 21)
cropping2d_4: Cropping2D | input: (None, 520, 520, 21) | output: (None, 512, 512, 21)

Figure 5: Our complete FCN model

VII. DISCUSSION OF RESULTS

We recognize that our model's performance is lacking when compared to models and implementations from other papers. This is in part due to the fact that we trained our model from scratch and we didn't have the best hardware setup, nor did we have enough time to train our models to perfection. Therefore, given the circumstances, we feel that our results are acceptable and are satisfied with our work. We realize that there is ground for improvement for our model and our methods. If under different circumstances, we wish to improve our results by making the following amendments to our process.

- **Load Pre-trained weights:** Loading pre-trained weights into our model would most definitely boost the performance of the model. If we were looking to make the best predictions and shooting for the highest accuracies, then we would no doubt load pre-trained weights. However, in this project, it was a part of our goal, or ambition, dare we say, to train a network from scratch and compare our results with other models – so we neglected loading pre-trained weights when training our model. At the end, when we were analyzing our results, we did try loading pre-trained weights into our model and got some segmentation images; here is one for example. As we can see, the model indeed seemed to perform better after loading pre-trained weights.



- **CRF and Batch-Normalization:** If we were to learn and borrow from the experiences of the many versions of DeepLab, we could try and apply CRF and Batch-Normalization to our model as well. We conject that applying these methods would also boost our model's performance.
- **Time & hardware:** In this project specifically, it took us one afternoon to run 15 epochs on our model. We didn't have a lot of wiggle room when it came to fine-tuning the model because training took such a long time. If we had more time we would be able to perfect the model more. We also would look for better hardware to accelerate our training process.

VIII. CONCLUSION.

As a general summary of our process and experience in this project: We used our knowledge and intuition to construct a FCN-8 in an attempt to perform semantic segmentation on the dataset of PASCAL VOC 2012 and replicate results in well-known papers. Although we concede that the results have much room for improvement, we are satisfied with our results given our constraints. We believe the architectural choices we made and mentioned in sections V – VII were helpful to our endeavor. We also have certain revisions we would like to make to our model that we believe would enhance our model even further; they are mentioned in VII.

All members of our team have a passion in deep learning and computer vision. The topic of semantic segmentation piqued each one of our interests as it could synthesize the content covered by this class with a topic that is significant and meaningful to each one of us. We understand that the field of computer vision is very competitive and very well developed. In recent years, there have come into existence models that can perform even better than actual human beings. Nonetheless, we felt that there would always be room for improvement and we chose semantic segmentation to be the project with which we would hone in our fundamental skills and prepare us for more future research and work in deep learning and computer vision. We understood from the get-go that training a deep and complicated model from scratch was going to be challenging and we may never achieve the exact results reported in well-known on our own given the constraints; but these things did not deter us from completing our project and achieve satisfactory results.

REFERENCES

[1] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3431-3440).

[2] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation.

[3] Treml, M., Arjona-Medina, J., Unterthiner, T., Durgesh, R., Friedmann, F., Schuberth, P., ... & Nessler, B. (2016, December). Speeding up semantic segmentation for autonomous driving. In *MLITS, NIPS Workshop* (Vol. 2, p. 7)

[4] Mody, P. (2018, August 7). Semantic Segmentation: Wiki, Applications and Resources. Retrieved from https://blog.playment.io/semantic-segmentation.

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[6] Gupta, D. (2019, June 6). A Beginner's guide to Deep Learning based Semantic Segmentation using Keras. Retrieved from https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html

[7] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

[8] Chen, L. C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 801-818).