Semantic Segmentation Using Deconvolution Network

Avyav Kumar (2013A7PS084G)
Anurag Rai (2013A7PS693G)
Rohan Shah (2013A7PS068G)

***Abstract*—We were given variable sized images of blood cells and were tasked with identifying the RBCs from it using computer vision and machine learning. The segmentation was to be done semantically. We had assigned 1 to each pixel that made up the RBC while assigned 0 to every other pixel otherwise called background. We used a architecture comprising of two parts-convolution and deconvolution networks. The convolution network takes in a RBG image of a particular size and extracts its features to generate a multi-dimension feature representation while the deconvolution network produces a object segmentation in grayscale of the actual image from the features extracted from the convolution network.**

I. Introduction

Semantic Segmentation of images is the process of partitioning an image into multiple segments or (set of pixels). The goal is to simplify and/or change the representation of the image into something that is more meaningful or easier to analyze. Semantic Segmentation in images (also known as image segmentation) is typically used to locate specific objects and boundaries in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

II. Background Reading

A. Convolutional Neural Networks

Convolutional neural networks (CNNs) consist of multiple layers of receptive fields. These are small neuron collections which process portions of the input image. Convolutional networks may include global or local pooling layers which combine outputs of neuron clusters. A convolution operation on small regions of input is introduced to reduce the number of free parameters and improve generalization. One major advantage of these networks is the use of shared weight in convolutional layers which means that the same filter (weight bank) is used for each pixel in the layer; this both reduces memory footprints and improves performance.

B. Deconvolution Neural Network

A multi-layer deconvolution network is composed of deconvolution, unspooling and rectified linear unit (ReLU) layers.

1) Unpooling

Pooling in convolution network is designed to filter noisy activations in a lower layer by abstracting activations in a receptive field with a single representative value. Although it helps classification by retaining only robust activations in upper layers, spatial information within a receptive field is lost during pooling, which may be critical for precise localization that is required for semantic segmentation. To resolve this issue, unspooling is done in a deconvolution network. Unpooling performs the reverse operation of pooling and reconstructs the original size of activations. Unpooling records the location of maximum activations selected during pooling operation and uses it to place each activation back in its original pooled location.

2) Deconvolution

The output of an unpooling layer is an enlarged, yet sparse activation map. The deconvolution layers densify the sparse activations obtained by unpooling through convolution-like operations with multiple learned filters. However, contrary to convolutional layers, which connect multiple input activations within a filter window to a single activation, deconvolutional layers associate a single input activation with multiple outputs. The output of the deconvolutional layer is an enlarged and dense activation map.

III. Methodology

We start off with a 128 x 128 x 3 image. Here the 128 x 128 is the pixel size of the image and 3 denotes the colour scale of the image as RBG. Initially we start off convolution with 3 x 3 filters. There will be 5 convolutional layers each with a different number of filters. Also there are 4 pooling layers to help abstract activations. The first convolutional layer takes in 32 filters and makes a map of 32 x 128 x 128. Then pooling occurs which cuts it down to 32 x 64 x 64. The next round contains 64 filters each of 3 x 3. Again, pooling follows it. Going on like this with 96, 128 and 160 filters respectively in the subsequent layers we finally get a map of 160 x 16 x 16.

Now starts the process of deconvolution. Deconvolution has 5 rounds of unpooling and 4 rounds of deconvolution. We start of first with unpooling the last convolutional layer. This is followed by a deconvolution layer with 128 filters. This continues in further layers with 96, 64 and 32 filters respectively. Finally we get an image of the same size as the input image but in grayscale (i.e. 128 x 128 x 1).

Since the feature map decreases with depth, layers near the input image will have fewer filters while layers higher up can have more. Thus we start with 32 filters in convolution and end with 160 filters in its last layer. A 2 x 2 pooling is used here. We have also normalized the image using mean and standard deviation.

IV. RESULTS

- 20 epochs over the test data
- The loss function used was binary_crossentropy, which is included in the KERAS framework. The entropy was further minimised.
- The dataset included 528 images and their associated 528 masks. The original test data was broken and re-stiched to increase the size of the dataset so that fitting becomes more optimised.
- Final loss reported – 0.03452
- Final accuracy gauged – 0.8039

REFERENCES

The following were referred to while understanding and implementing the project.

[1] Hyeonwoo Noh, Seunghoon Hong and Bohyung Han, "Learning Deconvolution Network for Semantic Segmentation", Department of Computer Science and Engineering, POSTECH, Korea - https://arxiv.org/abs/1505.04366

[2] https://github.com/jocicmarko/ultrasound-nerve-segmentation

[3] http://cs231n.github.io/convolutional-networks/

[4] https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

[5] https://keras.io/layers/convolutional/

[6] https://keras.io/backend/

[7] https://keras.io/getting-started/functional-api-guide/