DarrenNet: Fully Convolutional Network for Semantic Segmentation

Brandon Szeto [*]
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92122
`bszeto@ucsd.edu`

Darren Yu [†]
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92122
`damiyu@ucsd.edu`

Nathaniel Thomas [‡]
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92122
`nathomas@ucsd.edu`

Abstract

This abstract provides a succinct overview of our work focused on semantic segmentation using the PASCAL VOC-2007 dataset. Our approach involves leveraging a novel deep learning architecture tailored for semantic segmentation tasks. We meticulously preprocess the dataset, addressing challenges such as class imbalance and data augmentation to enhance model generalization. Through experimentation, we explore hyperparameter tuning strategies to optimize performance. Our results showcase a notable improvement in both percent correct and Mean Intersection over Union (IoU) metrics, underscoring the efficacy of our methodology. Additionally, we uncover intriguing insights into the model's behavior, shedding light on its strengths and potential areas for refinement. This work showcases a framework for accurate and nuanced semantic segmentation.

[*] 160 lbs
[†] 130 lbs
[‡] 200 lbs

Preprint. Under review.

Introduction

Semantic segmentation is the task of assigning each pixel in an image a specified label. A single image can be partitioned into multiple meaningful segments corresponding to a given object or region of interest. Labels are assigned to individual pixels, marking boundaries and shapes. Altogether, this task has many applications in computer vision including but not limited to autonomous driving, video surveillance, and object recognition.

In recent years, convolutional neural networks (CNNs) have proven to be an effective approach to semantic segmentation. Such models learn hierarchical features by capturing information using convolutional blocks. Convolutions and related techniques such as weight initialization, batch normalization, and pooling help avoid the flat, dense layers of a traditional fully connected neural network allowing for less computation and improved feature maps.

Xavier Weight Initialization

In our architecture, we use Xavier weight initialization. In Uniform Xavier Initialization, a layer's weights are chosen from a random uniform distribution bounded between

$$\pm\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \tag{1}$$

where n_i is the number of incoming network connections and n_{i+1} is the number of outgoing network connections.

In Normal Xavier Initialization, a layer's weights are chosen from a normal distribution with

$$\sigma = \frac{\sqrt{2}}{\sqrt{n_i + n_{i+1}}} \tag{2}$$

where n_i is the number of incoming network connections and n_{i+1} is the number of outgoing network connections.

The Xavier initialization was created in response to the problem of vanishing and exploding gradients in deep neural networks in the context of symmetric nonlinearities (sigmoid, tanh). The intuition is that the weights should not be intialized randomly, but rather proportional to the size of two connected layers. As a result, the variance of activations and gradients would be maintained through the layers of a deep network.

Kaiming Weight Initialization

With the rise of ReLU, the nonlinearity can no longer be assumed to be symmetric. As such, the assumptions made by the Xavier Weight Initialization fall apart. In 2015, He et. al demonstrated that a ReLU layer typically has a standard deviation close to

$$\sqrt{\frac{n_i}{2}} \tag{3}$$

where n_i is the number of incoming network connections. As such, weights initially chosen from a normal distribution should be weighted by $\sqrt{\frac{n_i}{2}}$, with bias tensors initalized to zero.

Batch Normalization

During the training of a neural network, at each layer, inputs (activations) change as the parameters of the previous layers get updated. Batch normalization normalizes the activations of each layer by subtracting the mean and dividing by the standard deviation of the activations within a mini-batch. We implement batch normalization using PyTorch's `nn` module implementation. For example, we define a layer used for batch normalization as

```
self.bnd1 = nn.BatchNorm2d(32)
```

This ensures that the inputs to each layer have a consistent distribution, which helps in stabilizing the training process.

«««< HEAD

Related Works

U-Net

U-Net Ronneberger et al. [2015] has proven to be highly effective for biomedical image segmentation tasks. Its architecture features a contracting path to capture context and an expansive path for precise localization. The innovative use of skip connections facilitates the flow of fine-grained details across different layers, enhancing the model's ability to accurately delineate object boundaries.

ResNet

ResNet He et al. [2015] addresses the challenges of training very deep neural networks by employing residual connections. These connections enable the direct flow of information across layers, mitigating the vanishing gradient problem and facilitating the training of extremely deep networks.

Relevance

We test the performance of both models on the image segmentation task using the PASCAL VOC-2007 dataset. We look for differences in the models and their relative performance in comparison to the basic fully connected network that we started with.

||||||| empty tree =======

Related Works

U-Net

U-Net **?** has proven to be highly effective for biomedical image segmentation tasks. Its architecture features a contracting path to capture context and an expansive path for precise localization. The innovative use of skip connections facilitates the flow of fine-grained details across different layers, enhancing the model's ability to accurately delineate object boundaries.

ERFNet

ERFNet **?** introduces a novel factorized convolutional layer that significantly reduces the number of parameters while maintaining expressive power. This reduction in parameters enables faster inference without compromising performance, making it well-suited for deployment in resource-constrained environments.

ResNet

ResNet **?** addresses the challenges of training very deep neural networks by employing residual connections. These connections enable the direct flow of information across layers, mitigating the vanishing gradient problem and facilitating the training of extremely deep networks.

Relevance to our model

We test the performance of the above models on the image segmentation task using the PASCAL VOC-2007 dataset. We look for differences in the models and their relative performance in comparison to the basic fully connected network that we started with.

»»»> final$_{w}riteup$

«««< HEAD

Methods

Baseline

Improvements Over Baseline

Experimentation

In the Methods section, you should describe the implementation and architectural details of your system - in particular, this addresses how you approached the problem and the design of your solution. For those who believe in reproducible science, this should be fine-grained enough such that somebody could implement your model/algorithm and reproduce the results you claim to achieve. (a) Baseline (5 pt): You should describe the baseline architecture, stating the appropriate activation function on the last layer of the network, the loss criterion, weights initialization scheme and the gradient descent optimizer you used. (b) Improvements over Baseline (5 pt): You should describe the approaches you took to improve over the baseline model. (c) Experimentation (10 pt): Describe your two experimental CNN architectures(parts 5.a and 5.b) and the U-Net, each in a table, which the first column indicate the particular layer of your network, and the other columns state the layer's dimensions (e.g. in-channels, out-channels, kernel-size, padding/stride) and activation function/non-linearity. Describe any regularization techniques (e.g. data augmentation) you used, parameter initializa- tion methods, gradient descent optimization, and how you addressed the class-imbalance problem . ||||||| empty tree =======

Methods

Baseline

(a) Baseline (5 pt): You should describe the baseline architecture, stating the appropriate activation function on the last layer of the network, the loss criterion, weights initialization scheme and the gradient descent optimizer you used.

Our baseline architecture consists of an encoder, decoder, classifier, and activation.

Encoder
We have five convolutional layers that increases the depth of the orginal 3 channels to 32 to 64 to 128 to 256 to 512 each using a size 3 kernel, padding of 1, a stride of 2, and no dilation. Each layer sees a small decrease in the height and width of the layer according to the expression $\frac{W-F+2P}{2}+1$. The outputs of each convolutional layer are subsequently passed through a ReLU activation function and a batch normalization layer.

Decoder
We have five deconvolutional, or upsampling layers that decreases the depth of the final 512 deep layer output from the encoder. This is decreases from 512 to 256 to 128 to 64 to 32 each using a size 3 kernel, padding of 1, a stride of 2, and no dilation. Each layer sees a small decrease in the height and width of the layer according to the expression $S(W-1)+F-2P$. Similarly, the outputs of each deconvolutional layer are subsequently passed through a ReLU activation function and a batch normalization layer.

Classifier and Activation
In our final layer, we have a 1×1 convolutional kernel

Improvements Over Baseline

(b) Improvements over Baseline (5 pt): You should describe the approaches you took to improve over the baseline model.

Experimentation

(c) Experimentation (10 pt): Describe your two experimental CNN architectures(parts 5.a and 5.b) and the U-Net, each in a table, which the first column indicate the particular layer of your network, and the other columns state the layer's dimensions (e.g. in-channels, out-channels, kernel-size, padding/stride) and activation function/non-linearity. Describe any regularization techniques (e.g. data augmentation) you used, parameter initializa- tion methods, gradient descent optimization, and how you addressed the class-imbalance problem .

»»»> final$_{w}riteup$

Results

In the Results section, you should demonstrate your baseline model's performance, perfor- mances for each of the parts of Q4.b and Q4.c. You should include both of the performance metrics described in Evaluation metrics (see part 1 above in Implementation Instructions). Please organize these results into a series of concise tables. The formatting is your choice, as long as it is easily interpretable. For each architecture include: (a) A single plot showing both training and validation loss curves; (b) Validation set pixel accuracy and average IoU. (c) Visualizations of the segmented output for any one image in the test set along with the original image. Use the color coding mapping in the voc.py for this (Please refer to visualize.ipynb for help on plotting).

Discussion

The Discussion section should again have 3 sub-sections for Q3, Q4 and Q5. In each section you should discuss the benefits (and drawbacks if any) of the approaches/architectures you used and some discussion of why you think you got the results you got. Please discuss the following important points as well: How did the performance of your implementations differ? Discuss the performance differences with respect to the baseline (part 3), improved baseline (part 4) and compare the other implementations (part 5a, 5b, 5c) and mention them in their respective sub-sections. Also, draw insights from the plotted loss curves, tables and the visualizations.

«««< HEAD

Contributions

Brandon Szeto:

Darren Yu: Abstract, team name

Nathaniel Thomas: ||||||| empty tree =======

Contributions

Brandon Szeto: Basic FCN, model saving and loading, cosine annealing, average IoU and pixel accuracy, architecture research and writeup

Darren Yu: CUDA support, Plot training versus validation loss

Nathaniel Thomas: Custom CLI, Apple silicon support, dataset augmentation, class imabalance problem, model inference image preview, transfer learning using ERFNet, ResNet, and UNet. »»»> final$_{w}riteup$

References

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015b.

Eduardo Romera, José M. Álvarez, Luis M. Bergasa, and Roberto Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015a.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015b.