Paper Implementation Report

Advanced trial of FCN and limitations

[1] Fully Convolutional Networks for Semantic Segmentation

[2] Learning Deconvolution Network for Semantic Segmentation

Ⅰ. Introduction

Background: The reason why I tried to implement these papers is that the need of semantic segmentation technologies will play a key role of the future industries related visual tasks such as autonomous vehicles, robotics and machine vision, etc., as far as I believe. Thanks to the development of deep learning, there has been great progress in a variety of visual task areas such as object recognition, object detection, and object segmentation. I suppose that these developments are very significant for above fields and especially the segmentation skill will be much important than ever in the future technologies of 4th Industrial revolution. Because in order to apply these techniques in real life, it is necessary not only to be accurate but also to be highly accurate and reliable. So I wanted to implement "*fully convolutional network (FCN)*", a paper that contributed a big step to semantic segmentation. In addition to merely implementing this, I tried to go a few more steps in the direction suggested by the author and to look at the limitations of this method. Also, I implemented *"Deconvolution network"* which pointed out and solved the problem of FCN. So I'll show what the results is and how it have changed.

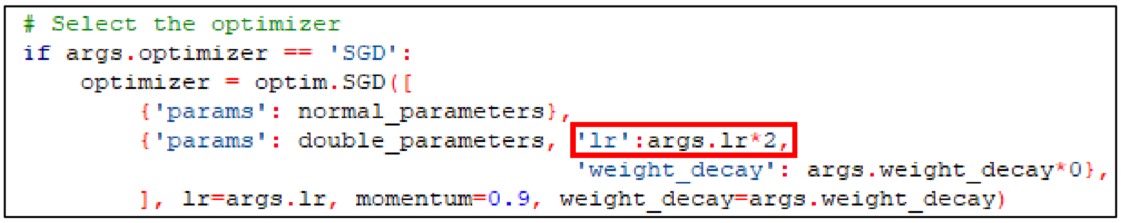
Key contributions: One of the key contribution of the FCN paper is that they extend the existing classification nets to semantic segmentation nets by replacing fully-connected layers with fully-convolutional layer. Fully-convolutional layer enable the network to contain the spatial information using convolutional layer with a 1x1 kernel size. Another one is that they improve not only segmentation performance but also the speed of learning and inference through multi-resolution layer combinations (*skip connections*). To demonstrate these major contributions, I conducted several experiments. I'll show their results in following sessions.

Ⅱ. Implementation details

I tried to royally follow the implementation details in this paper as far as possible. The author's original code was implemented in the *'caffe'*, a past library of python, <u>but the most attempts to implement this FCN paper were often overlooked by the detailed settings that the author mentioned.</u> So I applied the following two things faithfully, which most people did not take care of, and I followed most of the other detailed settings.

A. Doubled learning rate for bias terms

Most people who were not familiar with the library well implemented the model, but applied the same leveling rate to all layers without applying the "double learning rate for biases" that the author referred to in Section 4.3. As shown in Figure1, I divided bias and weight into different groups and applied twice the learning rate to only biases.

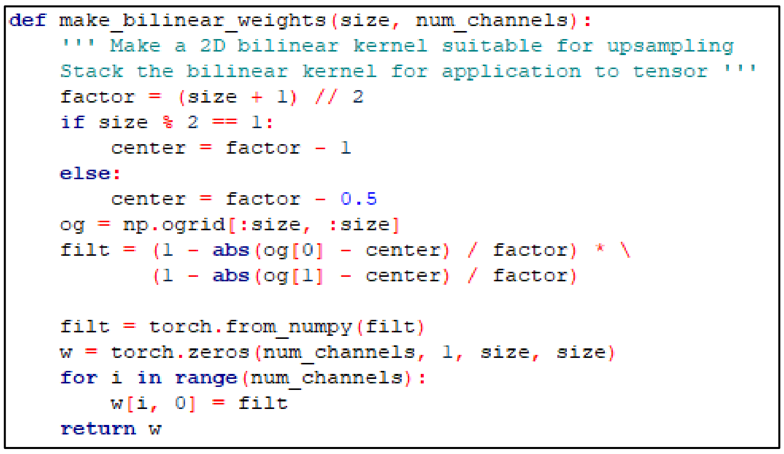


```python
# Select the optimizer
if args.optimizer == 'SGD':
    optimizer = optim.SGD([
        {'params': normal_parameters},
        {'params': double_parameters, 'lr':args.lr*2,
                                      'weight_decay': args.weight_decay*0},
        ], lr=args.lr, momentum=0.9, weight_decay=args.weight_decay)
```

Figure 1. The parts of my pytorch code for doubled learning rate for biases.

B. Bilinear interpolation weight initialization

One of the other settings that many people overlooked was that they did not initialize the weights of the deconvolution layer to operate as a bilinear interpolation. I initialize each deconvolution layer's weight to operate as bilinear interpolation for the first time, referring to the function that generates the bilinear interpolation filter in Figure2. And I made them learnable, except the final deconvolution layer.



```python
def make_bilinear_weights(size, num_channels):
    ''' Make a 2D bilinear kernel suitable for upsampling
    Stack the bilinear kernel for application to tensor '''
    factor = (size + 1) // 2
    if size % 2 == 1:
        center = factor - 1
    else:
        center = factor - 0.5
    og = np.ogrid[:size, :size]
    filt = (1 - abs(og[0] - center) / factor) * \
           (1 - abs(og[1] - center) / factor)

    filt = torch.from_numpy(filt)
    w = torch.zeros(num_channels, 1, size, size)
    for i in range(num_channels):
        w[i, 0] = filt
    return w
```

Figure 2. Function of making bilinear weights.

C. Other detailed settings

As noted in the paper, the batch size: 20, momentum: 0.9, weight decay: 0.0005, and SGD was used as an optimizer. In order to compare performance according to optimizer, Adam optimizer was also used to compare results. I applied zero initialization to all the score layers, and used dropout after the fully convolutional layers. The slightly different point in my case from the author is that all models were trained up to 200 epoch and the image was resized to 256 sizes.

Ⅲ. Experiments and results

To show the key contributions I mentioned above, the following experiments were carried out as below:

A. Adapting classifiers for dense segmentation prediction

First, I changed the classifier part of AlexNet, VGG and GoogLeNet to fully convolutional layers with a 1x1 kernel size, and fine-tuned all layers by backpropagation through the whole net. For AlexNet, the layers were somewhat different from the ones the author used, because I used a pre-trained model in

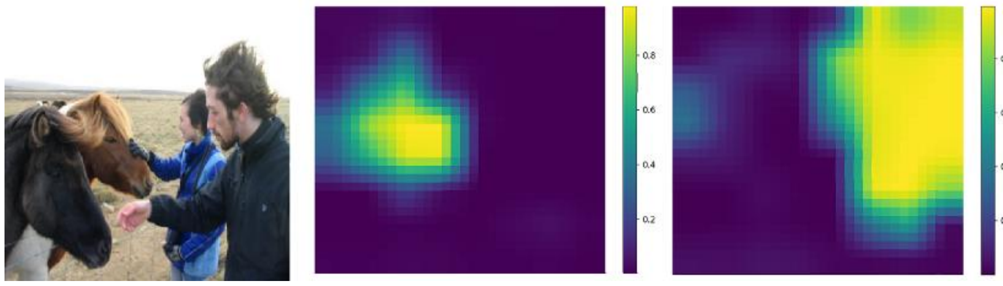


Figure 3. Visualized score map in FCN-VGG16.
(left: origin, middle: horse score, right: person score)

torchvision library. Figure3 is a visualization of score results from a validation image in PASCAL VOC 2011 dataset. If you look at the results of each channel which corresponds to horse and person, you can see that you are well predicting the part that corresponds to the actual image. It shows that the learned FCN well contains the spatial information.

	FCN-AlexNet	FCN-VGG16	FCN-GoogLeNet
Mean IoU (%)	26.4	**38.9**	33.6
Forward time	0.8 ms	16.9 ms	8.0 ms

Table 1. mean IoU and forward time results of the FCN-Classifiers.

Table 1 is the result of reproducing the results of Table 1 on paper. Overall mean IoU results was smaller than the values in the paper, but the tendency was similar. The forward time was much faster than the author's because the GPU I used was Titan V, so I think it was simply because the calculation speed was faster. This would have been meaningful if we checked it with the same GPU. However, the trend was similar.

B. The effects of the combination of course information

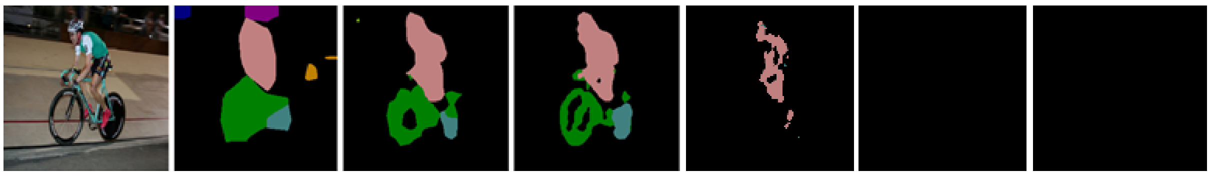


Figure 4. Each FCN's segmentation results of a validation image.
(origin, FCN32s, FCN16s, FCN8s, FCN4s, FCN2s, FCN1s)

Next, I checked the segmentation result of each FCN model with the added skip connection. FCN32s were initialized with the pre-trained parameters of VGG16 to learn with a learning rate of 10e-4. The other FCNs were immediately initialized with top-level learned course FCNs, and they were selected as the best performers by learning 10e-3, 10e-4, 10-e-5 and 10-e-6, respectively. Table2 shows that FCN8s had high values for ***pixel acc.*** and ***f.w. IoU***, but ***mean acc.*** and ***mean IoU*** had the highest values for FCN32s and FCN16s, respectively. However, Figure2 shows that from FCN32s to FCN8s, namely, **the more course layers are combined, the more detailed the segmentation result is until FCN8s.** Although the original paper showed only the results up to FCN8s, I also

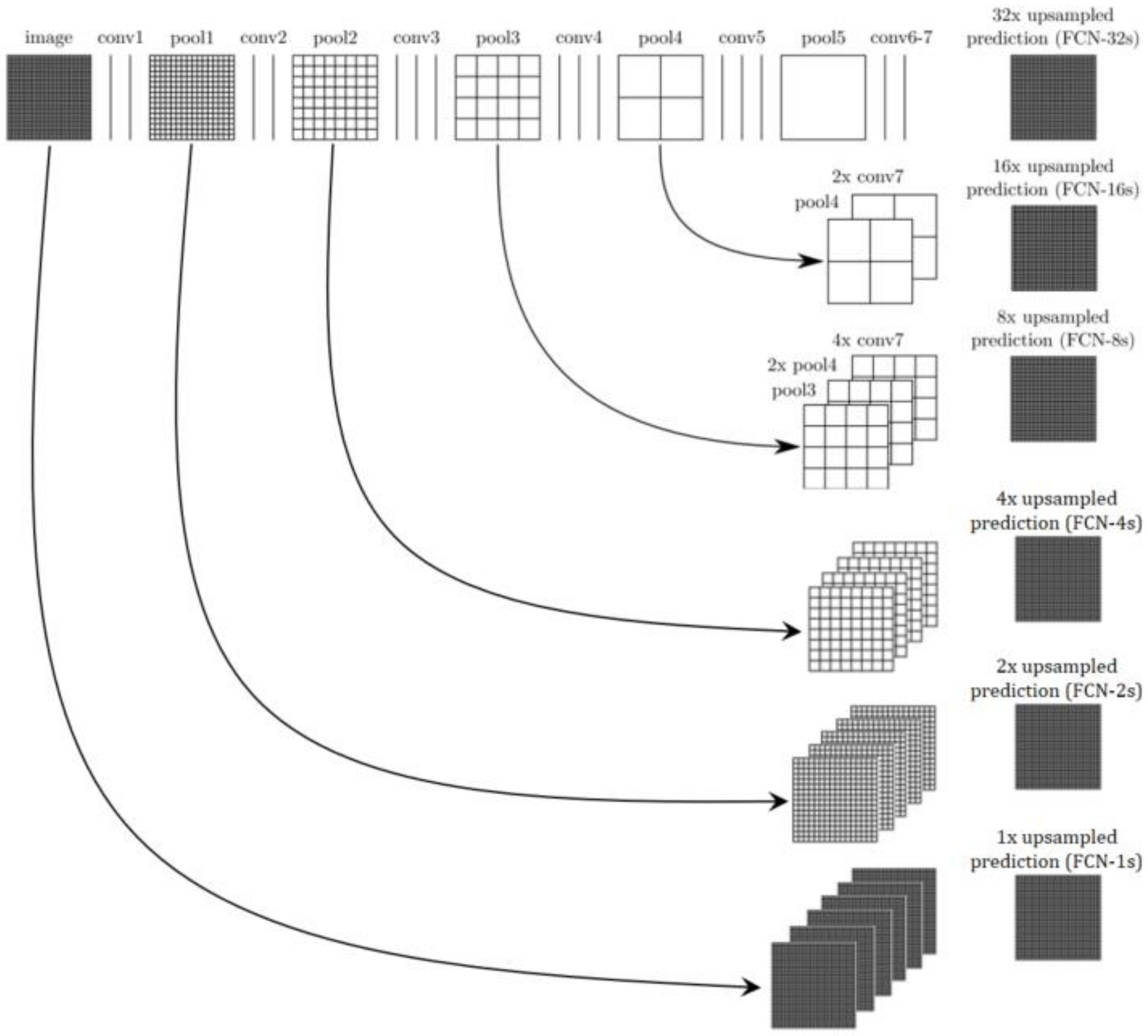


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)
4x upsampled prediction (FCN-4s)
2x upsampled prediction (FCN-2s)
1x upsampled prediction (FCN-1s)

Figure 5. FCNs with further course information combinations by skip connection.

	pixel acc.	mean acc.	mean IoU	f.w. IoU
FCN-32s-fixed	82.9	40.5	31.6	71.4
FCN-32s	82.3	**57.4**	39.0	73.0
FCN-16s	84.9	52.6	**40.3**	75.5
FCN-8s	**85.4**	50.2	39.0	**75.7**
FCN-4s	80.5	22.3	15.2	69.3
FCN-2s	76.7	8.4	5.2	62.3
FCN-1s	76.3	6.7	4.7	59.8

Table 2. Performance results of each FCN trained with VOC 2011 dataset.

experimented with FCN4s, FCN2s, and FCN1s by using the skip connection in the layer further ahead using the author's idea as shown in Figure5. The result was a little unexpected, because I thought the more information I put together, the better its performance would be. This may have been due to the information of the front layers which perceived edge features rather than higher level information, although they have much course information. The addition of low-level information may have confused segmentation tasks, even in FCN1s, the raw data was added as a skip connection, leaving the high-frequency information (which can be noise in the segmentation task) in the image intact. After FCN4s in Figure2, we can see that segmentation work has not been performed at all.

C. Comparison the optimizers between SGD and Adam

Although the paper used *SGD* to perform optimization, many recent studies have shown that *Adam* provides a much faster convergence speed and superior performance, so I also experimented with it.

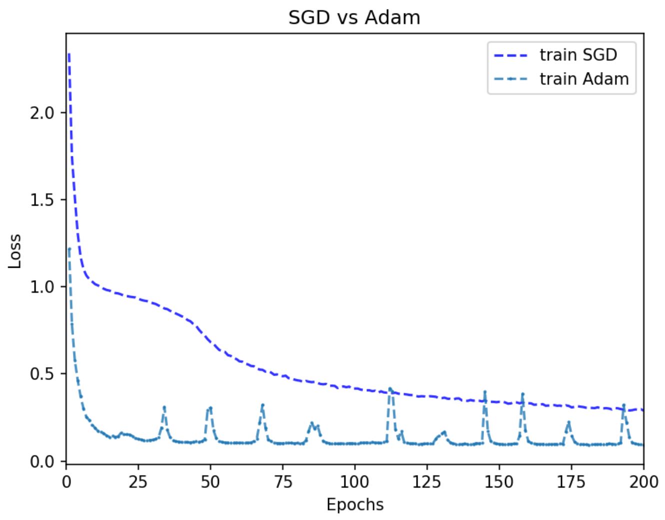


Figure 6. Loss Graph of FCN8s with SGD and Adam

As in Figure6, *Adam* optimizer was able to confirm earlier convergence than SGD, so I expected it to produce better results. *Adam*, however, was not as convergent as it could be, and the actual segmentation result was mostly worse than the SGD, as shown in Figure 7.

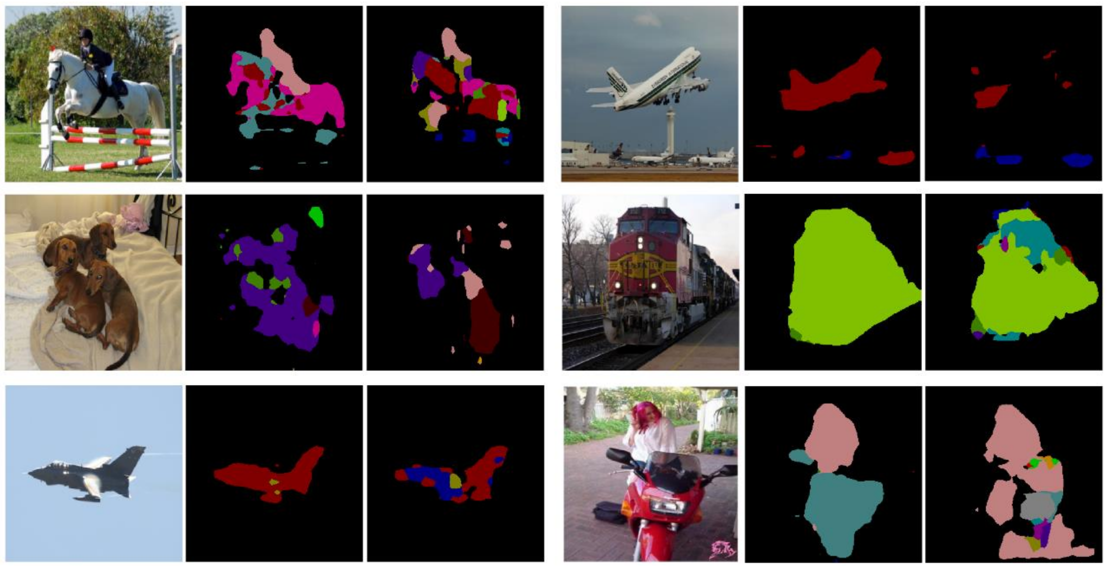


Figure 7. The results of SGD vs Adam with FCN8s.
(left: origin, middle: SGD, right: Adam)

Ⅳ. Additional approaches

One of the problems we found while implementing FCNs is that when we receive a large object that is full on the screen, it is not properly recognized as an object, but is recognized as split multiple objects as shown in Figure 8.

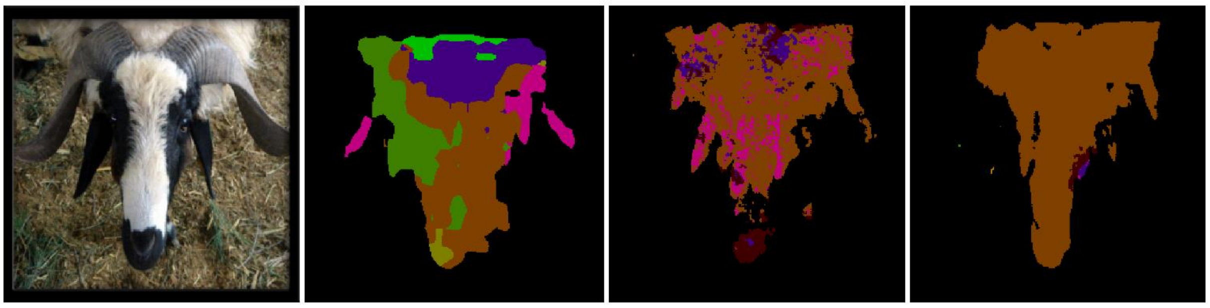


Figure 8. Segmentation result of FCN8s, DCN and DCN with random crop

To solve these problems, the "Deconvolution Network" was proposed, which enable to the robust segmentation of objects. DCN did not use the skip connections but used the same number of *Conv* and *pooling* layer as *Deconv* layer and *un-pooling* layer. When I build and test the DCN network as described in the paper, its performance has significantly decreased rather than FCN8s network. Especially, its mean IoU was less than half as low as FCN8s as described in Table3. In this case, they are trained with VOC 2012 dataset.

	pixel acc.	mean acc.	mean IoU	f.w. IoU
FCN-8s	85.1	53.9	39.7	76.5
DCN	80.2	21.9	14.7	69.2
DCN-randcrop	85.1	44.4	35.5	75.2

Table 3. Segmentation performance of FCN8s, DCN and DCN with random crop

Actually, the train image size was scaled and random cropped in the DCN paper, but this was not performed by the first implemented DCN to meet the training conditions equivalent to FCN8s. So I re-trained DCN with random crop. As a result, although the numerical performance was lower than that of FCN8s, as seen in Figure8, DCN was able to identify the larger object as being perceived as a single object.

Ⅴ. Discussion

Through these experiments, I was able to visually reproduce and verify the key distributions of each paper. FCN enables us to not only forward images regardless of image size but also easily expand task from classification to segmentation by using a fully conventional layer. Beyond what the author introduced in the paper, I have implemented and tested FCN4s, FCN2s, and FCN1s, and as Figure4 showed, the results were not good at all. This seems to be due to the additional addition of low-level or raw information that is close to noise when it comes to segmentation task. This can explain why the author mentioned only until FCN-8s in the paper.

However, the results of performances showed that the overall mean IoU results were lower than the values presented in the paper. To look back at the reasons for this, the author used local response normalization (LRN) in conv1, conv2, but I didn't apply it because recently LRN has been deprecated with the advent of BatchNorm. But I didn't apply Batch Norm either. That might make the problems with internal covariance shift. In addition, given that the segmentation quality of FCN8s and DCN(random crop) was better than FCN32s and FCN8s, we can confer that the pixel acc. and frequency weighted IoU seems to have an effect much on the performance in detail.