

# Rediscovering Semantic Segmentation 重新发现语义分割

The code and ideas discussed here resulted from some amazing collaboration with and .  
这里讨论的代码和思想来自于与和的一些惊人的协作。

emantic Segmentation is a machine learning technique that learns to identify the extents of individual objects in an image. Semantic segmentation gives machine learning systems the human-like ability to understand the contents of an image. It enables machine learning algorithms to locate the precise boundaries of objects, be it cars and pedestrians in a street image or heart, liver and kidneys in a medical image.  
语义分割是一种机器学习技术，它学习识别图像中单个对象的范围。语义分割给机器学习系统提供了人类理解图像内容的能力。它使机器学习算法能够精确定位物体的边界，无论是街道图像中的汽车和行人，还是医学图像中的心脏、肝脏和肾脏。

There are some excellent articles on the topic of semantic segmentation, perhaps the most comprehensive one is this blog:  
关于语义切分的话题，有一些很好的文章，也许最全面的是这个博客：

[**A 2017 Guide to Semantic Segmentation with Deep Learning** In this post, I review the literature on semantic segmentation. Most research on semantic segmentation use natural/real…blog.qure.ai](http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review)

Unlike most other articles on semantic segmentation, the aim of this blog post is to describe how to build a small semantic segmentation network that can be quickly trained and can be used to experiment with semantic segmentation .  
与其他大多数关于语义分割的文章不同，本文的目的是描述如何构建一个小型的语义分割网络，该网络可以快速训练并用于语义分割的实验。

This post explains how to reuse some layers of a convolution neural network (CNN) , trained to classify MNIST digits, and build a fully connected network(FCN) upon them, that can semantically segment multi-digit images.  
本文阐述了如何重用卷积神经网络（CNN）中的某些层，对MNIST数字进行分类训练，并在此基础上建立一个完全连接的网络（FCN），从而对多数字图像进行语义分割。

The dataset for semantic segmentation has been built by copying more than one 28px28px MNIST digits to a 64px84px image.  
通过将多个28px28px MNIST数字复制到64px84px图像，建立了语义分割数据集。

### Background 背景

here are different types of semantic segmentation networks and the focus here is on Fully Convolution Networks(FCNs). The first FCN was proposed in paper from Berkely. FCNs are built by extending normal convolution networks (CNN) and thus have more parameters and take longer to train than the latter. The work described here stemmed from an effort to build an FCN that is small enough to be trained on a typical laptop in a few minutes. The idea was to first build a dataset containing multiple digits in every image. The used to generate this derived dataset is here. Let us call it M2NIST (multi-digit MNIST) to avoid any confusion.  
这里有不同类型的语义分割网络，重点是全卷积网络（FCNs）。伯克利在论文中提出了第一个FCN。fcn是通过扩展普通卷积网络（CNN）来建立的，因此比后者具有更多的参数和更长的训练时间。本文描述的工作源于一项努力，即建立一个FCN，它足够小，可以在几分钟内在一台典型的笔记本电脑上进行训练。这个想法是首先建立一个包含每个图像中多个数字的数据集。用于生成此派生数据集的位于此处。让我们称之为M2NIST（多数字MNIST）以避免任何混淆。

### M2NIST M2NIST公司

Every image in M2NIST is grayscale (single channel), 64x84 pixels in size, and contains up to 3 digits from MNIST dataset. A typical image can look like this:  
M2NIST中的每个图像都是灰度（单通道），大小为64x84像素，并且包含来自MNIST数据集的最多3位数字。典型的图像可以如下所示：



The labels for the M2NIST dataset are segmentation masks. A segmentation mask is a binary image (pixel values 0 or 1),with the same height and width as the multi-digit image but with 10 channels, one for every digit from 0 to 9. The k-thchannel in the mask has only those pixels set to 1 that coincide with the location of digit kin the input multi-digit. If digit kis not present in the multi-digit, the k-thchannel in the mask has all its pixels set to 0. On the other hand, if the multi-digit contains more than one instance of the the k-thdigit, the k-thchannel will have all those pixels set to 1 that happen to coincide with either of the instances in the multi-digit. For example the mask for the multi-digit above looks like this:  
M2NIST数据集的标签是分段掩码。分割掩码是一个二值图像（像素值为0或1），其高度和宽度与多位数图像相同，但有10个通道，从0到9的每个数字对应一个通道。掩模中的k-thchannel只有那些像素被设置为1，与输入的多个数字中的数字k in的位置一致。如果多位数中不存在数字kis，则掩码中的k-thchannel将其所有像素设置为0。另一方面，若多个数字包含多个k-thdigit实例，则k-thchannel将所有这些像素设置为1，恰好和多个数字中的任何一个实例一致。例如，上面多个数字的掩码如下所示：



To keep things easy the M2NIST dataset combines digits from MNIST and does not perform any transform, for example, rotation or scaling. M2NIST does ensures that the digits do not overlap.  
为了简单起见，M2NIST数据集合并了MNIST中的数字，并且不执行任何转换，例如旋转或缩放。M2NIST确保数字不会重叠。

### The Idea Behind FCNs FCNs背后的理念

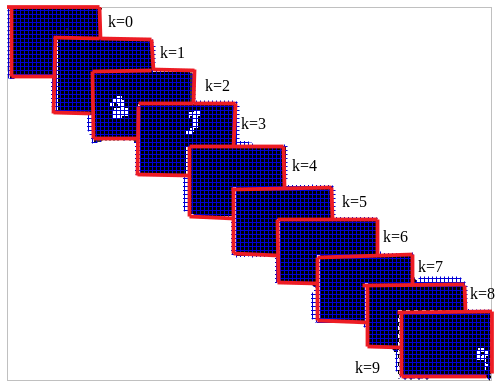
The idea behind FCNs is very simple. Like CNNs, FCNs use a cascade of convolution and pooling layers. The convolution and maxpooling layers reduce the spatial dimension of an input image and combine local patterns to generate more and more abstract ‘features’. This cascade is called an encoder as raw input is encoded into more abstract, encoded, features.  
FCNs背后的想法非常简单。与cnn一样，fcn使用一系列卷积和池层。卷积层和maxpooling层降低了输入图像的空间维数，并结合局部模式生成越来越抽象的“特征”。这个级联称为编码器，因为原始输入被编码成更抽象、编码的特性。

In a CNN, the encoder is followed by a few fully-connected layers that mix together the local features produced by the encoder into global predictions that tell a story about the presence or absence of objects of our interest.  
在CNN中，编码器后面跟着几个完全连接的层，这些层将编码器产生的局部特征混合到全局预测中，这些预测讲述了我们感兴趣的对象的存在或不存在的故事。

CNN = Encoder + Classifier  
CNN=编码器+分类器



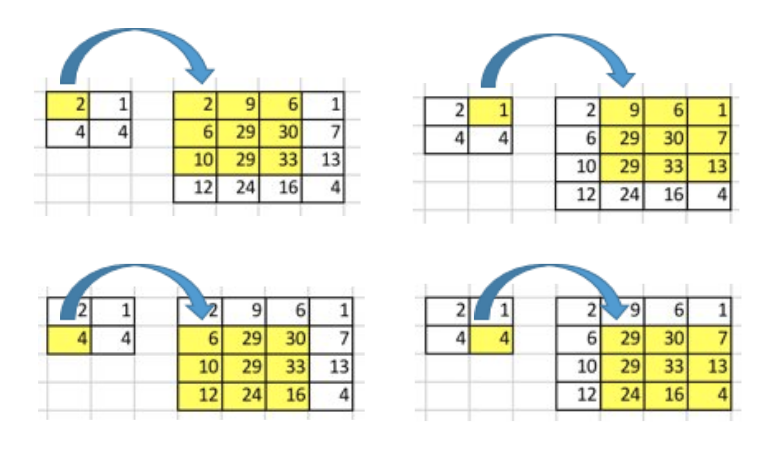
In an FCN, we are interested in predicting masks. A mask has nchannels if there are nclasses of objects that could be present in an input image. The pixel at row rand column cin the k-thchannel of the mask,\*\*\*\*predicts the probability of the pixel with coordinates (r,c)in the input belonging to class k. This is also known as pixel-wise dense prediction. Because the total probability of belonging to different classes for any pixel should add up to 1, the sum of values at (r,c)from channel 1 to nhave sum equal to 1.  
在FCN中，我们对预测面具感兴趣。如果输入图像中可能存在对象的NClass，则掩码具有nchannels。在掩模的k-thchannel的行和列处的像素，\*\*\*\*预测在属于k类的输入中具有坐标（r，c）的像素的概率。这也被称为逐像素密集预测。由于任何像素属于不同类的总概率应为1，因此从通道1到nhave和的（r，c）处的值之和等于1。



Let us understand how FCNs achieve pixel-wise dense prediction. FCNs first, gradually, expand the output features from the encoder stage using transpose convolution. Transpose convolution re-distributes the features back to pixel positions they came from. To understand how transpose convolution works, refer to this excellent post:  
让我们了解FCNs如何实现像素级的稠密预测。FCNs首先，使用转置卷积从编码器级逐步扩展输出特性。转置卷积将特征重新分布回它们来自的像素位置。要了解转置卷积是如何工作的，请参阅这篇优秀的文章：

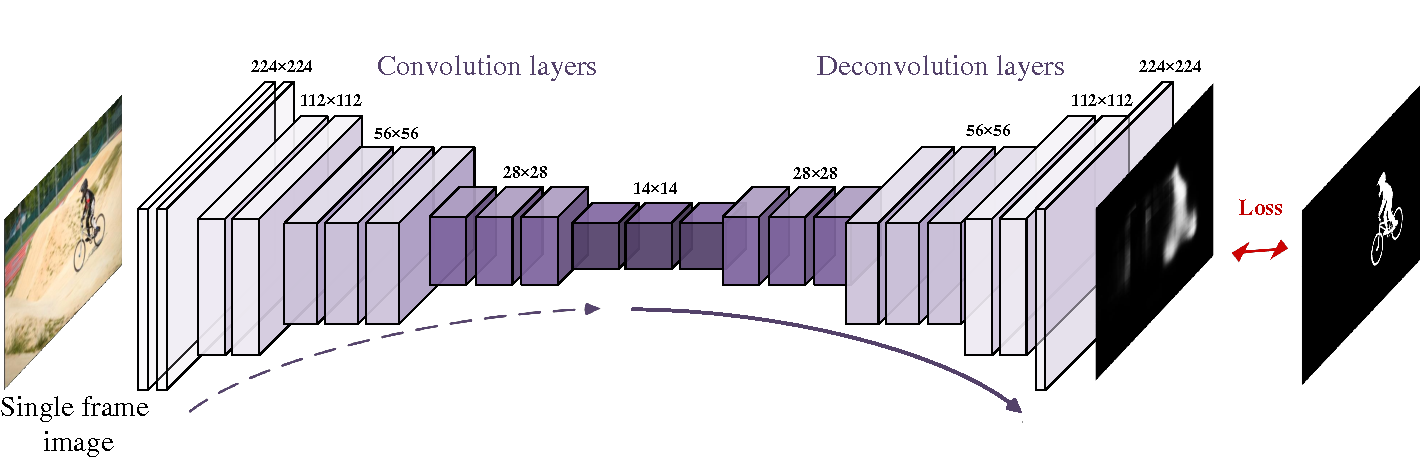
[**Up-sampling with Transposed Convolution** If you’ve heard about the transposed convolution and got confused what it actually means, this article is written for…towardsdatascience.com](https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0)

It is important to stress that transpose convolution does notundo convolution. It merely redistributes the output of some convolution in a fashion that is consistent with, but in the opposite direction of, the way in which convolution combines multiple values.  
重要的是要强调转置卷积不能撤消卷积。它只是重新分配一些卷积的输出，其方式与卷积组合多个值的方式一致，但方向相反。



The expansion or up-sampling, as it is called, is repeated, using multiple transpose convolutions, until the features have the same height and width as the input image. This essentially gives us features for every pixel position and constitutes the decoder stage of an FCN.  
使用多次转置卷积重复扩展或向上采样，直到特征具有与输入图像相同的高度和宽度。这基本上为每个像素位置提供了特征，并构成FCN的解码级。

FCN = Encoder + Decoder  
FCN=编码器+解码器



The output of the decoder is a volume with shape HxWxC,\*\*\*\*where HandWare the dimensions of the input image and Cis a hyper-parameter. The Cchannels are then combined into nchannels in a pixel-wise fashion, nbeing the number of object classes we care about. The pixel-wise combination of features values is done using normal 1x1 convolution. 1x1 convolutions are commonly used for this kind of .  
解码器的输出是一个形状为HxWxC，\*\*\*\*的卷，其中HandWare是输入图像的尺寸，Cis是一个超参数。然后，根据我们关心的对象类的数量，以像素方式将这些通道组合为通道。特征值的像素级组合使用正常的1x1卷积完成。1x1卷积通常用于这种情况。

In most cases we have C > nso it makes sense to call this operation a dimension reduction. It is also worth mentioning that, in most implementations, this dimension reduction is applied to the output of the encoder stage instead of the decoder’s output. This is done to .  
在大多数情况下，我们有C>nso，把这个操作称为降维是有意义的。值得一提的是，在大多数实现中，这种降维应用于编码器级的输出，而不是解码器的输出。就这样了。

Whether the encoder’s output is up-sampled by the decoder and then the decoder’s output dimension is reduced to nOR the encoder’s output dimension is immediately reduced to nand then the decoder up-samples this output, the final result has shape HxWxn. A Softmax classifier is then applied pixel-wise to predict the probability of each pixel belonging to each of the nclasses.  
无论解码器对编码器的输出进行上采样，然后将解码器的输出尺寸减小到或编码器的输出尺寸立即减小到nand，然后解码器对该输出进行上采样，最终结果都是HxWxn形状。然后逐像素应用Softmax分类器来预测属于每个nclass的每个像素的概率。

To take a concrete example, suppose the encoder’s output has shape 14x14x512, as in the FCN diagram above, and the number of classes, n, is 10. One option is to first reduce the thickness dimension using 1x1 convolutions. This gives us a 14x14x10 volume which is then up-sampled to 28x28x10, 56x56x10 and so on, until the output has shape HxWx10. The second option is to up-sample first, which gives us 28x28x512, 56x56x512 and so on until we reach HxWx512 and then use 1x1 convolution to reduce the thickness to HxWx10. Clearly the second option consumes more memory as all the intermediate outputs with thickness 512 will use more memory than intermediate outputs with thickness 10 that are produced with the first approach.  
举一个具体的例子，假设编码器的输出具有14x14x512形状，如上面的FCN图所示，并且类的数量n是10。一种选择是首先使用1x1卷积减小厚度尺寸。这给了我们一个14x14x10的体积，然后向上采样到28x28x10、56x56x10等等，直到输出具有形状HxWx10。第二种选择是先向上采样，这样我们得到28x28x512、56x56x512等等，直到达到HxWx512，然后使用1x1卷积将厚度减小到HxWx10。显然，第二个选项消耗更多的内存，因为厚度为512的所有中间输出将比厚度为10的中间输出使用更多的内存，这些中间输出是用第一种方法生成的。

With the encoder-decoder architecture in mind, let us see how to reuse parts of a CNN as the encoder for an FCN.  
考虑到编码器-解码器架构，让我们看看如何重用CNN的部分作为FCN的编码器。

### Repurposing an MNIST Classifier 重新调整MNIST分类器的用途

Typically, FCNs are built by extending existing CNN classification networks e.g. Vgg, Resnet or GoogLeNet. Not only are these architectures reused, their pre-trained weights are reused too, which significantly reduces the training time of the FCN.  
通常，FCNS是通过扩展现有的美国有线电视新闻网分类网络（例如VGG、RESNET或GoGoLetET）来构建的。这些体系结构不仅可以重用，而且可以重用预先训练好的权值，大大缩短了FCN的训练时间。

The recipe for converting a CNN into an FCN is described in the original as:  
将CNN转换为FCN的方法在原始版本中描述为：

We decapitate each net by discarding the final classifier layer, and convert all fully connected layers to convolutions.  
我们通过丢弃最终的分类器层来截取每个网络，并将所有完全连接的层转换为卷积。

The CNN used to build our FCN has a simple convolution-maxpooling-convolution-maxpooling-dense-dense architecture. The CNN architecture and training code can be found . The trained network is saved so that it can be reused. The network is defined like this:  
CNN用来构建FCN有一个简单的卷积maxpooling卷积maxpooling密集结构。可以找到CNN的架构和训练代码。保存经过训练的网络，以便可以重用。网络的定义如下：

To ‘decapitate’ the network, we remove the final classifier layer named dense10. The only remaining fully-connected layer named dense32is then replaced by a 1x1 convolution layer. This is something we have not discussed so far but is done in the original paper. In the code listed above, this amounts to removing the flattenand dense32layers and inserting a new 1x1 convolution with output thickness set to 32. This is equivalent to discarding everything after the last maxpooling layer pool2and adding the 1x1 convolution layer.  
为了“斩首”网络，我们删除了最后一个名为dense10的分类器层。剩下的唯一一个完全连接的层名为dense32，然后被1x1卷积层替换。这是我们到目前为止还没有讨论过的事情，但已经在原始文件中讨论过了。在上面列出的代码中，这相当于删除平坦和密集的32层，并插入一个输出厚度设置为32的新1x1卷积。这相当于丢弃最后一个maxpooling层pool2之后的所有内容，并添加1x1卷积层。

The code for building the initial versionof our FCN is on (The looks different but the gist is same). In the excerpt below, the output of the last maxpooling is extracted (viaget\_tensor\_by\_name()), it is then fed to a 1x1 convolution with output thickness 32. This convolution is the ‘replacement’ for the dense32layer found in the original CNN. Next the thickness is reduced to 10, once again using 1x1 convolution. This is the dimension reduction discussed earlier.  
构建FCN初始版本的代码已打开（看起来不同，但要点相同）。在下面的摘录中，提取最后一个maxpooling的输出（viaget\_tensor\_by\_name（）），然后将其馈送到输出厚度为32的1x1卷积。这个卷积是原始CNN中发现的dense32层的“替换”。接下来，使用1x1卷积再次将厚度减小到10。这是前面讨论的降维。

This finishes the encoder stage of our FCN. To build the decoder stage, we need to think about how and how much to scale the encoder’s output width and height.  
这就完成了FCN的编码器阶段。要构建解码器阶段，我们需要考虑如何以及在多大程度上缩放编码器的输出宽度和高度。

Although the convolution and maxpooling in the encoder come from a CNN for classifying MNIST images of size 28x28, they can be fed any image of any size. Convolution and maxpooling do not care about the height and width of their input, dense layers do but they have already been gotten rid of by decapitating the last dense layer and converting all other dense layers to 1x1 convolutions.  
尽管编码器中的卷积和maxpooling来自CNN，用于对28x28大小的MNIST图像进行分类，但它们可以被馈送任何大小的图像。卷积和maxpooling不关心它们的输入的高度和宽度，稠密层关心，但是它们已经通过去掉最后一个稠密层并将所有其他稠密层转换为1x1卷积而被去除。

When a 64x84x1 M2NIST image is fed to the encoder stage, the first convolution layer(from the original CNN) having kernel size k=5, stride s=1,and output depth f=8,produces an output with shape 60x80x8. The maxpooling layer with k=2and s=2 halves the size to 30x40x8. The next convolution with k=3,s=1,f=8produces an output with 28x38x8 and the size is again halved to 14x19x8 by the next maxpooling layer. To summarize:  
当64x84x1m2nist图像馈送到编码器级时，具有核大小k＝5、步长s＝1和输出深度f＝8的第一卷积层（来自原始CNN）产生形状为60x80x8的输出。k=2和s=2的maxpooling层将大小减半到30x40x8。下一个k=3，s=1，f=8的卷积产生28x38x8的输出，下一个maxpooling层再次将大小减半到14x19x8。总结一下：

the part of the FCN borrowed from the CNN ingests an image with shape 64x84x1 and outputs features with shape 14x19x8.  
从CNN借用的FCN部分摄取了一个形状为64x84x1的图像，并输出形状为14x19x8的特性。

The next layer in the encoder (the replacement for dense32)\*\*\*\*is a 1x1 convolution with output thicknessf=32. It recombines 14x19x8 features into new features with shape 14x19x32.  
编码器中的下一层（dense32的替换）\*\*\*\*是输出厚度f=32的1x1卷积。它将14x19x8功能重组为14x19x32形状的新功能。

The thickness of these features is then reduced (dimension reduction). This employs 1x1 convolution with thickness f=10. So the final features coming out of the encoder have shape 14x19x10. These features are then up-sampled by the decoder stage until their shape becomes to 64x84x10.  
然后这些特征的厚度减小（尺寸减小）。这采用厚度f=10的1x1卷积。所以编码器的最终特征是外形为14x19x10。然后，解码器级对这些特征进行上采样，直到它们的形状变为64x84x10。

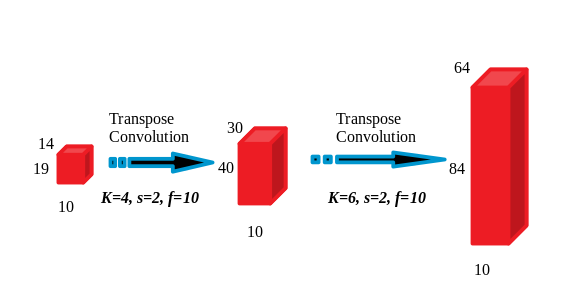
The decoder has to up-sample 14x19x10 features to 64x84x10 features.  
解码器必须将样本14x19x10功能增加到64x84x10功能。

The up-sampling is done in stages to avoid ugly patterns in the final output (mask). In our (early) implementation, the features were up-sampled from 14x19x10 to 30x40x10 and then up-sampled again to 64x84x10.  
上采样是分阶段进行的，以避免在最终输出（掩码）中出现难看的模式。在我们（早期）的实现中，功能从14x19x10向上采样到30x40x10，然后再次向上采样到64x84x10。

Up-sampling is done with transpose convolution which, like convolution, takes kernel size k,stride s, and number of filters (thickness) fas parameters. The number of filters is f=10for both transpose convolution operations, since we are not changing the thickness.  
上采样采用转置卷积，与卷积一样，它采用核大小k、步长s和滤波器数目（厚度）fas参数。对于两种转置卷积操作，滤波器的数目都是f=10，因为我们不改变厚度。

The stride is decided from the ratio of final and initial dimensions. For the first transpose convolution the ratio of heights (30/14) and widths(40/19) both is 2 so s=2is used. In the second transpose convolution, the ratios are 64/30 and 84/40, so again s=2is used.  
步幅由最终尺寸与初始尺寸之比决定。对于第一转置卷积，高度（30/14）和宽度（40/19）之比均为2，因此使用s=2。在第二转置卷积中，比率是64/30和84/40，因此再次使用s=2。

Deciding the kernel size is slightly tricky and involves some experimentation. For the first transpose convolution, using k=1exactly doubles the dimension from 14x19x10 to 28x38x10. To get to 30x40x10 k=2 and k=3 were tried but fell short. Finally k=4 worked. For the second transpose convolution, kernel size was found out to be k=6.  
决定内核大小有点棘手，需要一些实验。对于第一个转置卷积，使用k=1可以将尺寸从14x19x10扩大到28x38x10。为了达到30x40x10 k=2和k=3，我们尝试过，但没有成功。最后k=4起作用。对于第二个转置卷积，核大小被发现为k=6。

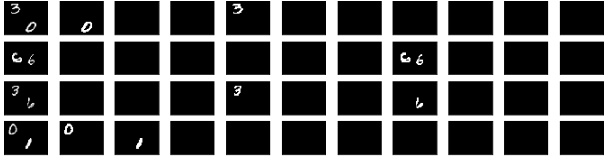


The code for the decoder is exactly two lines of Tensorflow API calls:  
解码器的代码正好是两行Tensorflow API调用：

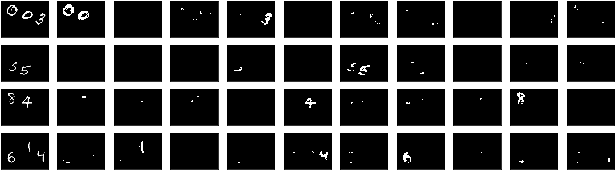
To perform pixel wise probability computation, the output of the decoder is fed to a Softmax layer. The softmax is applied along the thickness (channels).  
为了执行逐像素概率计算，解码器的输出被馈送到Softmax层。softmax沿厚度（通道）应用。

The FCN is trained using cross entropy cost function for 100–400 epochs on a Laptop with Nvidia 1050Ti GPU. The typical training time is of the order of a few minutes with 2000 samples.  
FCN使用交叉熵代价函数在一台装有Nvidia 1050Ti GPU的笔记本电脑上训练100-400个周期。典型的训练时间约为几分钟，有2000个样本。

This initial design had a high bias problems which was fixed in a later iteration. In addition there were a few logical and programming bugs that caused the network to perform sub-optimally. Here’s a snapshot from the best performing early design:  
这个最初的设计有一个很高的偏差问题，在以后的迭代中得到了解决。此外，还有一些逻辑和编程错误导致网络性能低于最佳状态。以下是性能最好的早期设计的快照：



After fixing the shortcomings in the network, it was able to perform near perfect segmentation. For example, here is the predicted output:  
在修正了网络中的缺点之后，它能够执行近乎完美的分割。例如，以下是预测输出：



### Takeaways and More 外卖等

The idea for this project came when teaching Semantic Segmentation during a connect program.  
这个项目的想法来自于在connect程序中教授语义分割。

It took around two weeks to do all the research and experimentation to get acceptable results. It was worth reinventing the wheel because the small footprint network enables hundreds or even thousands of experiments that would otherwise have been impossible, at least without massive computation power.  
花了大约两周的时间进行所有的研究和实验，以获得可接受的结果。它值得重新发明轮子，因为这个占地面积小的网络能够进行数百甚至数千个原本不可能进行的实验，至少没有巨大的计算能力。

Some of the experiments already done have given clues about which knobs to turn when tuning an FCN. Expect a follow up post with the nitty-gritty and gotchas of building the final version as well as covering ‘skip connections’ that have not been explored in this post.

The full code is available at <https://github.com/farhanhubble/udacity-connect/>. Feel free to fork.

Please leave comments below and follow 100 Shades of Machine Learning if you liked this post.