Brachial Plexus segmentation of ultrasound by CNN

David Quail
Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
dquail@ualberta.ca

Abstract

Pain management through the use of indwelling catheters that block or mitigate pain source, is a promising alternative to narcotics, which bring on several unwanted side effects. Accurate identification of nerve structures in ultrasound images is a fundamental step in effectively inserting a catheter into a patient requiring pain management. In this paper, we look into using a UNET convolutional neural network's ability to segment the brachial plexus from an ultrasound image of a patient, and compare it with a more traditional type of convolutional neural network performing the same segmentation task.

Background

Pain control in post surgery settings is a priority for health care providers. In addition to keeping the patient comfortable, pain management has other benefits. Pain control can help speed recover and may reduce the risk of developing certain complications after surgery, including blood clots and pneumonia. This is because, if pain is controlled, patients will be more able to complete tasks such as walking and deep breathing exercises.

The most common treatment of pain is the administration of narcotics. But these narcotics have a significant downside. These side effects include nausea, itching, and drowsiness.

A promising alternative to pain control is creating a nerve block. Unlike an epidural which controls pain over a large region of your body, a nerve block controls pain in a smaller region of the body, such as a limb. The nerve block is created by placing a thin catheter in the appropriate nerve region. The main advantage of these nerve blocks is that they avoid the side effects caused by narcotics, as stated above.

Creating a nerve block is done by using a needle to place a small catheter in the appropriate region. The main challenge in doing so is isolating the appropriate insertion place. Current methods involve using an ultrasound in real time to identify a nerve structure such as the brachial plexus. This requires the knowledge of a highly trained radiologist, and even then, is error prone. For these reasons, a less manual, and more accurate approach is desired.

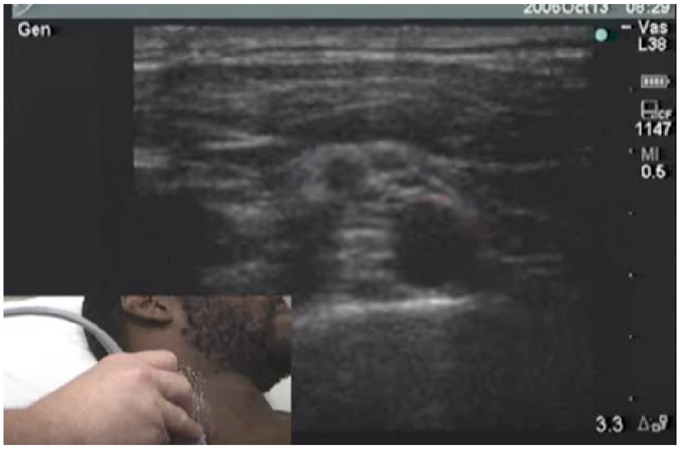


Figure 1: A patient receiving an ultrasound in an effort to identify the brachial plexus. An expert radiologist could identify the nerve in the image, but it requires years of advanced training.

A convolutional neural network (CNN) is a class of deep, feed forward artificial neural network that has been used successfully for analyzing visual imagery. They are inspired by biological processes represented within the visual cortex and in recent years, have demonstrated promise in medical image segmentation.

These networks are known to be shift and space invariant and can learn filters that were hand engineered in traditional algorithms. This independence from prior knowledge and human effort in feature design makes them an excellent candidate for an approach to automatic segmentation of nerves in ultrasound images.

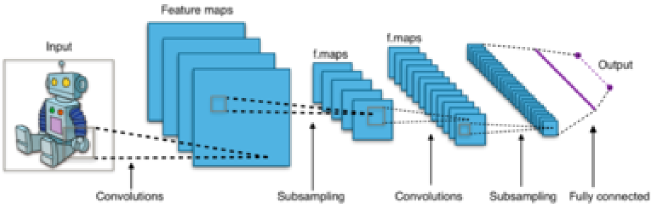


Figure 2: Typical CNN Architecture.

In this paper, we explore using a convolutional neural networks (CNN) to segment these ultrasound images. A UNET structure is considered the strongest for medical images because it applies a fully connected convolutional network architecture in a way that makes the most use out of limited data, We will tune parameters within the UNET and compare its performance to a simple Segnet structure

Experimental Setup

For the task of segmenting MRI data using our convolution neural network, we needed to consider the training data, as well as the network infrastructure for training and prediction.

The data

A Kaggle competition in 2016 made available a large training set of images where the Brachial Plexus (BP) nerve has been manually annotated in ultrasound images. The annotators were trained by experts and instructed to annotate images where they were confident about the existence of the BP landmark.. This data set includes images where the BP is not present, and as with any human-labeled data, will contain noise, artifacts, and potential mistakes from the ground truth.

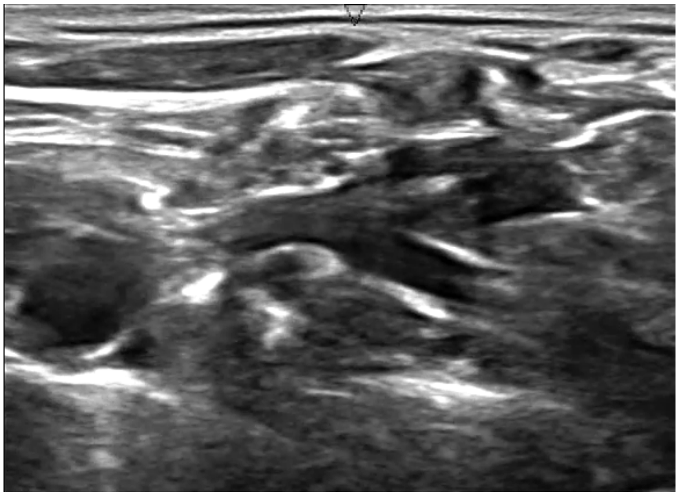


Figure 3: An ultrasound image which includes the unlabelled BP. To an untrained eye, spotting the Brachial Plexus is extremely difficult.

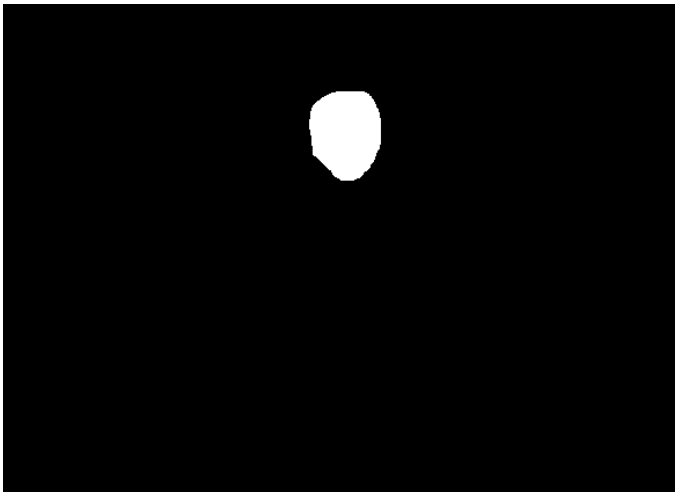


Figure 4: The human labelled mask for the ultrasound of the BP.

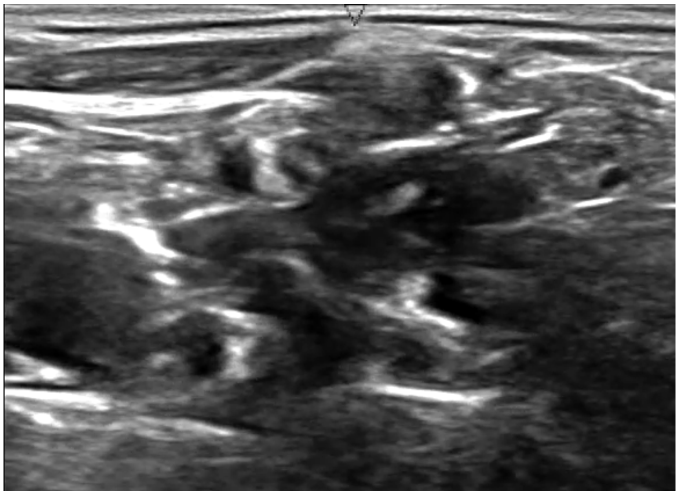


Figure 5: An ultrasound image where there is no BP. The resulting BP labelled mask will contain no segmented data.

CNN Implementation libraries

In order to build the different convolutional neural networks used to segment the ultrasound images, we wanted to use a high-level neural network API that enabled fast experimentation. The Keras API library is one such solution, and the one we chose. Keras is written in Python and is capable of running on top of TensorFlow - an open source software library for numerical computation using data flow graphs. Keras and TensorFlow support both convolutional networks and runs seamlessly on CPU and GPU.

Figure 6: Keras.

Experiments were run using an Amazon EC2 instance with a GPU using cuDNN - a deep neural network GPU accelerated library. This approach is much faster than a typical CPU because of the parallel computation it was designed form,

2 Dimensional U-NET CNN

Network design

The U-Net is a network design that relies on strong use of data augmentation to maximize the effect of the available annotated samples without overfitting the model. The architecture contains a contracting path which captures context and a symmetric expanding path that enables precise localization. It is called a ”U-net” because of it’s ”U” shaped network structure.

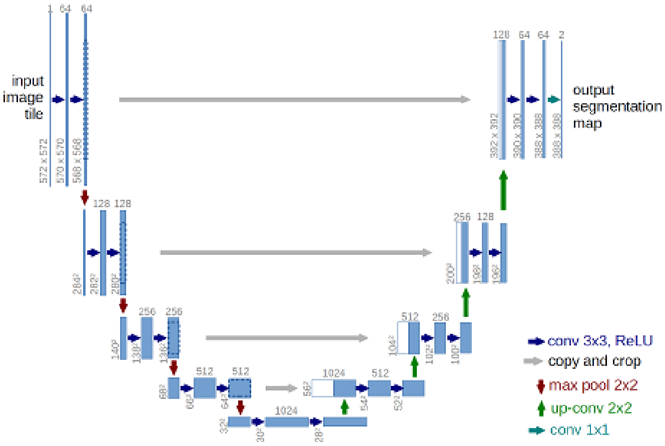
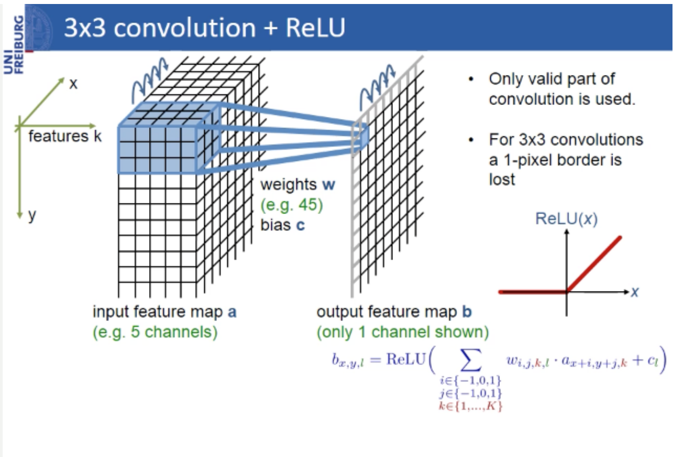


Figure 7: The high level basic U-Net architecture. The network gets its name from its U shape.

As seen above, the U-Net network takes a raw image as input and outputs an output segmentation map. It’s overall architecture is illustrated above. Like any convolutional neural network, it consists of a number of small operations, each of which corresponding to a small arrow. The image file is input into the network and the propagated through the network along all possible paths, and at the end, the ready segmentation map is output.

Each blue box corresponds to a multi channel feature layer. Most of the operations are convolutions followed by a non-linear feature activation.

Below is a diagram detailing one of the first convolution layer. The layer contains of the standard 3X3 convolution followed by a non-linear activation function. By design, only the valid part of the convolution is used, which means that for a 3X3 convolution, a 1 pixel border is lost. This allows large images to be processed in smaller tiles.



3x3 convolution + ReLU

- Only valid part of convolution is used.
- For 3x3 convolutions a 1-pixel border is lost

$$b_{x,y,l} = \mathrm{ReLU}\Big(\sum_{\substack{i\in\{-1,0,1\} \\ j\in\{-1,0,1\} \\ k\in\{1,\ldots,K\}}} w_{i,j,k,l} \cdot a_{x+i,y+j,k} + c_l \Big)$$

Figure 8: Convolution layer. Note that this diagram illustrates a 3X3 convolution layer with 3 channels. Since our ultrasound images are black and white, we will only be convolving 1 channel.

Max pooling operations as seen below are illustrated with down arrows in the architecture diagram. It reduces the x, y size of the feature map. This max pooling operation operates on each channel separately. This simply propagates the maximum activation from each 2X2 window to the next feature map. After each max pooling operation, the number of feature channels are increased by a factor of 2. All in all, these convolution and pooling operations gradually increase the “what” and decrease the “where.” In other words, it is able to identify shift and space invariant features.

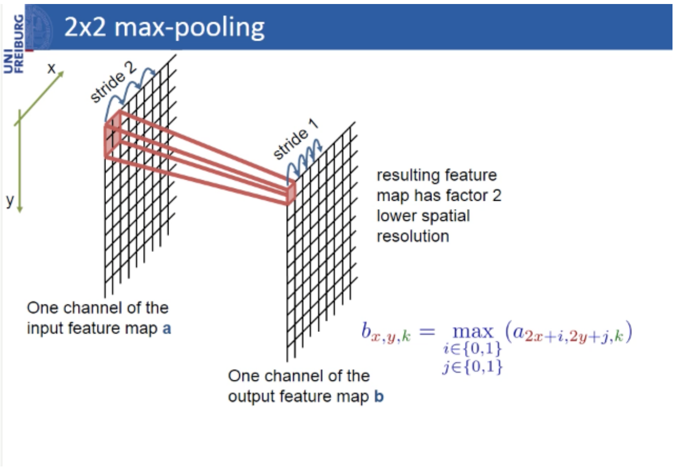

UNI FREIBURG
2x2 max-pooling
x
y
stride 2
stride 1
resulting feature map has factor 2 lower spatial resolution
One channel of the input feature map a
One channel of the output feature map b
$b_{x,y,k} = \max_{i\in\{0,1\}, j\in\{0,1\}} (a_{2x+i,2y+j,k})$

Figure 9: 2X2 max pooling.

Finally, after these contraction operations, expansion operations are performed to create a high segmentation map of the same resolution as the original image.

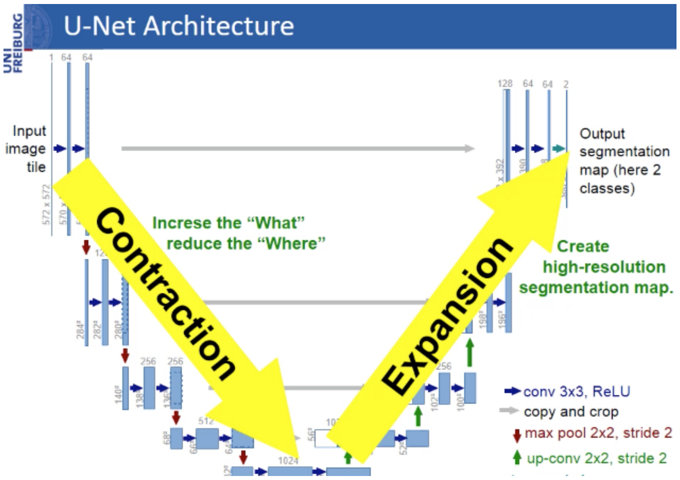
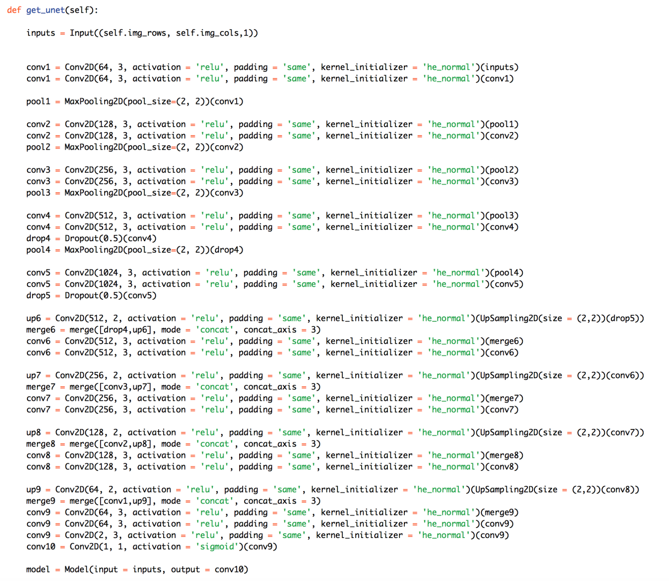


Figure 10: Expansion of the image back to its original resolution.

The following is the Python code used to create the UNET.


```
def get_unet(self):

    inputs = Input((self.img_rows, self.img_cols,1))


    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(inputs)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv1)

    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv4)
    drop4 = Dropout(0.5)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool4)
    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv5)
    drop5 = Dropout(0.5)(conv5)

    up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(drop5))
    merge6 = merge([drop4,up6], mode = 'concat', concat_axis = 3)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge6)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv6)

    up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv6))
    merge7 = merge([conv3,up7], mode = 'concat', concat_axis = 3)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge7)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv7)

    up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv7))
    merge8 = merge([conv2,up8], mode = 'concat', concat_axis = 3)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge8)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv8)

    up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv8))
    merge9 = merge([conv1,up9], mode = 'concat', concat_axis = 3)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge9)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
    conv9 = Conv2D(2, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
    conv10 = Conv2D(1, 1, activation = 'sigmoid')(conv9)

    model = Model(input = inputs, output = conv10)
```

Figure 11: Creating the UNET using Keras and Tensorflow.

A Dice coefficient (also known as Sorensen Index) is used as a loss function. The index is intended to be applied as an indicator of presence / absence of data, and over all, is a statistic used to compare the similarity of two samples. In this case, the samples are the prediction output, and the ground truth of the segmented pixels.

$$DCF = \frac{2|X \cap Y|}{|X| + |Y|}$$

A more illustrative diagram of how a U-Net is constructed for the segmentation of a medical image is seen below.

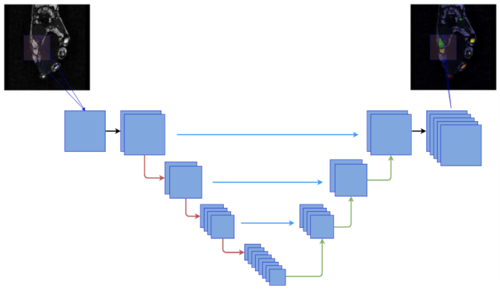


Figure 12: A U-Net architecture in a more abstract form. The actual number of feature maps illustrated is symbolic. The green arrows do un-sampling and black ones keep the original size of input, while arrows marked red are blocks of layers that down-sample their input. The cyan arrows are long skip connections which concatenate feature maps from the first stage to feature maps in the second stage. This results in the expanding path having access to twice as many features as the contracting path.

Results

The results for each experiment used the dice coefficient as a measure of accuracy. This is appropriate since it is a good statistic for measuring the similarity between the predicted BP pixels, and the ground truth BP pixels according to the training masks.

For each experiment, the training examples were randomly split. 80 percent were reserved for training, and the remaining 20 percent for testing the performance of the resulting model.

After the model has been trained, a mask is created for each test image. Examples of such masks are seen below.

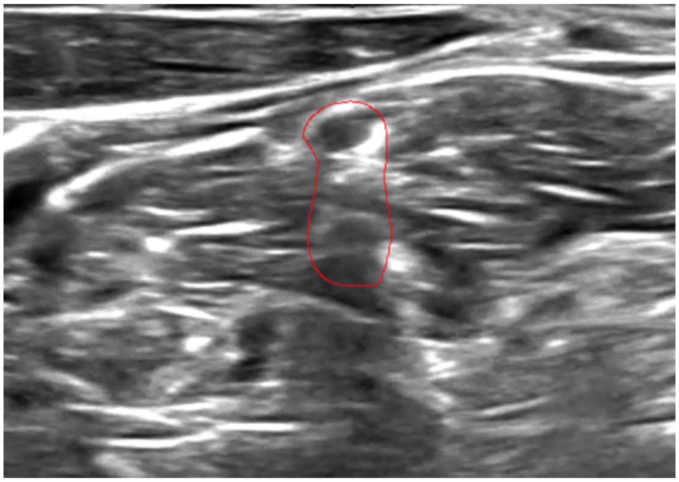


Figure 13: An example of a resulting image segmentation. For the purposes of illustration, the area inside the red circle is the area that has been segmented as the BP

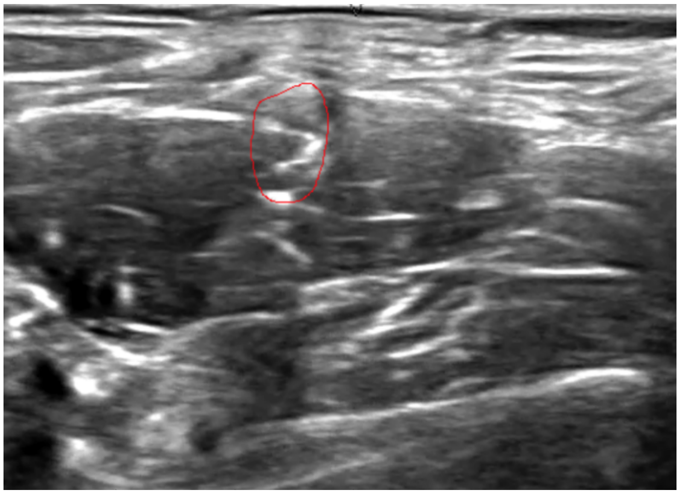


Figure 14: A second example of the BP segmentation.

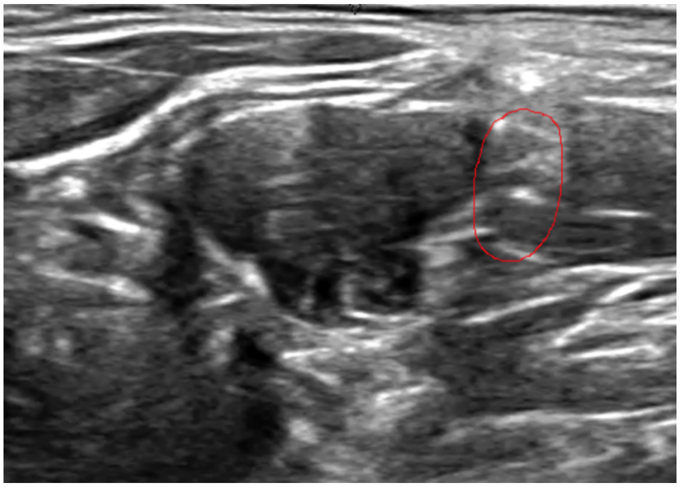


Figure 15: A third example of a BP segmentation

The dice coefficient (DCF) is used for a performance measure as well as the loss function.

Basic UNET

A basic UNET architecture as described in the 2 Dimensional UNET section above was used first. As with all experiments, the Kaggle training data was split into training examples (80 percent) and test examples (20 percent). The CNN was trained against the training set and the its performance measured using the test examples.

For our initial UNET architecture, a DCF of 0.58 is achieved after 20 epochs of training with batch sizes of 32.

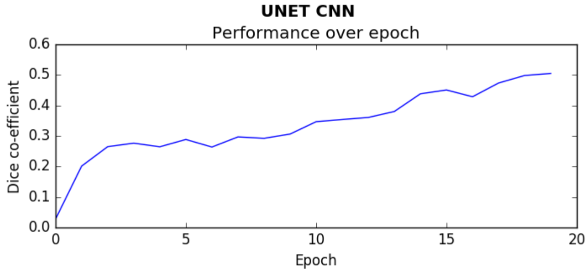


Figure 16: The learning curve over 20 epochs of training for a CNN with a basic UNET architecture.

UNET with data augmentation

Although a strength of UNET architectures is their supposed ability to perform accurately with minimal training data, we wanted to see how augmenting the training examples would effect the results. Doing so would theoretically allow the network to learn invariance to deformations without seeing them in the segmentation training data. In biomedical segmentation, this is important, since deformation used to be the most common variation in tissue and these deformations can be simulated efficiently.

Keras was used to apply transformations in the following manner:

Data augmentation here

The descriptions for the augmentation variables are as follows:

- rotation_range = Degree range for random rotations.
- width_shift_range = Range for random horizontal shifts. (fraction of total width).
- height_shift_range = Range for random vertical shifts. (fraction of total height)
- shear_range = Shear Intensity (Shear angle in counter-clockwise direction as radians)
- zoom_range = Range for random zoom.

With data augmentation, the DCF improves to 0.698.

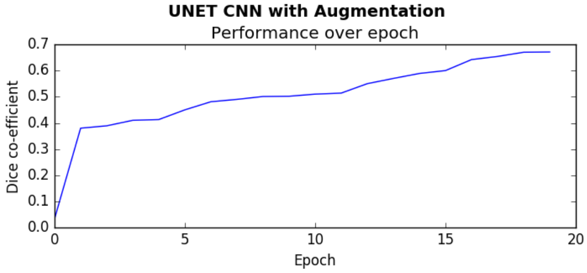


Figure 17: The learning rate when augmenting the image data using keras data augmentation abilities over 20 epochs of training.

Segnet

As stated before, a UNET architecture is generally considered to be the most appropriate architecture for medical images because of the ability to make accurate predictions with relatively low volumes of training data. So we wanted to compare the results with a more basic fully connected convolutional neural network. A basic Segnet architecture was used for our training purposes. It is not the intention of this paper to go into depth about the structure of a Segnet. More information can be found on the subject in other papers such as ”SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation.” However, it is worth giving a very basic description of a SegNet.

A novelty of SegNet is the way in which the decoder upsamples its lower resolution input feature maps. The decoder uses pooling indices computed in the maxpooling step of the corresponding encoder to perform upsampling. By doing so, the need for learning to upsample is removed. The resulting upsampled maps are then convolved with trainable filters producing dense feature maps.

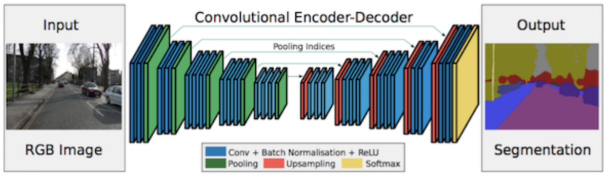


Figure 18: The architecture of a Segnet architecture.

With no data augmentation, the Segnet architecture was able to score a DCF of 0.54 after training of 20 epochs. This is quite close to the UNET architecture which was somewhat surprising. The reason that they may be close is likely in the fact that the training data is actually quite significant in this challenge. The strength of the UNET architecture is in that it requires very few training examples.

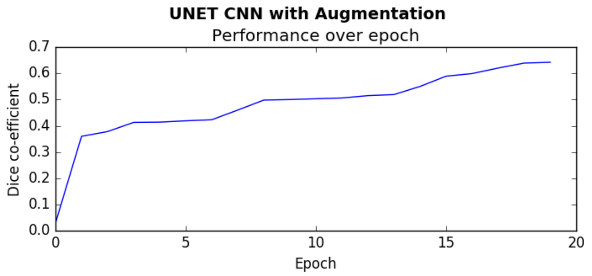


Figure 19: Accuracy using a Segnet architecture is comparable to a UNET architecture

To further test the idea that the UNET architecture remains performant on minimal data, we decreased the test data by an order of magnitude and re-ran training and prediction for both UNET and Segnet architectures.

UNET with only 100 training examples

Given only 100 training examples rather than 2300, the UNET architectures performance decreased by 5 percent to a DCF of 0.53.

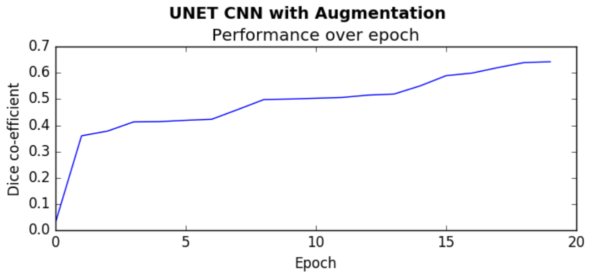


Figure 20: Accuracy using a UNET architecture with only 100 training examples.

Segnet with only 100 training examples

Given only 100 training examples, the performance of the Segnet architecture suffers much more than the UNET architecture did under similar training example constraints. The performance went from a DCF of 0.54 to 0.42.

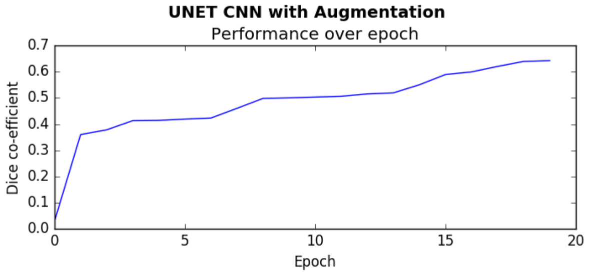


Figure 21: Accuracy of a UNET with less training data. Note that the performance is not impacted as much with less data.

Overall comparisons

Overall, the UNET architecture performed best. But it's strength was demonstrated when applied to a minimal amount of training examples.

Results after trainingt			
Network	Training size	Augment	DCF
UNET	2399	NO	0.54
UNET	2399	YES	0.59
UNET	100	NO	0.51
UNET	2399	NO	0.54
UNET	2399	YES	0.59
UNET	100	NO	0.51

Conclusions

The results using a UNET architecture are promising and demonstrate it's strengths in medical image segmentation. Given small amounts of training data, it out performed the Segnet architecture, which is unsurprising. With large amounts of data, it still outperformed the Segnet by a significant margin, which is somewhat surprising.

Future work

There are a number of different network configurations that could be considered. Thoughts for improving the accuracy could include:

- Fully Convolutional Network 8, 16, 32
- Fully Convolutional DenseNet

- Mask R-CNN
- PSPNet
- RefineNet
- G-FRNet
- DecoupleNet

In addition to trying other network structures, one could attempt to tune the parameters of each network in many different ways including:

- Changing the resolution of the images. Instead of 64X80 for example, you could try 80X112
- Attempting other data augmentations including elastic transforms.
- Likely many more. I don't pretend to be an expert with tuning CNNs.

References

Littman, M. L.; Sutton, R. S.; Singh, S.; et al. 2002. Predictive representations of state. *Advances in neural information processing systems* 2:1555–1562.

Modayil, J., and Sutton, R. S. 2014. Prediction driven behavior: Learning predictions that drive fixed responses. In *The AAAI-14 Workshop on Artificial Intelligence and Robotics, Quebec City, Quebec, Canada*.

Ring, M. 2017. Representing knowledge as predictions.

Sutton, R. S.; Modayil, J.; Delp, M.; Degris, T.; Pilarski, P. M.; White, A.; and Precup, D. 2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 761–768. International Foundation for Autonomous Agents and Multiagent Systems.

White, A. 2015. *Developing a predictive approach to knowledge*. Ph.D. Dissertation, University of Alberta.