Capstone Project

Jeff Delaney
August 23rd, 2016

I. Definition

Problem Overview

Time is the most precious resource for skilled healthcare professionals. Both physicians and patients have a major incentive to adopt data-driven solutions that can improve the quality of medical care. The analysis of ultrasound images is a tedious task that drains thousands of valuable hours from physicians that could otherwise be spent on patient critical tasks. Applying machine learning to this domain has the potential to increase the productivity of physicians by automating exhausting tasks.

An opportunity to contribute in this field occurred with the 2016 Ultrasound Nerve Segmentation[*] competition on Kaggle.

The problem at hand is to identify a grouping of nerves known as the Brachial Plexus (BP) from raw ultrasound images. Physicians must identify these nerve structures before inserting a pain management catheter prior to a surgical procedure. The training set is provided by Kaggle and contains images where the nerve has been manually annotated and outlined by trained medical professionals. Each image in the training set contains a mask that highlights the BP nerve area. As you can see from the image below, this nerve is distinguished by subtle patterns and is hardly prominent in to the untrained human eye.

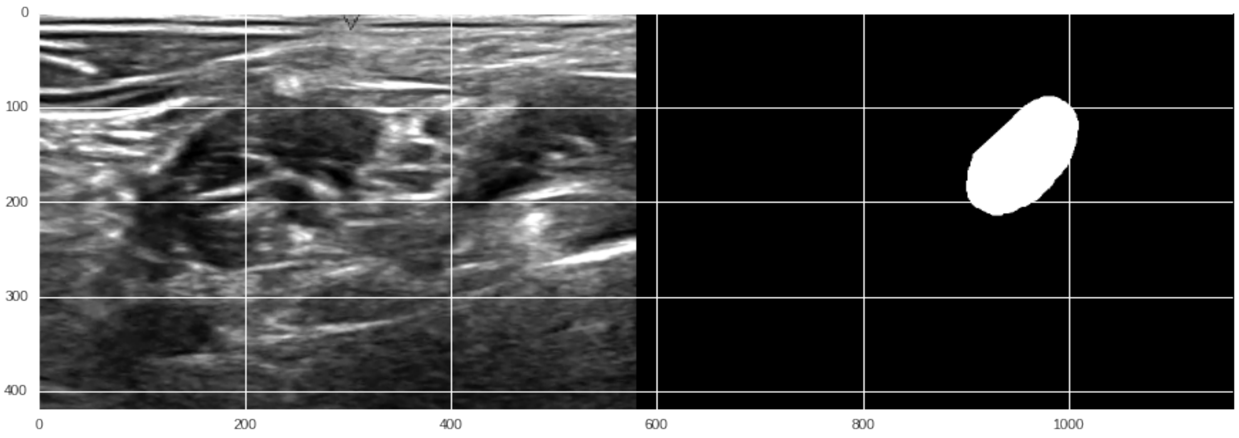

Fig. 1. Data features are raw sonogram images (left) and labels are mask annotations (right)

[*] Competition Details: https://www.kaggle.com/c/ultrasound-nerve-segmentation

The challenge is to two-fold - (1) classify the presence of the BP Nerve and (2) auto-encode a mask that outlines the nerve area. The available training data includes 5,635 sonogram images that have been manually annotated by medical professionals.

The high dimensionality and requirement for image auto-encoding in this challenge is ideal for deep learning, particularity convolutional neural networks. A variety of convolutional neural network architectures have been applied to similar image recognition tasks with impressive levels of success. The following steps have been taken to approach this problem:

1. Exploratory data analysis of the sonogram image data
2. Train deep learning models to detect and encode the BP nerve
3. Validate effectiveness of the model
4. Benchmark results on the active Kaggle competition

Metrics

The primary metric for evaluation is the **Sørensen–Dice Coefficient**. It is also commonly called the *similarity coefficient* because its initial application to was to compare the similarity of plant species in botany [1]. Today, it is a common method for analyzing pixel-wise agreement between two images.

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|}$$

The basic python implementation of the dice coefficient is shown below, where the inputs are flattened arrays representing the predicted and true mask images.

```
def dice(X, Y):

    intersection = 2 * sum(X * Y)
    coef = intersection / ( sum(X) + sum(Y) )
    return coef
```

For this task, X represents predicted pixels and Y represents actual pixels. Similarity ranges between 1 and 0, with 1 being a perfect match.

An effective model must demonstrate the ability to detect images that should be annotated, then score a high dice coefficient by auto-encoding a mask that outlines the BP nerve.
It is import to note that the dice coefficient as a competition metric places a higher penalty on false positives. In the Kaggle test dataset, a false positive results in a score of 0.0, while a false negative results in a Dice Coefficient of 0.51, which is the lower benchmark for model performance.

II. Analysis

Data Exploration and Visualization

Open Computer Vision (cv2) is a powerful open source library for performing image transformation and analysis. It was used to perform the exploratory and preprocessing tasks described in this section. The first step is to convert the raw .tif images into numpy arrays. The raw images are imported as grayscale at 580x420 pixels, producing a shape of (1, 580, 420).

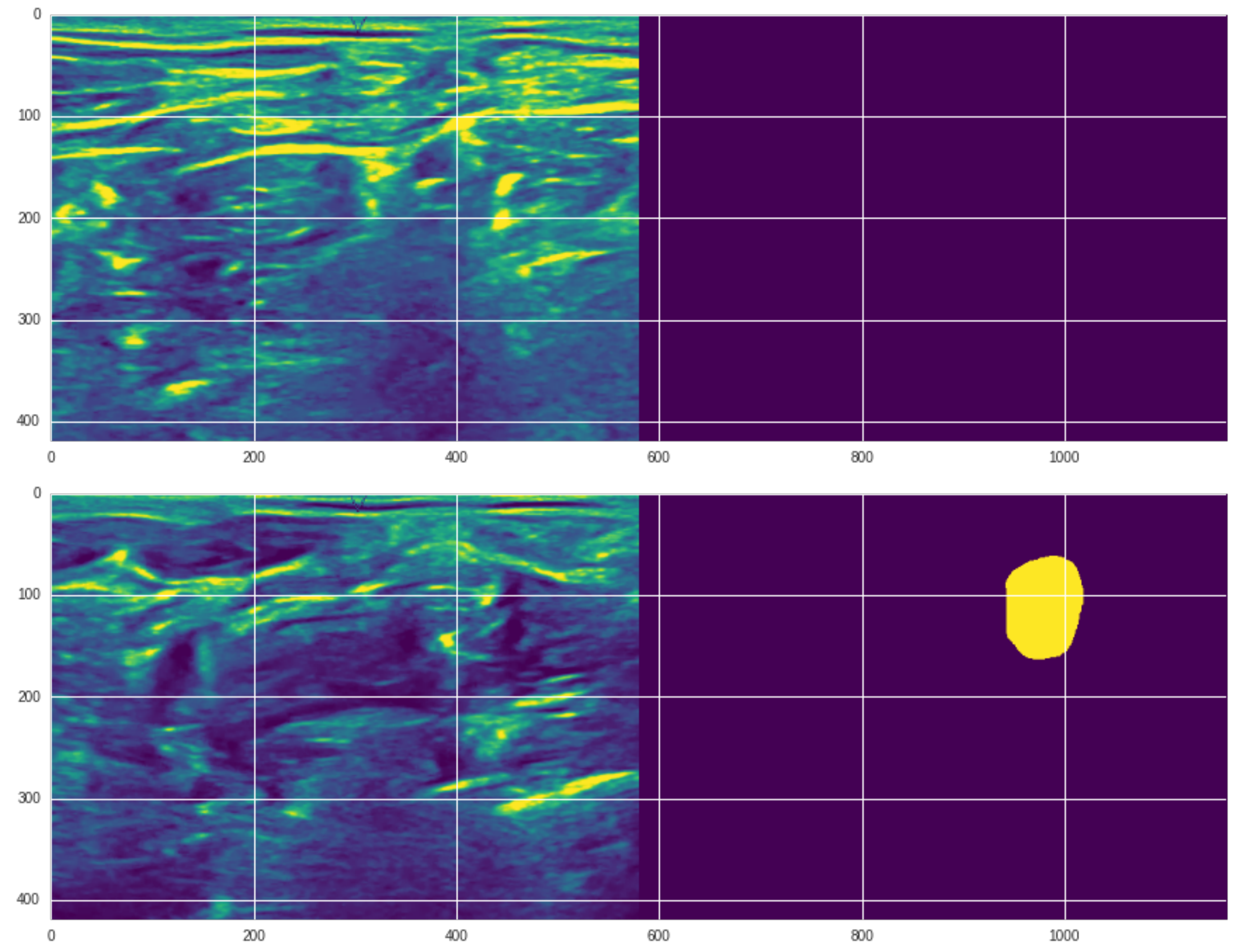

Fig. 2. A comparison of a sonogram image with a blank mask (top) versus an annotated mask (bottom). The images are quite noisy and discernable features are almost nonexistent.

After loading the raw images, it is apparent that there is a significant amount of noise. After visually inspecting 50 random images, it was difficult to identify consistent patterns and features related to the nerve. Strategic preprocessing of the images will be essential before training the auto-encoder.

One of the keys to success in the Kaggle competition is to minimize false positives. The scoring system punishes false positives far more heavily than false negatives. In the training set, 41% of image masks are annotated.

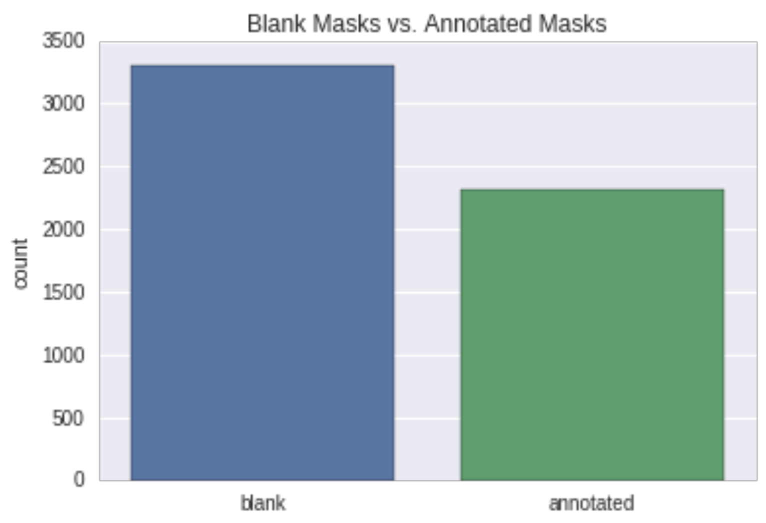

Fig. 3. Balance of blank (negative) and annotated (positive) masks in the dataset.

The auto-encoder will need to be able to learn the approximate shape and location of annotated images. The mean mask shape and location provides an idea of how the final predictions should look.

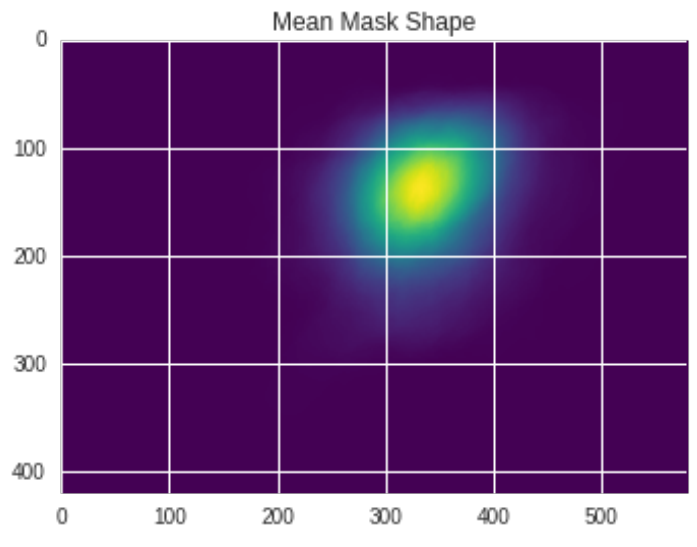

Fig. 4. The average mask shape and location across the entire dataset.

It is also useful to apply a logarithmic correction the the masks, which shows a layering of the individual annotations. This preserves the range to which mask pixels extend in the training set. Darker regions show areas where a mask is less likely to appear.

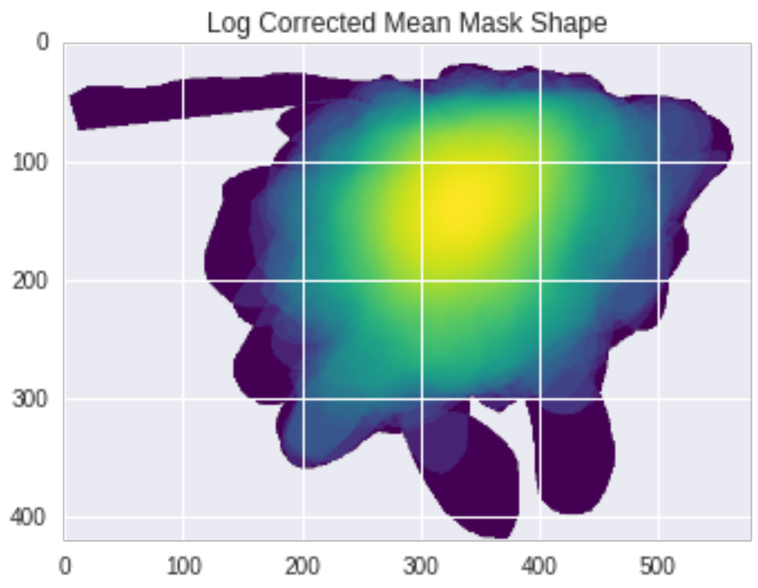

Fig. 5. Log corrected average mask, essentially a heat map layering of all masks.

The log corrected visualization highlights the potential for human error in the mask annotations. The labels for this dataset were manually annotated by physicians, which only roughly outline the BP Nerve. It is possible that positive images were either annotated with flaws or omitted.

Balancing the brightness and contrast in the images will help highlight underlying features. First, the mean distribution of pixels in blank and annotated images were compared. The distributions were similar, but blank masks were more likely to have a right skew - or more pixels closer to zero (black). This leads to a conclusion that darker images are correlated to blank masks.

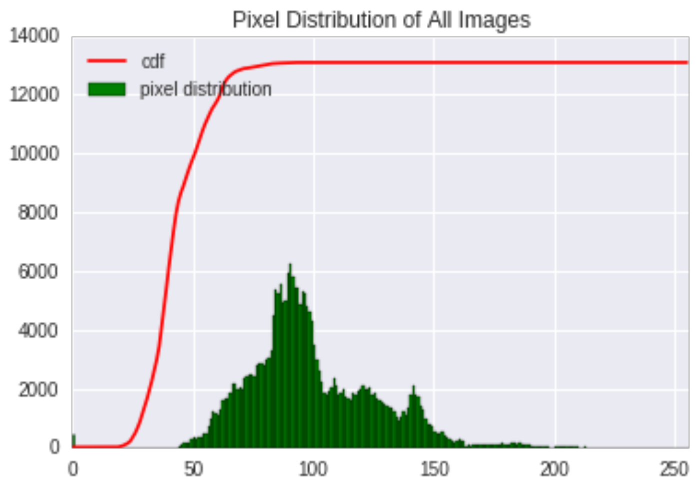

Fig. 6. Distribution of grayscale pixels in all training images.

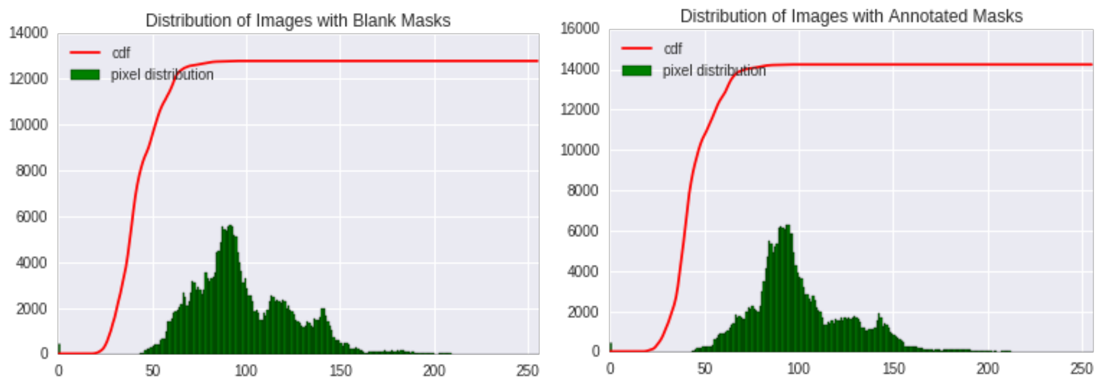

Fig. 7. Distribution comparison of blank (left) and annotated (right) images.

Histogram equalization will make the differences between lighter and darker images stand out. As a preprocessing step, it will improve the feature detection ability of the neural network by allowing edges and corners to stand out.

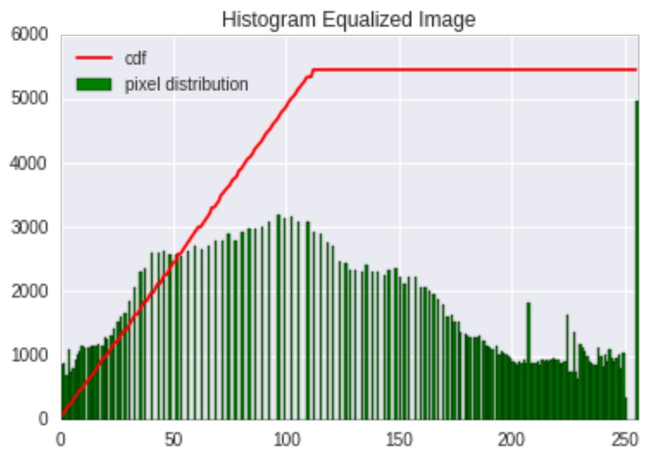

Fig. 8. An example of an image after performing histogram equalization.

Summary of Key Findings

- Annotations are present on 41% of the training data.
- Annotations are approximately elliptical in shape.
- Annotations are generally found in the upper-right quadrant of the images.
- Annotations have a larger proportion of light pixels.

Algorithms and Techniques

Several different proven convolutional neural network (CNN) architectures were experimented with during this project, but most failed to generate an impressive Dice Coefficient. Certain aspects of these architectures had to be modified to accommodate auto-encoding, but the core learning principles were maintained.

- VGG ConvNet [3]
- U-Net [5]

Of these architectures, the U-Net proved to be the most effective solution for this problem and exceeding the benchmark Dice Coefficient of 0.51. Before describing the specifics of the U-Net architecture in the implementation section, I will briefly discuss the core CNN features that have been applied in this paper.

Convolutional Neural Network Basics

CNNs excel at image recognition by applying a series of kernel filters to training images to detect important features. The filters are tiled together to provide a modified version of the input image – a process which is then repeated for each subsequent layer. After each batch of samples, the network will update the weights of features based on the target labels. The result is a graph of local connectivity patterns between a stack of convolutional layers.

After running all training samples through the network, the process is repeated with the initial weights learned from the previous iteration or epoch. The training samples will be fed to the network for multiple epochs, until the target metric (Dice Coefficient) reaches a plateau.

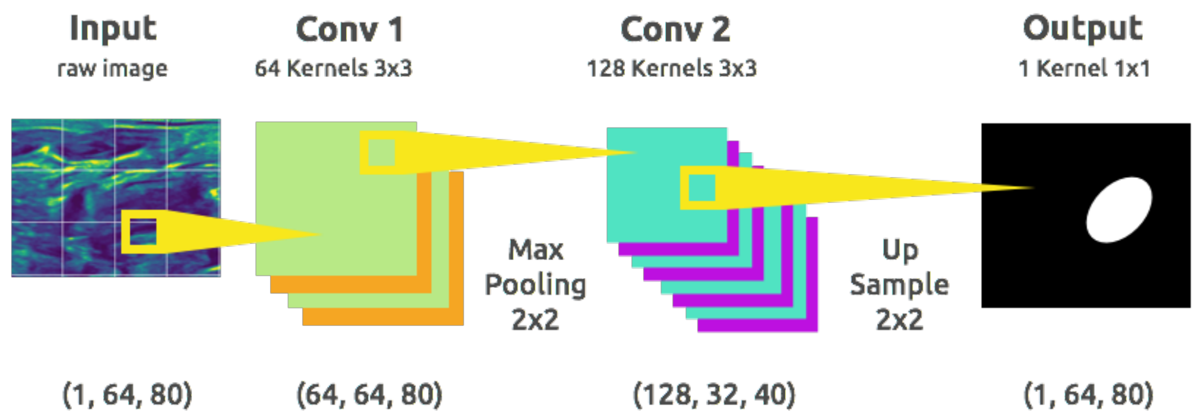

Fig. 9. The basic flow of data through the CNN. The tuples on the bottom show the change in tensor shape after each layer.

Convolutional Layer

An 2D image convolution is the element-by-element multiplication of two matrices. In this implementation, the first matrix is 3x3 section of the image and the other is a 3x3 kernel. The output is the sum of this operation, which is a component of the feature map. Visually, a kernel produces many common image processing effects, such as sharpen, blur, sobel, emboss, etc. When convolutions are applied in many different variations, they have the ability to detect important features in images because patterns are likely to emerge that correlate with the labels.

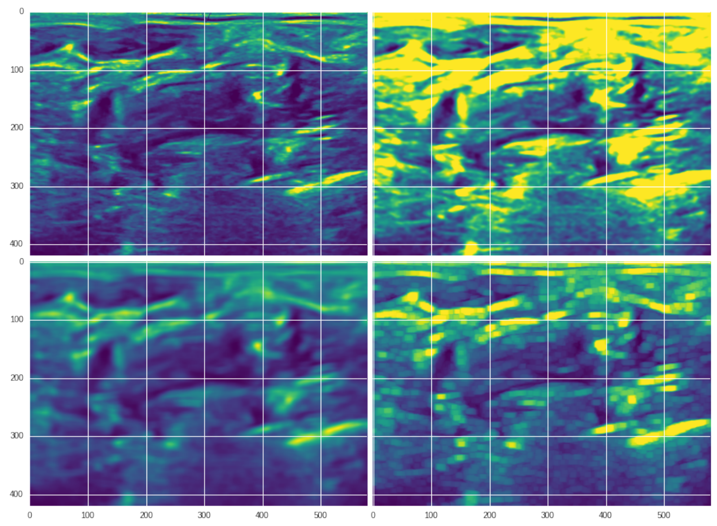

Fig. 10. Random kernel filters applied to the entire image.

Max Pooling Layer

Max Pooling is a non-linear technique for downsampling an input image. It works by reducing the feature map produced by a convolutional layer into sub-regions based on the maximum activation value. In a 2x2 pool, this allows every non-overlapping 2x2 matrix in the image to be expressed as a singular value. Max Pooling reduces the computational requirements of subsequent layers because spatial size is decreased by 75%. In other words, four pixels can now be expressed as one pixel [2].

Upsampling Layer

When designing an image auto-encoder, it is necessary to have a multidimensional output shape from the network, 1x64x80 in this case. This differs from a typical categorical cross entropy classifier that might have a shape of y, where y is a finite value of possible labels. After each max pooling layer, the shape of the output tensor will be downampled, so it is necessary to reshape the tensor. A 2x2 upsampling layer will simply copy the rows and columns of the image by size[rows] and size[columns], restoring the data to the target output shape [8].

Benchmark

A specific benchmark derived from Kaggle is a Dice Coefficient of 0.51. This score can be achieved by submitting all blank masks on the public leaderboard.

Because this project was conducted during an active Kaggle competition, the competitive benchmark was naturally the public Kaggle leaderboard. The public leaderboard is calculated with 20% of the total test data, while the private uses the remaining 80% for scoring the official competition. This allowed the model to be actively compared against models from over 900 competing data scientists.

III. Implementation

Data Preprocessing

Below is a summary of the preprocessing decisions and justification based on the exploratory data analysis.

Convert to Grayscale: Sonogram images are black/white, so a grayscale conversion does not result in information loss.

Histogram Equalization: Balancing the brightness and contrast adds clarity to the BP nerve feature in the images.

Gaussian Blur: The images contain many sharp edges and shapes resembling the BP nerve that could result in false positives. A Gaussian blur filter softens these shapes.

Resize to 64x80: Due to memory constraints, reducing the number of parameters is critical. The parameter count changes exponentially with changes to image dimensions.

Convert to Float32: Pixel values are converted to float32 to run the code on a CUDA-enabled GPU.

Implementation

Two key libraries are used to implement the CNN - Theano and Keras.

Keras is a deep learning framework that makes it possible implement the neural networks with minimal boilerplate code. Keras provides a high level wrapper over Theano and TensorFlow to facilitate fast experimentation with different CNN architectures.

Theano is a Python library developed by the Université de Montréal and designed to handle complex matrix computations. It serves as the backend for Keras and facilitates the computation of multidimensional tensors that reside on the dataflow graph [8]. It also allows the code to run on a GPU at 150x faster than CPU. For the scope of this project, it provides identical functionality as Google's TensorFlow and could be swapped out without any changes to the existing code.

Initial Approaches and Failures

Before describing the final model implementation, it is important to point out the initial failures that were experienced. The first model implementation was the VGG-16 ConvNet [3], which uses a series of convolutional layers followed by max pooling, then flattens the output into three fully connected layers. For the nerve segmentation challenge, the fully connected layers were replaced with upsampling and a 1x1 convolution to retain the shape required for auto-encoding. The VGG approach is excellent for a classifier, but it did not work well with auto-encoding due to the need to add a 16x16 upsampling layer before the final convolution. This results in masks that are encoded in the correct location, but with square-like shapes caused by 16x copying of the same pixel values. The results of the VGG-16 approach are summarized in the following table:

Model	VGG-16 [3]
Total Parameters	4,205,153
Layers	23 convolutions, 6 pooling/upsampling
Batch Size	64
Epochs	30
Training Time	48s per epoch
Training Dice	0.6466
Validation Dice	0.5750
Kaggle Dice	~ 0.58

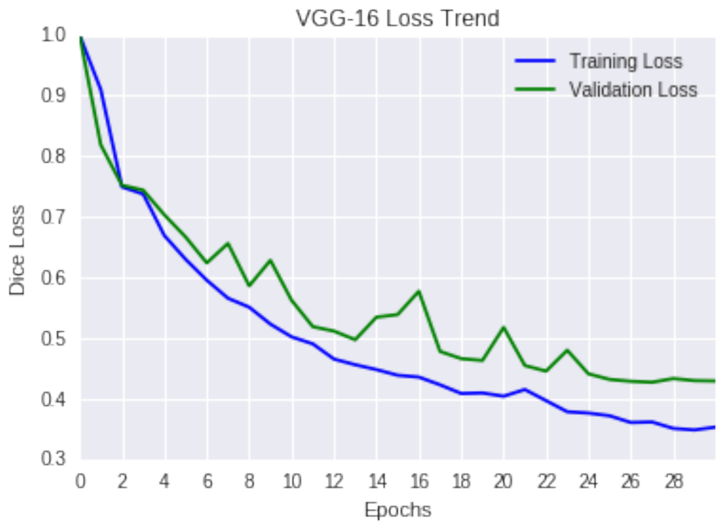

Fig. 11. Dice Loss trend on the VGG-16 over 30 epochs.

U-Net Architecture

The U-Net architecture is a neural network consisting of two separate paths of inputs. On the left contracting path, the inputs are gradually contracted with pooling, which is the typical pattern of most neural network architectures, such as the popular VGG and LeNet. The U-Net includes pairs of of 3x3 unpadded convolutions, each followed by a ReLU activation. After each pair of convolutions, a 2x2 max pooling layer is added for downsampling. After each pooling layer, the number of kernels in the following convolutions are doubled.

On the right upward path, the inputs are gradually expanded with upsampling, then concatenated to the corresponding convolution from the left path. After each concatenation (merge) layer, two additional 3x3 convolutions are applied. Finally, a 1x1 convolution is included to remap the tensor to the target output shape of 1x64x80.

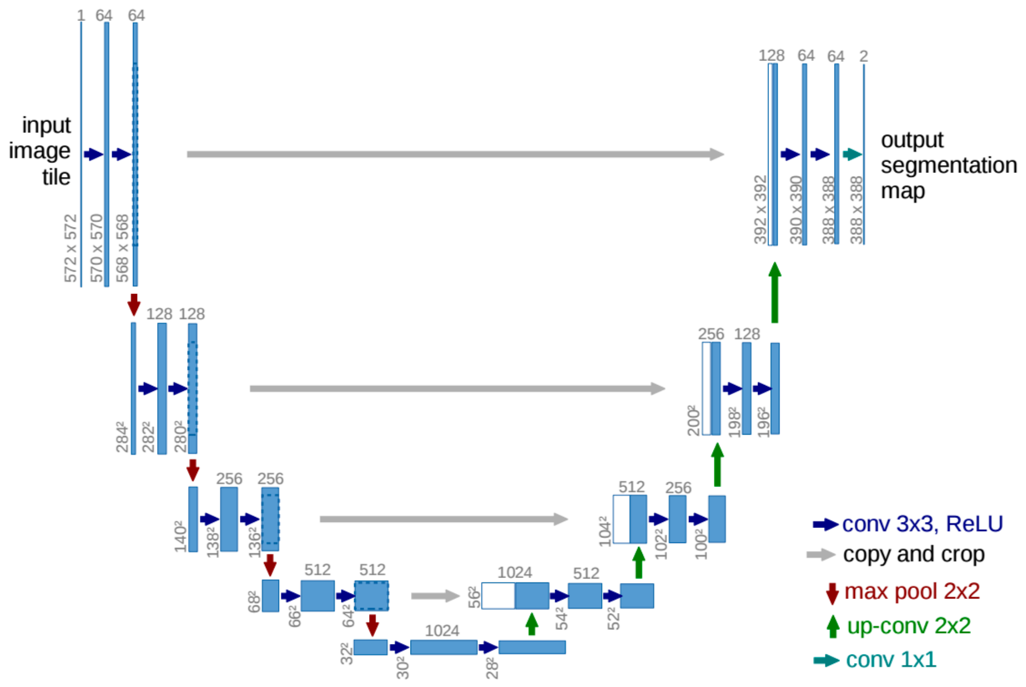

Fig. 12. U-Net Architecture [5]. Credit (Ronneberger, Fischer, Brox. 2015).

The Keras adaptation of U-Net [5, 7] allows each layer in the U-Net to be expressed with a single line of code. Compared to Fig. 11, the number of filters were scaled down by a magnitude of 2. For instance, the first layer uses 32 kernels rather than 64 to reduce the memory load.

Refinement and Control

Loss Function and Objective. The loss function drives the refinement of weights for data features. After each batch of inputs, the mean of the output array across the entire feature map is calculated based on the loss function, in this case, the Dice Coefficient. By default, Keras does not support the Dice Coefficient as a loss function. It would be possible to use 'categorical cross entropy' as a loss function for this project, but it is not an optimal benchmark for the Kaggle competition.

Optimizer. An optimizer is used to compute gradients for the loss function then apply gradients to tensor variables. There are many different approaches to neural network optimization, but Adaptive Momentum Estimation, or Adam, has proven to be optimal for this application. Adam is a stochastic gradient-based optimization that uses bias-correction and momentum [6].

The stepsize or learning rate is the only important consideration for the optimizer in this implementation. If the learning rate is too high, the Dice Coefficient will converge quickly without generating a useful output, encoding only blank masks. On the other hand, a learning rate too small will cause the algorithm to plateau at a low Dice Coefficient, resulting in random noisy masks. For this data set a learning rate of 0.00001 produces a CNN that plateaus after approximately 20 epochs.

IV. Results

Model Evaluation

Overall, the model performed relatively well given the complexity of the problem. It generated a Dice Coefficient score of 0.62557 on the final Kaggle Leaderboard, which is among the top 21% of entries. By comparison, the winning model generated a score of 0.73226.

Model	U-Net [5, 7]
Total Parameters	7,772,321
Layers	19 convolutions, 8 pooling/upsampling
Batch Size	64
Epochs	30
Training Time[*]	25s per epoch
Training Dice	0.6503
Validation Dice	0.6000
Kaggle Dice	0.62557

[*] Trained on CUDA-enabled GTX1070

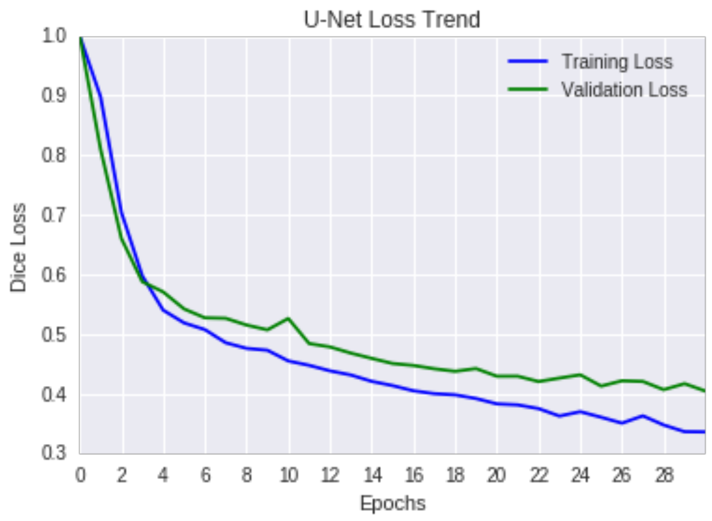

Fig. 13. The trend of Dice Coefficient loss on the U-Net over 30 epochs. Notice a more stable correlation between the training and validation loss compared to the VGG-16.

Justification

Most CNNs are used for categorical classification – not auto-encoding. A classic example is the MNIST dataset, which includes handwritten digits that can be classified from 0 to 9. This type of network can be flattened down and run through a fully-connected layer that outputs a prediction for each of the 10 possible classes.

The nerve segmentation problem is inherently more complicated because the output tensor must retain its 4D shape (Batch, 1, 64, 80). The U-Net architecture generates a large amount of overlap and duplication that is likely excessive for something like the MNIST dataset. However, it excels at dealing with the noise and asymmetry found in the nerve segmentation images. It also provides an efficient method for downsampling the images, then upsampling back up to the required shape for auto-encoding. Simply upsampling the data after pooling would result in low resolution mask predictions. The U-Net provides an innovative way to deal with this problem by merging upsampled and downsampled layers, then running additional convolutional layers after each step.

In terms of model robustness, the variance between the validation loss and training loss on the U-Net was much lower. The training Dice Coefficient loss for both the VGG-16 and U-Net achieved similar scores, but the U-Net validation loss was 0.025 lower. The loss trend on the VGG-16 also had a tendency to spike upwards at random, which is expected behavior with overfitting. This leads to the conclusion that the U-Net model is superior at generalizing the the complexity of the sonogram images.

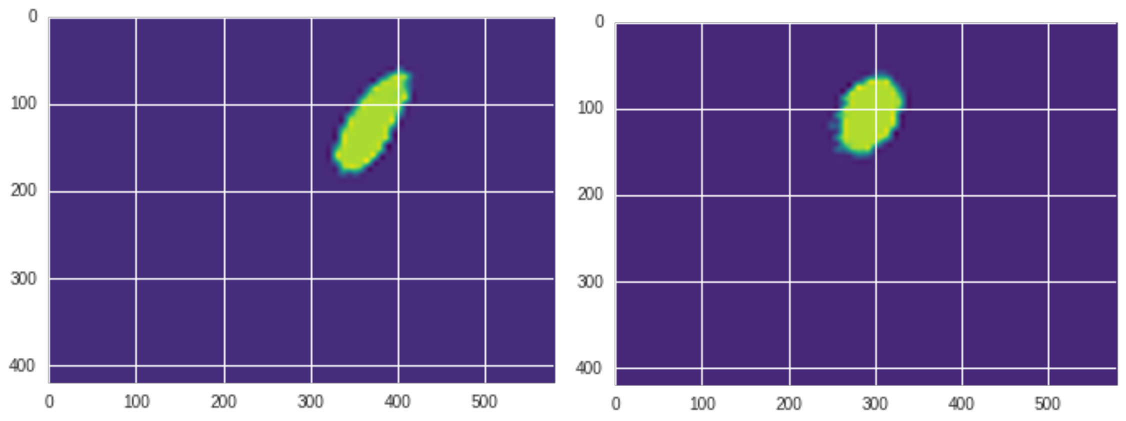

Fig. 14. Sample of predicted masks as the final output from the auto-encoder. As expected, the predictions reside in the top right quadrant.

V. Conclusion

Reflection

Overall, this was a very intriguing and challenging problem that required many hours of experimentation. I started this project without any experience using Keras and have come to appreciate the highly readable code that it produces. This was also a great introduction to Kaggle and I look forward to competing in upcoming challenges.

The initial exploratory data analysis became an extremely useful asset while approaching this problem. Understanding the expected shape and location of the image masks made it easier to validate the outputs of the auto-encoder and estimate if a certain output would achieve a high score on Kaggle. It also helped facilitate strategic preprocessing of the images based on feedback from neural network performance. Over the course of this project, many different architectures were attempted, but only the VGG-16 and U-Net exceeded the target benchmark. I went into this project expecting modifications to these architectures to yield better results, but found the defaults described in their respective papers [3, 5] to be optimal.

The most difficult aspect of the project was eliminating false positives. Ultimately, a naïve approach was taken by simply removing masks that did not meet a certain size threshold. This worked well for the competition, but may not be ideal in a real world medical imaging. Modifying the loss function and building a binary cross entropy classifier to detect blank/annotated masks were attempted, but did not manage to increase the Dice Coefficient.

Improvement

Professional medical advice. Identifying the BP nerve is not an easy task for the untrained eye. When examining two images that seem to have similar features, one could be positive and the other negative. Gaining additional insight from a trained medical professional about specific patterns could provide clues for improved image preprocessing steps and algorithm tuning.

Random image transformations. In other sonogram image applications, random image transformations reportedly had a positive impact on neural net training [5]. Most of the images of the BP nerve have masks that congregate in the upper right corner. Skewing, flipping, cropping, and rotating a small percentage of inputs could prevent overfitting and improve the overall Dice Coefficient score.

In the coming weeks, the top performers in the Kaggle competition will likely describe their models in the forums. The top scoring model was able to encode masks that were approximately 13% more similar to the test data compared to the model presented in this paper. I am looking forward to learning from their achievements and believe this could result in exciting new innovations in the medical imaging sector.

References

1. Dice, Lee R. "Measures of the amount of ecologic association between species." Ecology 26, no. 3 (1945): 297-302.

2. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “The Deep Learning Book”. Unpublished MIT Press. 2016.

3. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556. 2014.

4. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

5. Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 234-241. Springer International Publishing, 2015.

6. Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv:1412.6980. 2014.

7. Marko Jocić. Deep Learning Tutorial for Kaggle Ultrasound Nerve Segmentation competition, using Keras. Github Repository. https://github.com/jocicmarko/ultrasound-nerve-segmentation. 2016.

8. François Chollet. Keras Deep Learning library for Theano and TensorFlow. Github Repository. https://github.com/fchollet/keras. 2016.