Capstone Project

Machine Learning Engineer Nanodegree

Matthew Zhou
August 10, 2016

I. Definition

Project Overview

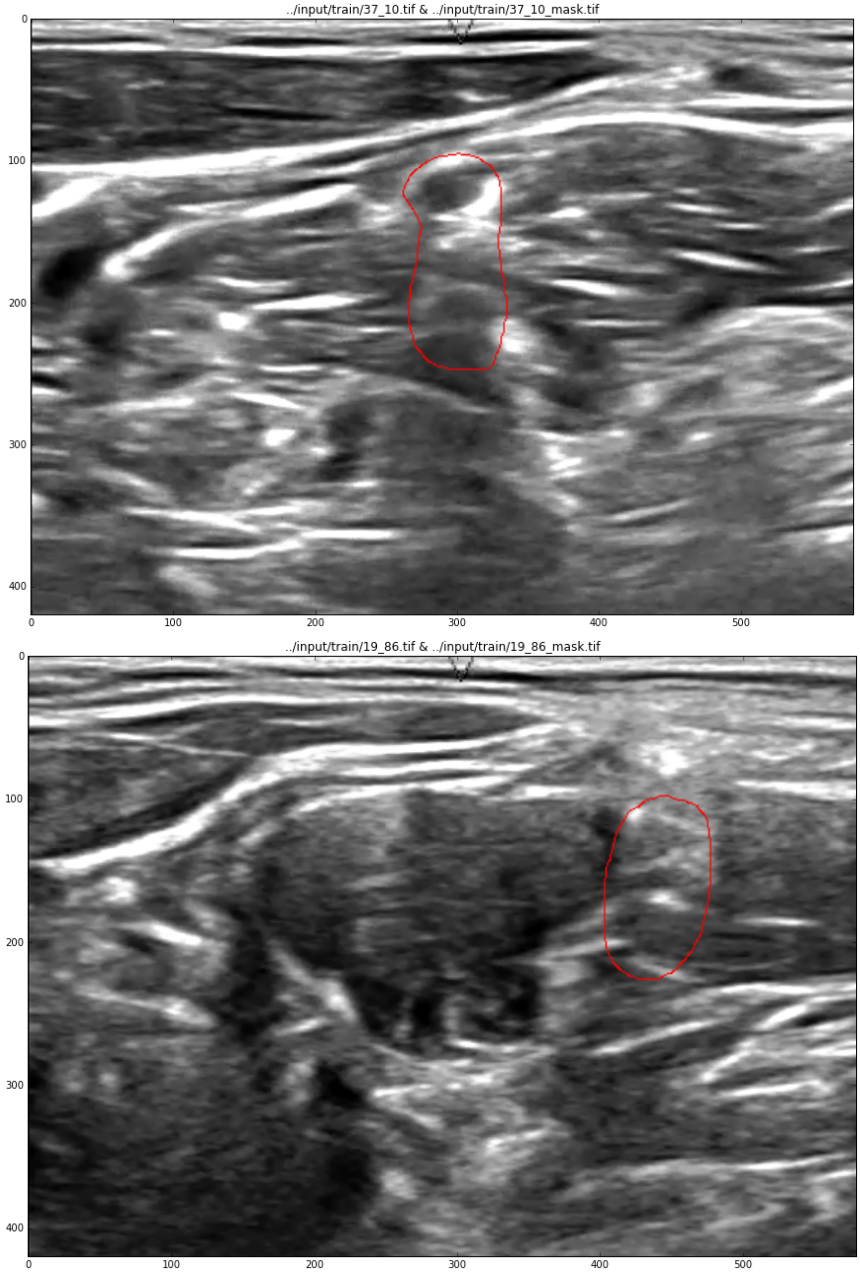
Surgery is painful and dangerous – entire teams of trained surgeons, anesthesiologists, nurses, and technicians all require years of training and skill to manage the process of physically interacting with the human body. A key aspect of surgical outcomes is not only the success of the operation but also the post-op recovery time of the patient and the pain/disability suffered during the surgery itself. Mistakes in incisions, incorrectly identified internal structures, and improper placement of surgical implements can create new damage in the patient's body. Pain management is a crucial consideration as well – patients that present with an existing tolerance to anesthesia require differing considerations for either dosage or method of pain management[1]. According to the American Society of Anesthesiologists, dosing the body with a non-targeted anesthesia can bring side effects such as nausea, vomiting, and chronic pain. An alternative method is preferable: the insertion of an indwelling catheter that targets nerves and delivers anesthesia in micro-doses locally to the peripheral nervous system.

Medical surgery is quickly incorporating image recognition technology into its repertoire of tools, with various studies demonstrating the efficacy of convolutional neural networks in identifying cancerous regions in colonoscopic imaging[2] and wound segmentation based on depth and width[3]. The identification of nerves within the neck through ultrasound images is a task that can be automated through the application of convolutional neural networks in order to apply targeted pain management during surgeries.

Problem Statement

The identification of nerves is a difficult task that usually requires extensive study and training for surgeons to be able to consistently perform. Two examples of a nerve with the "mask" delineating the nerve superimposed on the image is shown below:

[1] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3788244/
[2] http://www.ncbi.nlm.nih.gov/pubmed/20156019
[3] http://web.eecs.umich.edu/~honglak/embc2015.pdf



The nerve itself is typically identified through the surrounding tissue and structures – the goal of this project is to train a convolutional neural network that will read in the circled mask area, compare it to the surrounding tissue, and then reconstruct that circle mask for other ultrasound images.

Metrics

The Kaggle competition metric used in evaluating the final generated mask images is the Dice Coefficient, which measures the pixel-wise agreement between the generated mask image and the real value. The Dice Coefficient formula is given by:

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

Where X is the predicted set of pixels and Y is the ground truth. The Dice coefficient is 1 when both X and Y sets are empty, and the dice coefficient will be consolidated into a singular metric by taking the mean of the dice coefficients of each image in the testing set.

Two key consideration in selecting the benchmark are the concepts of precision and recall, where precision describes the true positives returned by the algorithm out of all predictions made by the model and recall describes the true positives returned out of all real true positives.

$$P = \frac{T_p}{T_p+F_p} \qquad R = \frac{T_p}{T_p+F_n}$$

For the purposes of training the algorithm, the logloss function was imported from sklearn.metrics in order to minimize the log difference between the predicted pixel set and the true value set in terms of probability. The equation for logloss is given below:

-log P(yt|yp) = -(yt log(yp) + (1 - yt) log(1 - yp))

II. Analysis

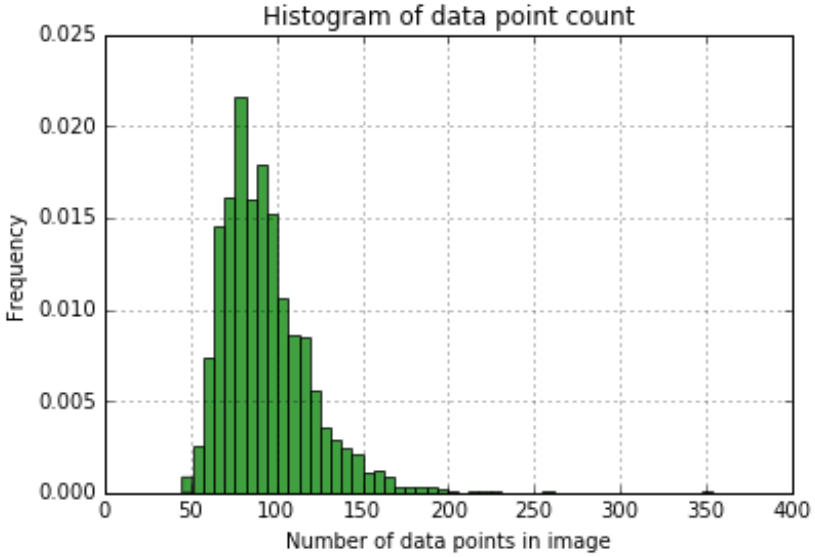
Data Exploration

The training data consisted of 5635 images that encompassed 47 distinct patients, with ~120 ultrasound images per patient. The test dataset contained 5508 images, non-segmented by patient, thus making it inadvisable to train the model by stratifying each patient. Every training image has an accompanying .tif file containing the nerve mask image (although some of the files had no masks, indicating that a nerve was not present). The original image and mask files have dimensions of 580x420 pixels, a size that had to be scaled down in order to remain computationally viable.

Exploratory Visualization

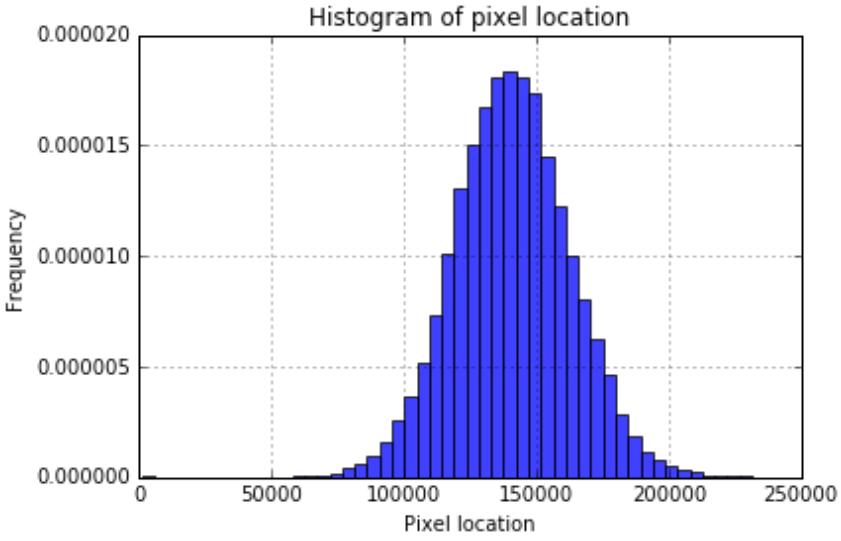
The image data consists of pixels that have been converted to grayscale binary values, a logical visualization to create would be a density plot for the pixel values after removing the images

where there was no existing target mask:



The mask images were quite small, between 50 and 250 pixels, compared to the overall ultrasound image that contained 243,600 pixels per image and also demonstrated a right skew. The algorithm is attempting to predict a relatively small amount of points compared to a large dataset and thus false positives are more likely than false negatives.

A histogram depicting the location of the target mask pixels on the overall image revealed a Gaussian distribution around a specific pixel location over all the images – the target masks tended to stay around the same region within the ultrasounds and did not tend to occur in different locations on the image. A substantial number of mask images had no identified masks – this information was excluded for this histogram.

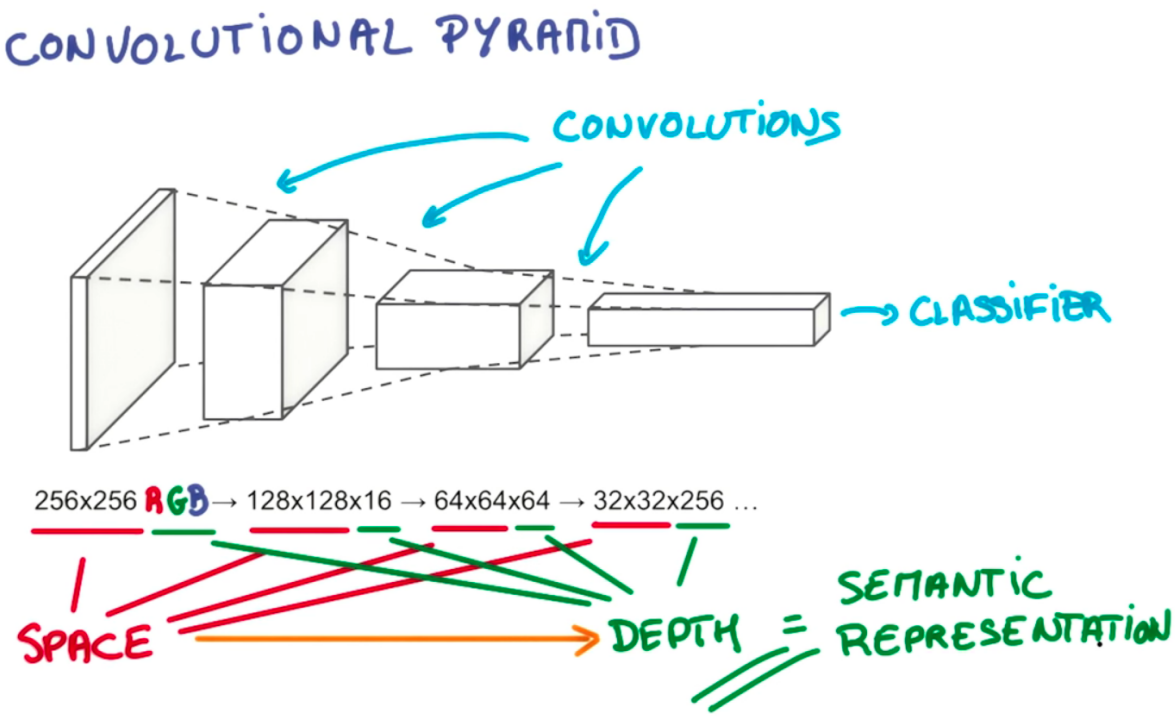


An additional complication acknowledged by the competition sponsors is that the mask images are manually identified and generated – there are labeling errors amongst the images that were

not explicitly pointed out. A structural source of error will be the inherent error within the manually selected labels that should be considered during data cleaning and feature transformation.

Algorithms and Techniques

The architecture for this convolutional neural network is a standard sliding window convolutional neural network as demonstrated in the diagram below:



Convolutional neural networks are a powerful deep learning technique that intuitively grasps indicators for image recognition problems. These networks are different from the general neural network structure due to their ability to share parameters across space. CNNs have been shown to consistently outperform other algorithms in image recognition tasks like smile detection by ranges of 19.78-33.9% in accuracy[4]. Each sequential layer in the network is able to analyze the input image across different levels of abstraction in order to identify key features that help with output classification. Sliding window convolutional neural networks are able to apply transformations to pixel data through the usage of "kernels" that make fixed length strides horizontally and vertically across an image while performing a mathematical operation on pixels within the window to generate a new image object. On a more intuitive level, these convolutions and pooling transformations allow the algorithm to capture edges, blobs, corners, etc. within images.

[4] https://arxiv.org/pdf/1508.06535.pdf

Convolutional Layer:
Takes in inputs and applies a n-specified amount of filters with a set size (usually an odd integer) in order to generate a shape with smaller height and width but greater depth. Applying filters that are the same height and width as the input dimensions would be no different than applying a regular neural network to the problem. The goal is to transform the spatial information in the input images and then applying a classifier (Fully Connected layer) on top of that output.

MaxPooling Layer:
A subsampling step that consolidates the shapes generated through the filters in the convolution layer by taking the maximal value in non-overlapping rectangles from the input image and thus reducing dimensionality.

Dropout Layer:
Dropout is a regularization technique that randomly removes a node with (1-p) probability during training – this guards against overfitting and forces the neural network to remain flexible and avoid memorizing specific patterns.

ReLu Activation:
A layer that applies a non-linear operation such as $h = \max(0,a)$ where $a = wx+b$ to the convolution layer outputs – this reduces the likelihood of the gradient to vanish.

Dense (Fully Connected) Layer:
Dense layers are traditional neural network layers where each node is connected to one another. These are usually used to consolidate information into the number of output classes for prediction. These weights are optimized through the backpropagation algorithm. In this case, a Dense layer would predict 2 classes – whether the pixel corresponded to a nerve or not.

Softmax:
A data operation applied to the outputs to transform them into probabilities that sum to 1.

Stochastic Gradient Descent Optimization:
This is an optimization technique that updates the parameters for each training example in a way that allows large fluctuations that possibly enable the model to jump out of local minima when performing gradient descent.

Adaptive Moment Estimation (ADAM) Gradient Descent Optimization:
A gradient descent algorithm that computes adaptive learning rates for each parameter and keeps an exponentially decaying average of past gradients.

Adamax Gradient Descent Optimization:
A variant of ADAM gradient descent based on the infinity norm[5].

[5] http://arxiv.org/abs/1412.6980v8

Mini-batch training:
Mini-batch training allows the algorithm to train on subsets of the training data – this is useful when there is a large amount of redundant data. Mini-batches are more computationally efficient than iterating through every training example – larger mini-batches of N reduce overfitting due to gradient averaging per batch that reduces variance in the SGD. The algorithm is able to take N-times larger step sizes compared to using a single example per iteration (step size being the learning rate of the algorithm), speeding up processing time.

Benchmark

The minimum benchmark to be surpassed is a naïve assumption that there are no nerves present in any of the test images. Since the target masks for nerves are so small compared to the overall image, the Dice coefficient for a prediction of zeroes for all images yields 0.51031 – a score that would place 634^{th}/839 data scientists on the Kaggle leaderboard. This is signaling that avoiding false positives and misidentifying nerveless regions as nerves has just as much priority as avoiding false negatives and missing real nerves. A Dice coefficient above 0.51031 signals that there is a higher chance of being right than wrong with a prediction of a nerve compared to a naïve assumption that there are no nerves.

III. Methodology

Data Preprocessing

A given file named “train_masks.csv” held the run-length encoding for each training mask in a numeric sequence that corresponded to a subject and image number. A sequence of “1 3 10 5” implies a list of value “ 1 2 3 10 11 12 13 14”, where a number is followed by the length of its sequential range. The pixels are ordered from top to bottom, then left to right. This method transforms the predicted mask outputs into a smaller file size but requires a function to convert the run-length values to image data and vice versa.

Training files were read in as grayscale images and normalized by 255 pixels to achieve a 0-1 range of values. The images were loaded into a numpy array and reshaped to fit the dimensions necessary for the CNN.

Implementation

The algorithm does not produce an exact mask area that captures a nerve region – rather, it predicts individual pixel locations where a nerve is predicted to exist. Using the dice coefficient to minimize loss encountered issues due to the predictions not being image segmentations and caused the algorithm to fail to converge. Log loss was used instead in order to measure the dissimilarity between prediction and ground truth pixel values.

Refinement

Parameter knobs that could be tuned in this CNN architecture include:

- The number of filters within each convolutional layer
- The number and width of fully connected layers
- The depth of the CNN
- Dropout Rate
- Learning Rate
- Optimizer: SGD/Adam/Adamax
- Epoch Length

IV. Results

Model Evaluation and Validation

Cross Validation scores across folds:
The training data was randomly split into 10 folds using sklearn's KFold where n-1 folds are used for training and one fold is used as a hold out set for model validation. This allows every sub-fold of the training data to be both trained and evaluated – the 10 resulting models are used to predict the test nerve masks and averaged together. The predictions for each hold out fold were saved and then compared to the entire training dataset at the end of training to provide an overall validation metric. Results from parameter tuning are showcased below in addition to intuitions about how parameter changes impacted model performance:

The test set consists of 1102 images that the model has not seen – a submission file is sent into Kaggle which returns the Dice coefficient score describing the amount of overlap between the predicted masks and the real masks.

Baseline Parameters:

- 2 Convolutional Layers with 4 and 8 filters of size 3X3 and ReLu activation
- MaxPooling layers with pool sizes of 2X2
- 50 Epochs
- Dropout rate = 0.25
- Batch Size = 32
- Image Resizing: 32X32
- Learning Rate: 1X10^-3
- Optimizer: SGD (momentum = 0.9, Nesterov Momentum = True)

Training Set Log Loss: 0.545224482683
Leaderboard Dice Coefficient: 0.56412

Table 1: Parameter Adjustment Metrics

Parameter Change	Training Set Log Loss	Dice Coefficient	Score Change
Doubling image size to 64X64	0.49795	0.55673	Decrease
Increasing batch size to 48	0.55901	0.56302	Decrease
Increasing epoch length to 100	0.52994	0.56963	**Increase**
Decreasing dropout rate to 0.15	0.52531	0.56085	Decrease
Increasing epoch length to 400	0.50840	0.57176	**Increase**
Increasing dropout to 0.35 and epoch length to 100	0.52930	0.56784	Decrease
Increasing conv layers filter numbers to 8/16	0.51406	0.56924	**Increase**
Increasing conv layers filter numbers to 16/32	0.48888	0.55538	Decrease
Increasing epoch length to 400 and LR to 3X10^-4	0.51376	0.56588	Decrease
Adding an extra convolution layer of 16 filters	0.59569	N/A	Decrease
Adding a Dense Layer of 8 units	0.55296	0.55603	Decrease
Adam Optimizer	0.50601	0.56996	**Increase**
Adamax Optimizer	0.51335	0.57223	**Increase**
Adamax Optimizer and increased epochs to 400	0.50561	0.56337	Decrease

The best model had two convolution layers with 4 and 8 filters and ReLu activation. The dropout rate was set to 0.25 and Adamax was used to optimize the parameters. The model trained on 50 epochs with batch sizes of 32. The input images were rescaled to 32X32 pixels. Evaluating this on the test set generated a score of 0.57223, well beyond the 0.51031 naïve benchmark that was set at the beginning of this project and a score that places 304th/925 data scientists on the Kaggle public leaderboard.

V. Conclusion

Reflection

A substantial portion of time was spent with the data ETL process due to the unusual problem posed by superimposing nerve masks as segmentation labels for a corresponding image rather predicting a singular value. Image augmentation proved to be a tougher challenge than originally imagined due to this segmented label aspect of the problem – any transformations for the training image would have to be mirrored for the mask label. This is not a function supported in the Keras ImageDataGenerator library, which would require manual editing in order to support this functionality.

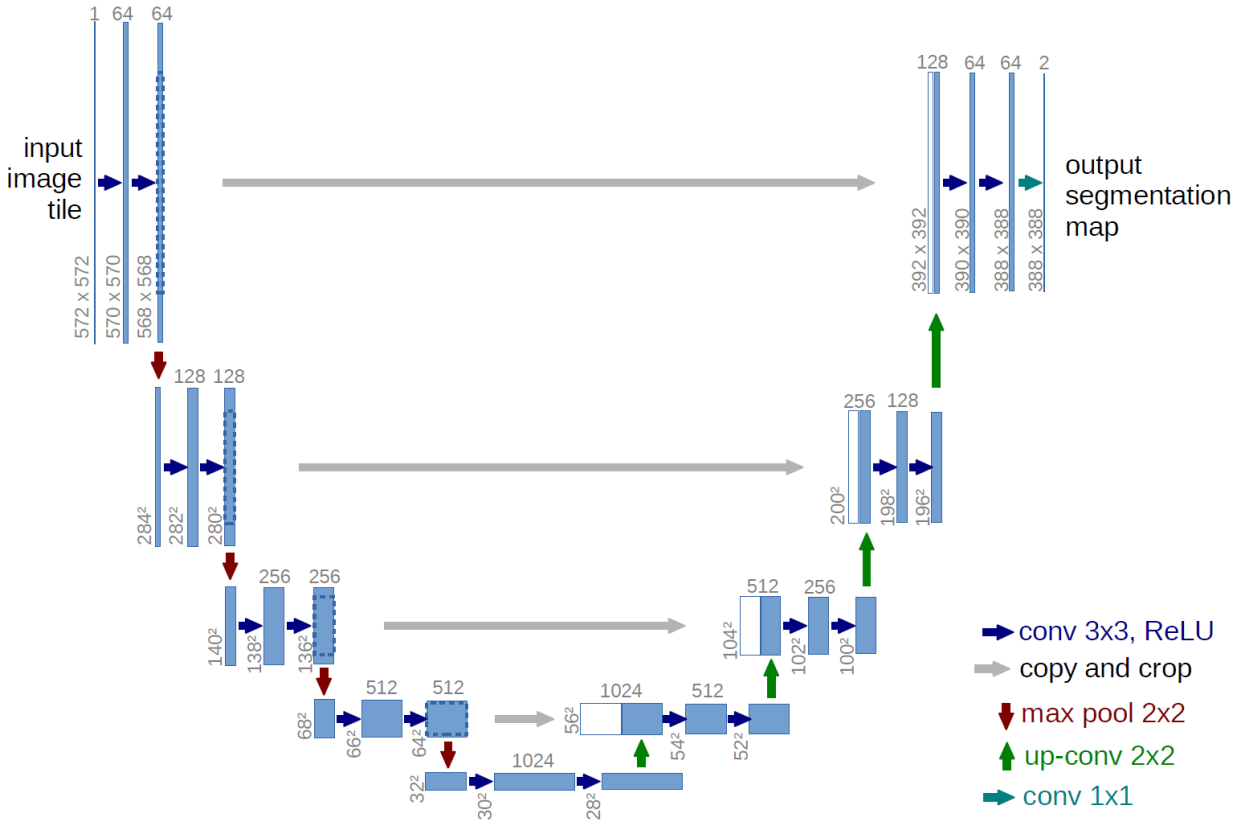
The process of parameter tuning was done in an iterative process where careful documentation of validation set log-loss metrics allowed general intuitions on what parameter adjustments led to model improvement.

The original benchmark was to beat the naïve assumption (Dice Coefficient: 0.51031) that there are no nerves present in these images, essentially challenging the model to be able to make more valid nerve predictions than invalid nerve predictions. The model succeeded in surpassing this benchmark: the top-scoring tested model scored a dice coefficient of 0.57176 and placing 304th/925 data scientists on the Kaggle public leaderboard attempting to solve this problem.

Improvement

The U-Net has been demonstrated to outperform sliding-window convolutional networks in image segmentation[6] and would provide a stronger model for building an autoencoder to solve this problem. Autoencoders are neural networks that are able to learn efficient representations, or encoding, of a given dataset and are typically used in dimensionality reduction (although they are now becoming more frequently used in generative functions). The U-Net would generate image segmentations rather than a pixel-by-pixel prediction. The U-Net architecture is described below:

[6] http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a/



The main obstacle in implementing the U-Net CNN architecture is the prohibitively expensive computational cost associated with using layers wider than 64 neurons and more than two or three layers in the network. Training with 5 layers of 32, 64, 128, 256, and 512 filters with a fully connected layer of two outputs required two and a half hours per epoch – an unfeasible amount of time without using a GPU.

Image augmentation techniques would also be beneficial – generating rotated/shifted training data would allow the neural network more flexibility to adjust to masks being in different positions within the overall image. The exploratory data analysis showed that the masks generally stayed in the same regions across a Gaussian distribution but being able to account for that difference would still reduce test error.

Overall, the convolutional neural network does a powerful job in identifying areas in which nerves might exist and generates results that are verified to be more right than wrong. This serves as a valuable step towards machines being able to identify nerves in procedures such as laparoscopic surgery or robot-assisted surgery. A new benchmark that could be used to improve this model is improving the accuracy of the labeled masks in the training data and building a model that can identify nerves in ultrasound images beyond human standards by testing average ability of physicians to identify nerves.