A Deep Learning Approach for Blood Vessel Localization in Ultrasound Images

(An ongoing basic research)

Internship Report

Azeem Bootwala

Maastricht,January,2018

Project Supervisor:-Dr.ing. Peter Brands

Contents

1. Abstract

The carotid arteries are major vessels in the neck that supply blood to the brain, neck and face. **Carotid artery stenosis** is a common problem where narrowing of the artery due to plaque buildup or atherosclerosis does not usually cause symptoms until it becomes severe. A sudden clot in the carotid artery can interrupt blood flow to the brain causing stroke. **Carotid artery aneurysm** a condition where a weak area in the artery allows the artery to bulge out like a balloon at each heartbeat . Aneurysms pose a risk in breaking causing severe bleeding and stroke.

As a result assessing carotid artery properties like arterial wall thickness, lumen diameter are important parameters through which risk of stroke or Ischemic stroke can be predicted. With the volumes of lifestyle and dietary diseases increasing rapidly, and substantial time required to detect them, the demand to automate detection and prediction arterial properties as well as the disease associated with it in order to assist the doctors for faster and accurate judgement is now more than ever.

Hence ESAOTE started investigating a state of the art machine learning approaches for detection and localization of carotid artery in ultrasound images. In this study we look at a particular sub-field in the paradigm of machine learning called Deep-learning.

Hence, the aim of this study is to train a deep neural network (DNN) model that can identify and locate a bounding box around blood vessels in US images with high level of detail. For this study a specialized family of neural network architecture (Convolutional Neural networks) proven to give best results for images was implemented.

2. Introduction

Humans glance at an image and instantly know the object information of the image and their specific location in the field of view. The human visual system is fast and accurate, allowing us to perform complex task without us realizing it.

Fast and accurate diagnosis of aneurysm or stenosis could aid doctors in better prognosis of diseases and increase throughput.

The purpose of this study is to localize blood vessel in ultrasound medical images using computer vision with deep learning techniques. Specifically localization of carotid artery using a bounding box. This deep learning model will be integrated into the embedded system and will be one of the pre-processing steps to detect the blood vessel in an image .

The application demands of the project thus requires the algorithm (the trained neural network) to be close to real-time and must be easily deployable on embedded systems.

The current state of the art object localization systems use object classifications at different scales of the image. Recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and runs the classifier at each instance of the proposed bounding box.After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections and rescore the objects based on other objects in the scene[2].

For the purpose of this project we reframed object localization as a single regression problem, from image pixels to bounding box in one go. This was done mainly for two reasons, firstly the task involved detecting only one object per image, however this approach can be easily scaled for multiple objects. Secondly the goal was to make the process as fast as possible without loss of accuracy.

3. Network Design

The network architecture particularly used was VGG network[3]. VGG stands for Visual Geometry Group (Department of engineering science, University of Oxford). The reason why VGG was the architecture of choice was because , it implemented 3x3 colvolutions. The benefit of 3x3 convolutions is that it takes into account small sub-sections of the image and hence has less parameters. Due to this advantage, it is now possible increase the depth of the network, keeping memory as a fixed resource.

Having a wide receptive field like a 11x11[4] or a 7x7 aggressively reduces the spatial resolution [3].

3.1 Training

3.1 .1 Architecture

During training the input is fixed to 512x512 images, before feeding the images into the network , each image was converted to grayscale, by taking the mean over the 3rd axis i.e 512x512x3 ->512x512x1. Each image pixel was then normalized (between 0 and 1) and fed into the neural network. The convolution stride is fixed to 1 pixel and the spatial padding of conv layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 × 3 conv. layers[3]. This can be very well explained by the following formula.

$O_w = \frac{Xw - K + 2p}{S} + 1$ and $O_h = \frac{Xh - K + 2P}{S} + 1$

where O_w and O_h are output width & height respectively

X_w and X_h are the width and height of the input image

K = Kernel size (also called the filter width or height)

P = Padding

S = Stride length

Having padding 1 for a 3x3 filter/kernel with stride 1 would result in the same size output as the input. Hence spatial resolution is preserved as we go deep into the network . Spatial pooling is performed by 6 pooling layers and not all convolutional layers are followed by pooling. Pooling parameters are fixed too. Pooling is performed over 2x2 pixel window with stride of 2 which reduces the width and the height of the inputs by half . The output of the last convolutional layer is then reshaped to a 1-dimensional array followed by 2 fully connected layers of 4096 hidden units. The training is performed with the batch size of 4 and a learning rate of 10^{-7}. However results from other hyperparameters would be explained later.

The final layer has 4 outputs , each representing X_{min},Y_{min} ,X_{max},Y_{max} for the bounding box. Note that every Convolution layer is followed by a relu (rectified linear unit) non linearity.

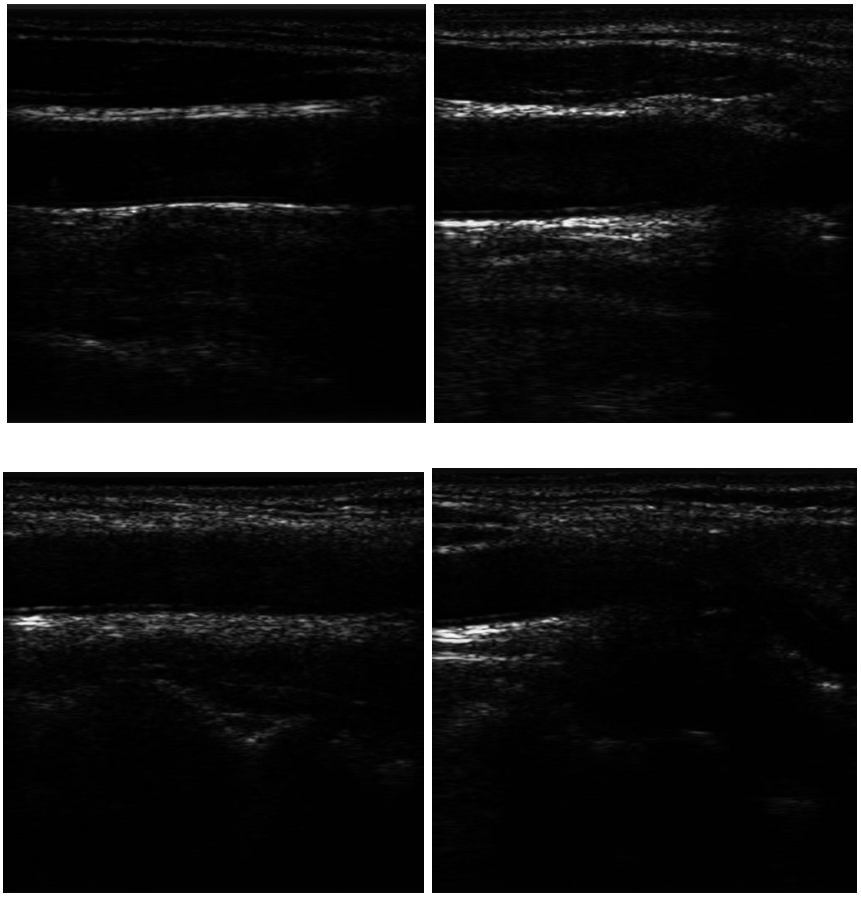


Figure 1:- Example Images, each image is 512x512 pixels. Each image poses its unique challenge from having multiple potential vessel like structures to smaller search space.

Table:-1 ConvNet configuration . The ReLu activation is not shown for brevity. But is applied after every convolution.

Layer Type	Output shape	Connected to
Input layer (None,512,512,1)	(None, 512, 512, 1)	None
Conv2d_1 (Conv2D)	(None, 512,512,64)	Input layer
Conv2d_2 (Conv2D)	(None, 512, 512, 64)	Conv2d_1
Max_pooling2d_1	(None, 256,256,64)	Conv2d_2
Conv2d_3 (Conv2D)	(None,256,256,128)	Max_pooling2d_1
Conv2d_4 (Conv2D)	(None, 256, 256, 128)	Conv2d_3
Max_pooling2d_2	(None,128,128,128)	Conv2d_4 (Conv2D)

Conv2d_5 (Conv2D)	(None, 128,128,256)	Max_pooling2d_2
Conv2d_6 (Conv2D)	(None, 128, 128, 256)	Conv2d_5 (Conv2D)
Conv2d_7(Conv2D)	(None, 128, 128 , 256)	Conv2d_6 (Conv2D)
Max_pooling2d_3	(None, 64, 64, 256)	Conv2d_7(Conv2D)
Conv2d_8 (Conv2D)	(None, 64,64,512)	Max_pooling2d_3
Conv2d_9 (Conv2D)	(None, 64,64,512)	Conv2d_8 (Conv2D)
Conv2d_10 (Conv2D)	(None, 64,64,512)	Conv2d_9 (Conv2D)
Max_pooling2d_4	(None, 32,32,512)	Conv2d_10 (Conv2D)
Conv2d_11 (Conv2D)	(None, 32,32,512)	Max_pooling2d_4
Conv2d_12 (Conv2D)	(None, 32,32,512)	Conv2d_11 (Conv2D)
Conv2d_13 (Conv2D)	(None, 32,32,512)	Conv2d_12 (Conv2D)
Max_pooling2d_5	(None ,16,16,512)	Conv2d_13 (Conv2D)
Conv2d_14 (Conv2D)	(None ,16,16,512)	Max_pooling2d_5
Conv2d_15 (Conv2D)	(None ,16,16,512)	Conv2d_14 (Conv2D)
Conv2d_16 (Conv2D)	(None ,16,16,512)	Conv2d_15 (Conv2D)
Max_pooling2d_6	(None ,8,8,512)	Conv2d_16 (Conv2D)
Reshaping(None, 8,8,512)->(None, 32768) FC_1 (None, 32768)	(None, 4096)	Max_pooling2d_6
FC_2	(None, 4096)	FC_1
Linear Output	(None, 4)	FC_2

3.1.2 Cost Function

For the current project, where we formalized object localization as a simple regression task, The cost function for the choice of parameters (weights and biases) is the mean squared error cost function (MSE).

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

m	The number of training examples
$x^{(i)}$	The input vector for the i^{th} training example
$y^{(i)}$	The class label for the i^{th} training example
Θ	The chosen parameter values or "weights" $(\Theta_0, \Theta_1, \Theta_2)$
$h_\theta\left(x^{(i)}\right)$	The algorithm's prediction for the i^{th} training example using the parameters Θ.

The MSE measures the average amount that the models prediction varies from the actual ground truth. A higher cost value indicates the model prediction is far off as compared to the ground truth and vice versa. The objective of the learning algorithm is to find the best set of weights that minimizes the cost.

Cost Function – "One Half Mean Squared Error":

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Objective:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Update rules:

$$\theta_0 := \theta_0 - \alpha \frac{d}{d\theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_0, \theta_1)$$

Derivatives:

$$\frac{d}{d\theta_0} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\frac{d}{d\theta_1} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x^{(i)}$$

Fig 2:- An example of the derivative of the cost function with respect to the parameters. Note that the given example is an explanation of a single hidden layer with 1 weight matrix and 1 bias vector.

3.1.3 Optimization

Traditional deep-learning approaches , used full gradient descent to minimize the cost/given the parameters. Which makes sense as we want to maximize the likelihood over the entire training set.

But this approach does not work on large datasets as the calculations of the gradients is 0(N).

Stochastic-Batch gradient descent is based on the fact that all samples are independent and identically distributed. So the weight update depends on a batch of training samples at a time.

Batch gradient descent :- A tradeoff between full and stochastic methods is stochastic batch gradient descent. We split our training data into **6900 batches of size 4x512x512x1**. An compute the cost based on each batch on each iteration. The cost function is less erratic as compared to pure stochastic gradient descent.

Adam Optimization:-Our choice of optimization was Adam (a variant of gradient descent), Adam Optimization is like RMSprop(another variant of gradient descent) with momentum but not exactly the same.

if we have a closer look at RMSProp the cache of the RMSProp is calculated by the formula

$$cache^{t} = decay \times cache^{t-1} + (1-decay) \times gradient^{2} \quad \text{------- (1)}$$

where decay = 0.99

$$weight = weight - lr * \frac{gradient}{\sqrt{cache+eps}} \quad \text{-------- (2)}$$

Since equation (1) is also a formalization of the running mean or exponentially smooth averages $cache_t$ is a running mean of the $gradient^2$ in other words also called the 2nd moment.

As shown in equation (2) since we take the square root of the cache during our parameter update , hence this method is called the RMSProp or the (Root Mean squared Propagation).

First and Second Moments:- In probability theory E(X) called the expected value of (X) given X being a random variable, is the mean of X.

So Mean(X)= m^t =E(X) (also called the first moment of X)

Mean(X^2) = $cache^t$ = E(X) (also called the second moment of X)

$$m^t = \mu * m^{t-1} + (1-\mu) * g^t \text{ hence } m^t \approx E(g) \text{ ---(3)}$$

where $\mu = 0.99$

Equation (3) is similar to the momentum update and equation (1) similar to the cache update . Adam Optimizer uses both the 1st and 2nd moments of (g) hence the term "Adaptive moment Estimation" or better known as Adam.

Now to explain the Adam optimization we will combine the 1st and the 3rd equation.

A typical problem with adam is the initialization of m^t and $cache^t$. If we initialize m^t and $cache^t$ to 0 , then the first output of $cache^t$ and m^t are biased towards 0. As the output is very lightly weighted towards the first gradient value.

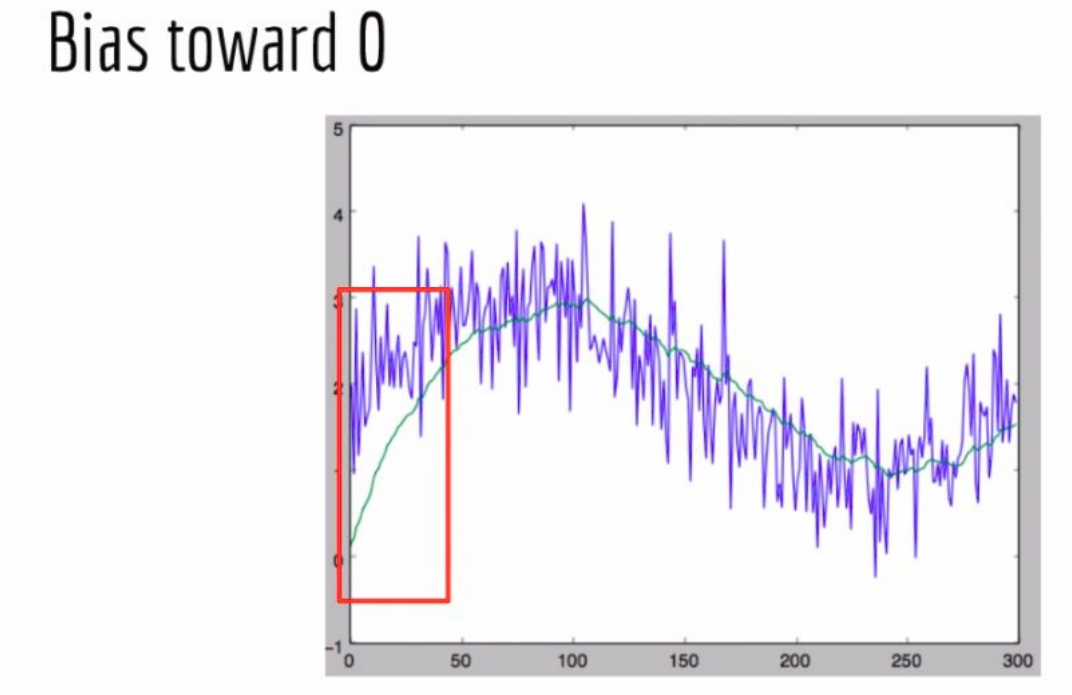


Fig 3:- Output of exponentially smooth averaging, with no bias correction on time series data.

Bias Correction:- To avoid this we implement a bias correction. Where we implement. $\widehat{cache}^t$ and $\widehat{m}^t$.

$$\widehat{cache}(t) = \frac{cache}{1-decay^t} \text{ ---(4)}$$

$$\widehat{m}^t = \frac{m^t}{1-\mu^t} \text{ -----(5)}$$

The final weight update is then

$$weight = weight - lr * \frac{\hat{m}^t}{\sqrt{\hat{cache}^t + eps}}$$

3.1.4 Batch Normalization

We know that normalizing the data for machine learning models is a good idea. Since our non-linear activation functions are most dynamic in the range of [-1,1]. We accomplish this by subtracting the mean and dividing by the standard deviation.

i.e input data = (data-mean)/std_dev

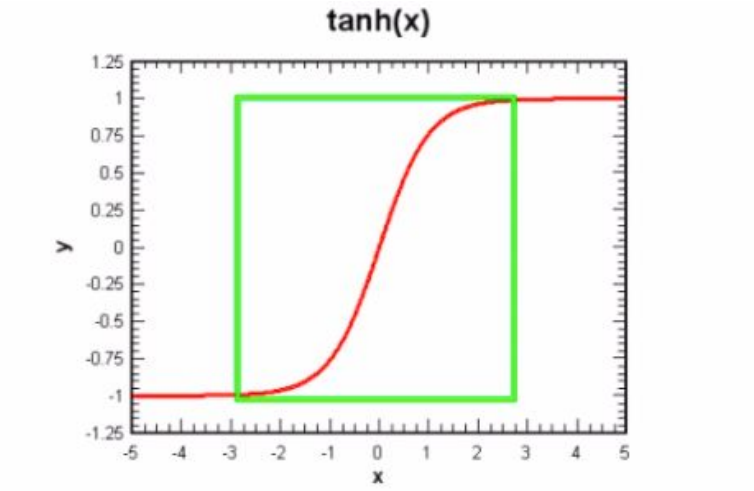
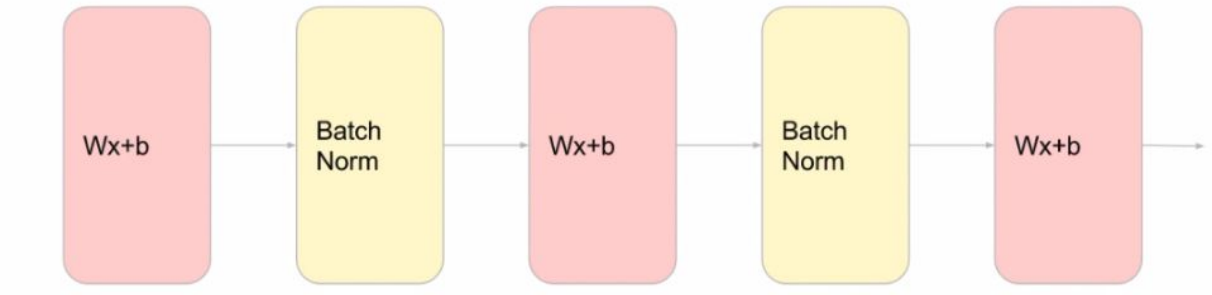


Fig4:- Shows a response of a tanh nonlinearity, with values being dynamic in the range [-1,1]



So Instead of normalizing before feeding into the neural network , we normalize out activations at every layer.

$X_b = Next\ batch\ of\ data$

$\mu_b = Mean\ of\ X_b$

$\sigma_b = std(X_b)$

$Y_b = \frac{(X_b - \mu_b)}{\sigma_b}$ where Y_b is the normalized activation with mean subtracted and divided by standard deviation.

$$Y_b = \frac{X_b - \mu_b}{\sqrt{\mu_b + eps}} \text{ ----(6)}$$

$$Y = \gamma * Y_b + \beta \text{ ----(7)}$$

We then rescale the batch by γ and add add shift β to let the neural network (gradient descent) decide the best distribution for it.

During test time, we take the running mean μ and running variance σ^2 of the entire training samples.

$$\mu = \mu * decay + (1 - decay) * \mu_b \text{ ---(8)}$$

$$\sigma^2 = \sigma^2 * decay + (1 - decay) * \sigma^2_b \text{ ---(9)}$$

At test time , the test batch is then normalized using the above mean μ and variance σ^2

So μ and σ^2 collected during train time.

$$\widehat{X}_{\text{test}} = \frac{X_{test} - \mu}{\sqrt{\sigma^2 + eps}} \text{ ---(10)}$$

$$\widehat{Y}_{\text{test}} = \gamma * \widehat{X}_{\text{test}} + \beta \text{ --(11)}$$

As we perform batch gradient descent , our distribution of X changes as we train the network. So the weights have to compensate for the shifts and this increases training time. As it requires lowering the learning rate and careful weight initialization.

When applying batch normalization let's say we have perfectly normalized data with mean 0 and variance 1. This has two effects:-

1. It allows us to increase the learning rate and faster convergence
2. It also has a mild regularization effect and inputs and weights do not need to have extreme values.

In the given project , our choice of optimization is Adam and we implemented batch normalization on a separate network .

4. Inference

Inferencing was done on 20K images. The test data was divided into 1221 batches of 4 images each. The architecture for testing and training is the same.

One exception being, in the version where batch normalization is implemented, the images are normalized using the mean and standard deviation of the whole training set.

4.1 Evaluation Metric

In order to decide whether or not the neural network provides a good enough estimate of the position of the Start point, it's necessary to know the distance between the predicted start point and the one provided by the doctor. However, if it's compared it pixel-wise, the results are different depending on the size of the image used at every moment moreover, it would not have any direct translation in the real world. Therefore, it has been decided that the comparison will be done in real millimetres. This requires obtaining the conversion factor between mm and pixels in the image, taking into account that vertical pixels do not need to represent the same distance.

The information needed are obtained from the .ZRF files. The value Zwin in the .ZRF files represents the total height in mm of the ultrasound image.

The value Zwin translates the vertical pixels per mm using the following computations .

$$\frac{pixels\ per\ mm}{1\ mm} = \frac{total\ image\ pixels\ (vertical)}{Zwin}$$

For an image of 512x512 pixels .

$$\frac{Vertical\ pixels}{1\ mm} = 14.628 \text{ pixels}$$

Since the common carotid artery lumen diameter ranged from 4.3 mm to 7.7 mm[8].Keeping the mean of 6 mm as diameter, the radial length is about 3 mm. The midpoint of the predicted bounding box (vertically) and the midpoint of the ground truth bounding box has to be in the range 0 to 3 mm, for a validation sample to be considered a good prediction.

However for better functionality of the rule based algorithm, it is preferred for the vertical midpoint of the bounding box be as close to the center of the lumen. Hence we keep this threshold to 2 mm which translates to roughly 28 pixels.

$$Y = \frac{Y_{MAX} + Y_{min}}{2}$$

$$Y_{pred} = \frac{Y_{max\ pred} + Y_{min\ pred}}{2}$$

$$if \left| Y - Y_{pred} \right| \{<= 28\ pixels\ then\ 1 = good\ prediction$$

$$else = 0\}$$

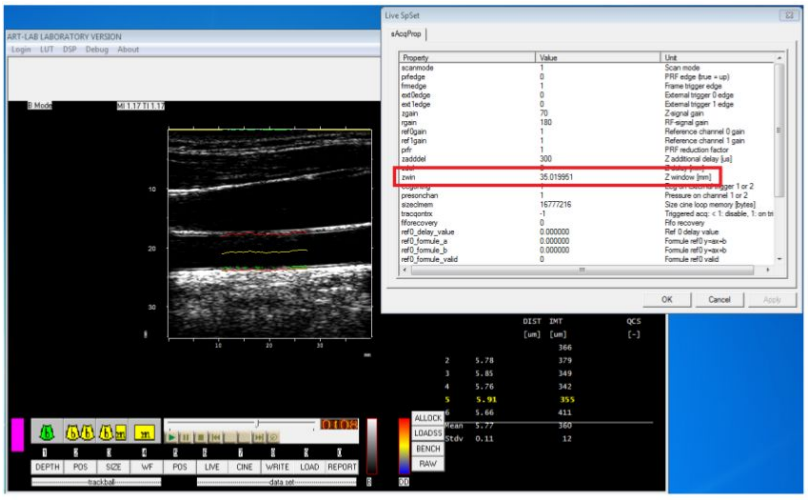


Fig 5:-Value of zwin captured using the ARTLAB software

4.2 Experimental Results

The network was tweaked for various hyperparameters like , learning rate , number of epochs and pixel thresholds. However , we now keep the pixel threshold as constant.

Training No	Learning rate	Epochs	Optimizer	pixel Threshold	Accuracy (in %)
1	$10e^{-8}$	5	Adam	22	55
2	$10e^{-7}$	10	Adam	28	97.23
3 With Batch_norm	$10e^{-6}$	8	Adam	28	66
4 with Batch_norm	$10e^{-7}$	10	Adam	28	81

5 with batchNorm and dropout	$10e^{-7}$	11	Adam	28	79.5

Best results were observed, for configuration no 2 . Where the accuracy achieved is 97.2% which is a good result.

```
batch 1217 of 1221 cost: 252, accuracy: 96 percent
18859 19504
batch 1218 of 1221 cost: 384, accuracy: 96 percent
18875 19520
batch 1219 of 1221 cost: 281, accuracy: 96 percent
18891 19536
batch 1220 of 1221 cost: 379, accuracy: 96 percent
(virtual_platform) esaote@esaote-Vostro-230:~/object_detection_AB$
```

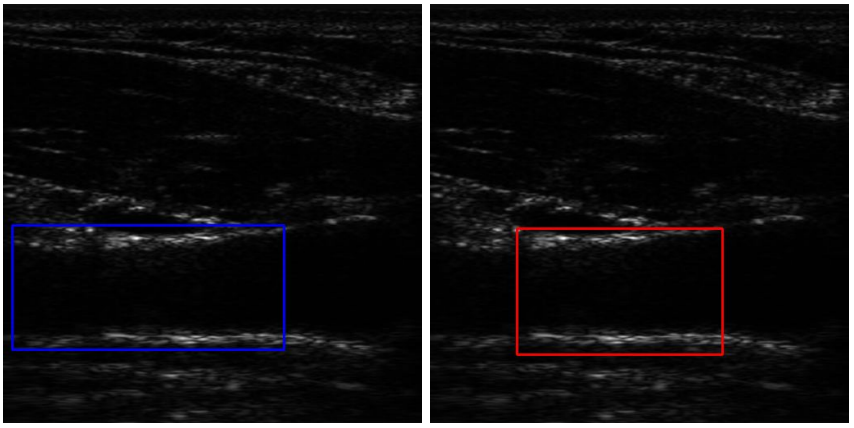
Fig 6:-An snippet of the results showing 18891 samples of 19536 being able to detect the blood vessel

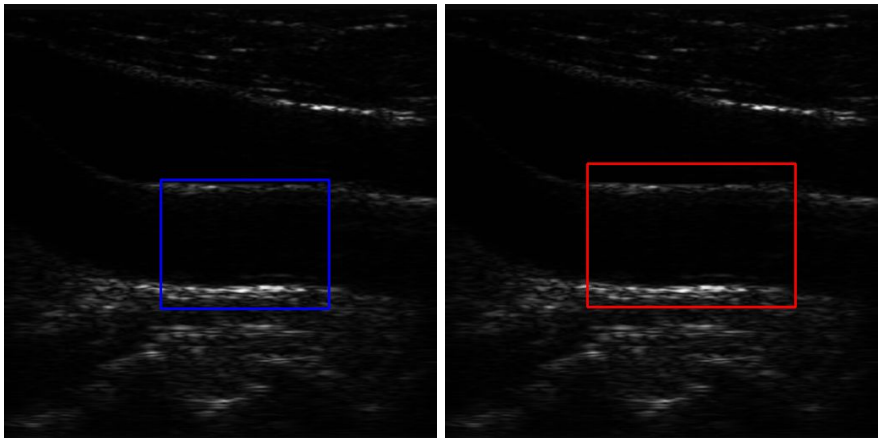
4.3 Final Results

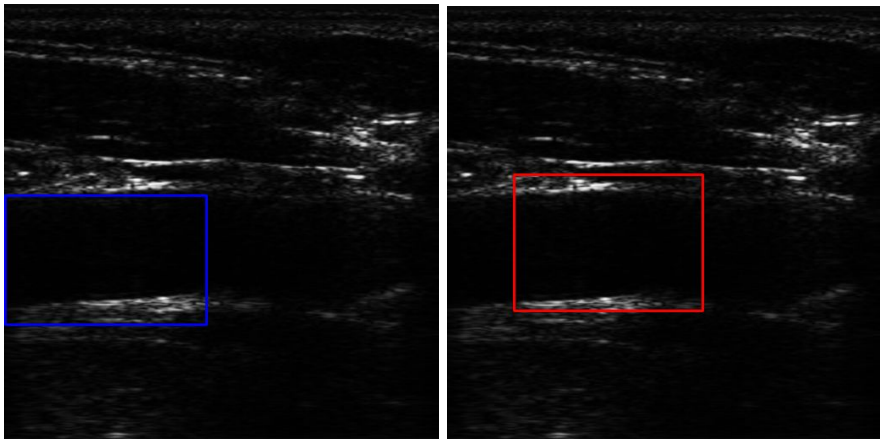
In the following set of images, the images on the left (bounding box =blue) are the ones that were placed by the doctor. The images on the right(bounding box =red) are the predictions of the neural network.

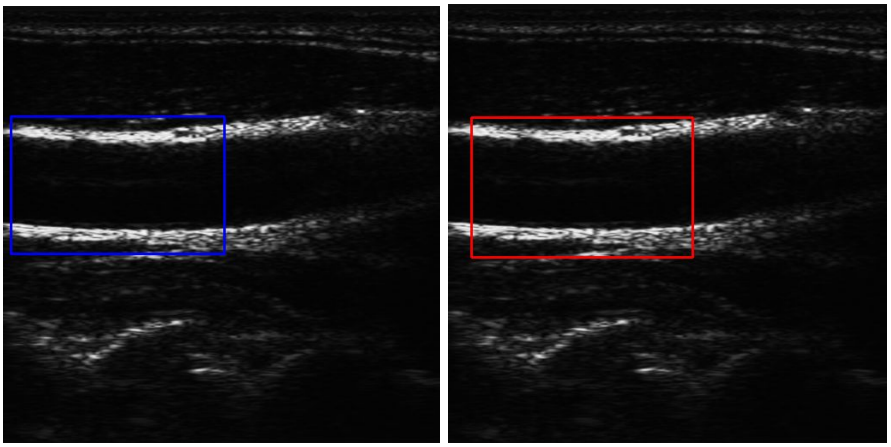
Good Results

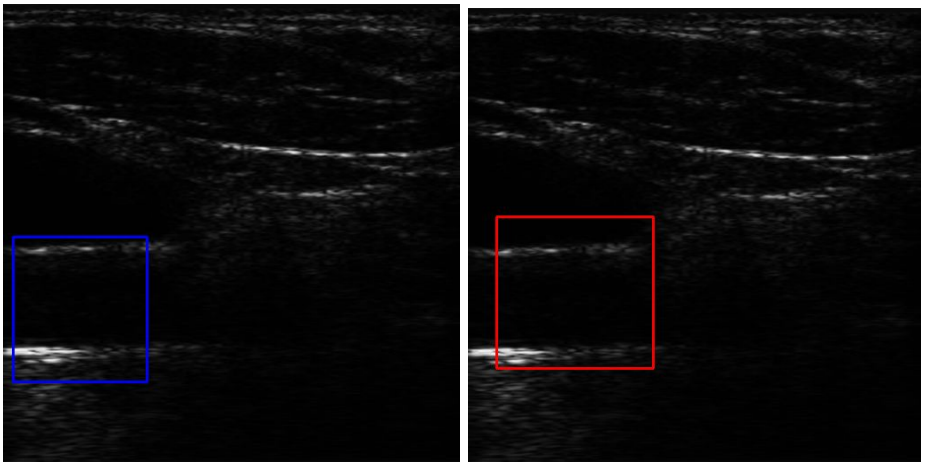
Ground Truth **Prediction**

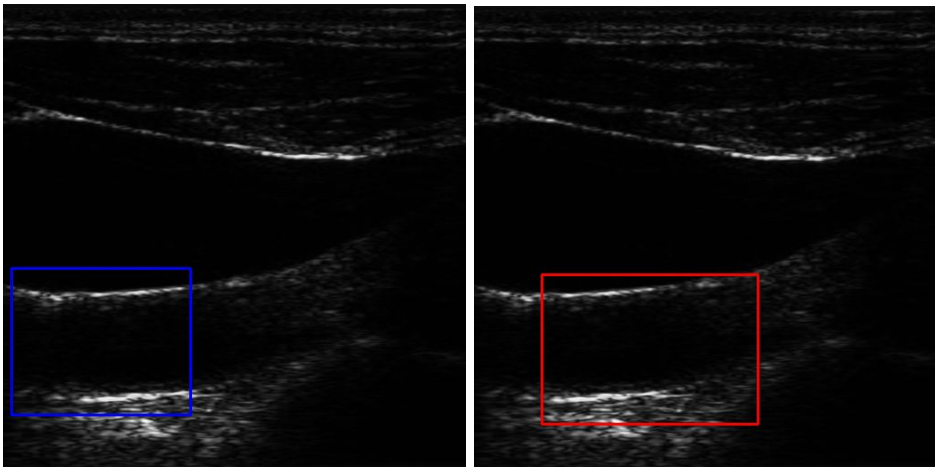


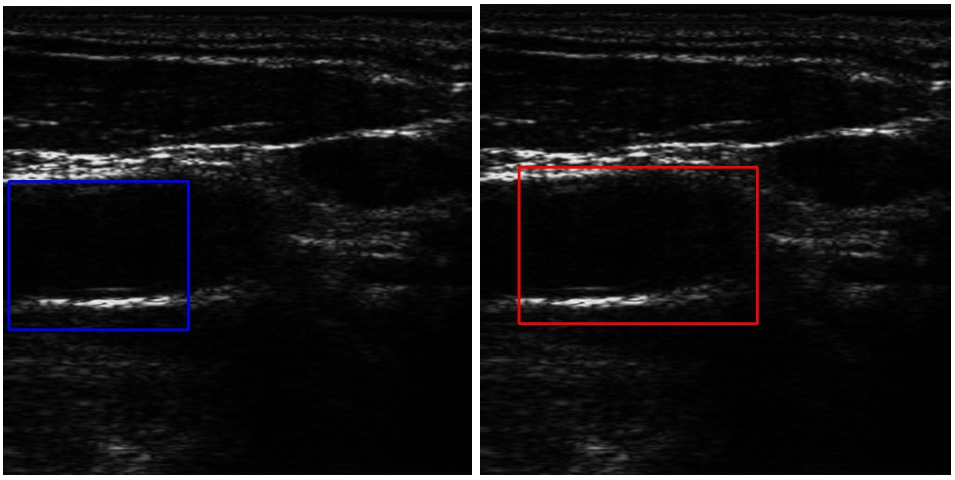






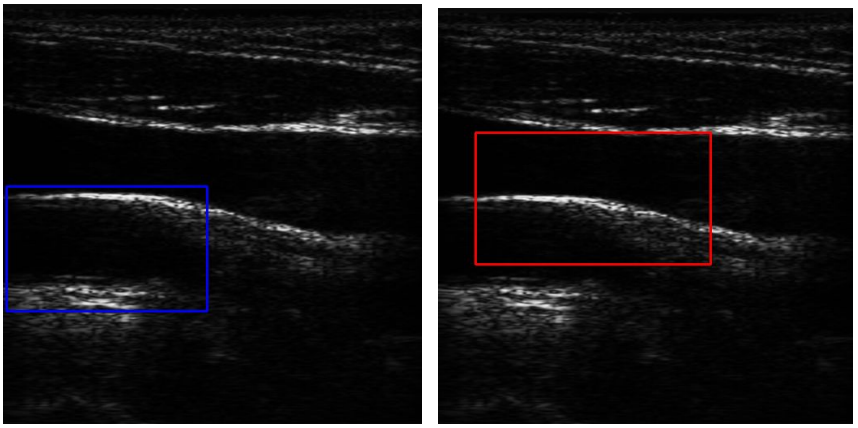


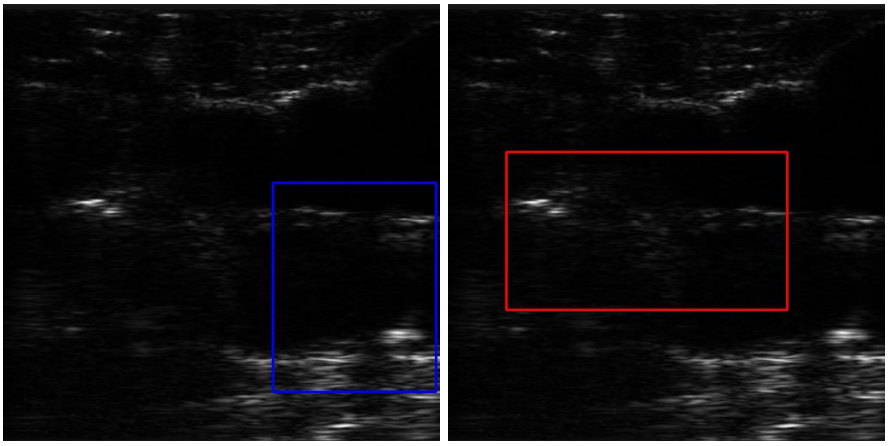




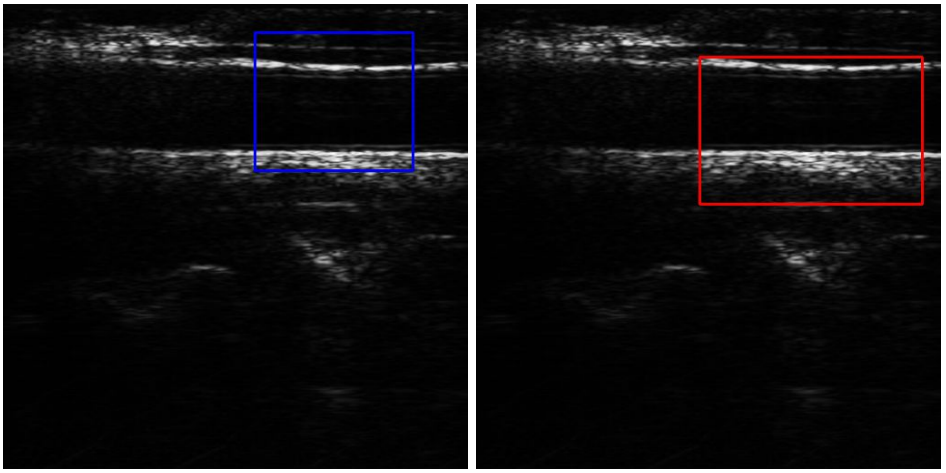
Here are a few tough examples, with potential vessel like structure where the model localizes the vessel correctly.

Following are some examples where the bounding box is not exactly where the vessel is but somewhat around it





Some cases the model gives really poor results , either due to potential vessel like structures with averages or pulls the boxes towards itself or cases where the scan is really bad , where it becomes difficult to localize the vessels even for professionals.



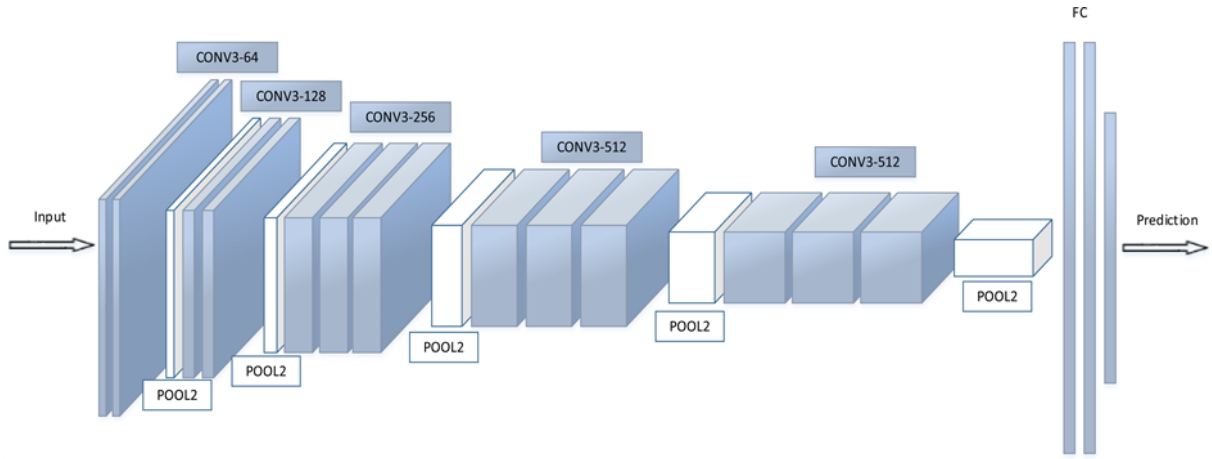
5 Discussion

A single caveat to this approach is that , the algorithm, will predict the most likely vessel structure , even if there is no vessel in the image. A single extension to this problem would be the YOLO algorithm , where in addition to localizing the vessel, we have one more element in the output vector which give the probability of a vessel being present. It should draw a box only if the probability is above a certain threshold eg >0.5 and can be trained by a simple softmax cross-entropy loss function or a cross entropy loss function. However since all the training examples present in the study had blood vessels. It is then important to convert our image into grids of 7x7 or 11x11 and for each of the grids we will now predict if the blood vessel is present in that grid or not. So the output vector will be approximately of the size 7x7x5 or 11x11x5 depending on the grid size we chose. The first element of the 3rd dimension will signify the probability of a vessel and the next 4 elements will give us the bounding box coordinates, for multiple detection a non-maximum suppression algorithm should be applied to clean out less probable bounding boxes, just like the YOLO algorithm.

For this purpose, the target vector for each image will also be of size 7x7x5. However due to shortage of time and converting the entire process from DIGITS to pure python/tensorflow code where the problem can be more explicitly solved for the task at hand , left no time to develop this further. However it could surely be the direction future students could look into and should give better and robust results .

5 Appendix

Network topology



References

[1] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In Computer Vision– ECCV 2008, pages 2–15. Springer, 2008.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640, 2015.

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015

[4] Alex Krizhevsky , Ilya Sutskever , Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks

[5] http://mccormickml.com/2014/03/04/gradient-descent-derivation/

[6] Diederik P. Kingma, Jimmy Ba Adam: A Method for Stochastic Optimization

[7] Sergey Ioffe, Christian Szegedy Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

[8] Limbu YR1, Gurung G, Malla R, Rajbhandari R, Regmi SR.Assessment of carotid artery dimensions by ultrasound in non-smoker healthy adults of both sexes.