Applying Deep Learning to Medical Images
Ricardo Diniz, nº 73591, MEEC

Abstract - Deep convolutional networks have recently been embraced as a competitive solution for visual recognition tasks. Among these networks, the fully convolutional neural networks (FCNN), have been gaining traction, due to their advantageous characteristics. FCNN, using layer skipping, achieved excellent results when provided enough training samples and computational power. Their popularity surged because they drop the classic fully connected layer in favor of another set of convolutional layers. The U-Net architecture was inspired by the FCNN and built upon it, introducing the concept of symmetrical paths of compression and decompression, which allow for more precise localization and pattern detection. U-Nets are superior to their predecessors, outperforming them at a reduced memory and computational footprint.

While the FCNN and the U-Net were designed for two-dimensional inputs, a new contender (V-Net) targeted medical and biomedical third-dimensional images. The V-Net also introduces a new objective function, discards pooling layers in favor of more convolutional layers and uses residual propagation. V-Nets have achieved a good performance across all visual recognition tasks, being comparable to the state-of-the-art solutions while requiring a fraction of the processing time.

The primary objective of this thesis is to implement several U-Net and V-Net variants, to attest to their good performance on visual segmentation tasks of medical data, and to assess how the objective function, the receptive fields, the residual propagation, activation functions and the optimization method impact the model’s performance. A secondary objective of this thesis is to bridge the gap between theoretical knowledge and practical implementations, by analyzing Google’s Tensorflow application programming interface (API) which was designed specifically for distributed computing in machine learning.

Index Terms— FCNN; U-Net; V-Net; Deep Learning; Image Segmentation; Tensorflow.

I. INTRODUCTION

Convolutional Neural Networks have existed for a long time [8], however until recent years the available processing power, limited size of available datasets, hardware limitations such as memory storage and the shallowness of the proposed architectures worked against their good performance. In 2012 AlexNet [1] was introduced to the community and quickly became the first widespread deep convolutional net. It was based on a paper which focuses on gradient based learning [2] and won that year’s ImageNet Large Scale Visual Recognition Competition (ILSVRC). From then onwards the focus on CNNs brought many and great advancements to their architectures by tweaking activation and objective functions, adding more layers or reformulating how these interact with each other, creating new layers, and other innovations [3], [4] and [5].

The FCNN architecture [6] has proven itself to be competitive for image segmentation at a relatively low memory and computational power footprint when compared to alternatives. In the medical and biomedical domains, the V-Net [7] and its predecessor the U-Net [8], both types of FCNNs, have been widely accepted as cutting edge solutions to several visual recognition tasks due to their ability to achieve good results from small datasets and in a timely manner. By expanding on them and understanding the improvements one offers over the other, as well as how to implement them in modern API we can attest to their worth and, simultaneously, consider possible enhancements.

II. OVERVIEW

Convolutional neural networks are typically used to tackle classification tasks. In image processing for visual recognition tasks, particularly in the medical and biomedical field, every single pixel or voxel needs to be classified [9]. In this section a brief introduction to a deep learning pipeline focused on tackling such tasks is provided, followed by a comprehensive approach to two convolutional neural network model architectures, the U-Net and the V-Net, including the specifics of their implementation along with their respective author’s insights. Both architectures are based on the FCCN [6].

A. Deep Learning Pipeline

Preprocessing data is usually the first step of the pipeline of a neural network. It entails the loading, reading or otherwise mapping, as well as eventual sorting and potential augmentation of the original data. The work developed only contemplates some of these techniques, namely, cropping, artificial sample enlargement by mirroring the original, border erosion between adjacent objects and irregular borders, normalization of pixel values and binary conversion applied to the label masks.

Training and evaluation cycles are typically the deep learning pipeline’s bottleneck, regardless of whether computations are distributed or non-distributed. It is at this stage that the model effectively learns how to tackle the task at hand by finding and modelling the patterns it is being taught. This can be achieved through a combination of operations stacked by layers, the initialization of the network’s weights and biases, regularization such as batch normalization or dropout, the intended objective function, the performance metrics in play and the number of epochs used. The inputs consist of tensors with pixel information in a matrix-like format, the corresponding channels and the labelled binary masks of the targets. The output of the model is a tensor that contains the probability distribution over each individual pixel of how likely it is to belong to the target class and is obtained via the use of a sigmoid function, which is related to the logistic function in the sense that the former is a specific form of the latter. The model’s output logits are therefore converted to a real number between zero and one. After establishing the network’s inputs and outputs the next step is to define the objective or loss function which the model will try to optimize and the process through which this optimization will be monitored and assessed. It is at
this stage that the architecture is chosen, either the U-Net or the V-Net.

Binary sigmoid cross-entropy loss penalizes any incorrect prediction of a pixel value the same way, for example, given a pixel that belongs to the foreground, if the probability of that pixel belonging to the foreground is determined to be below 0.5 than the penalty will be the same as scoring it as 0 probability and, likewise, if that probability is above or equal to 0.5 no penalty will be factored. The dice-coefficient loss measures the similarity of each predicted mask with the corresponding ground-truth and heavily penalizes any diverging pixels, ideally until all probabilities are either 1 or 0 for each pixel in the predictions mask. In practice the use of the dice coefficient for two-dimensional inputs may be ineffective, possibly due to a much slower convergence to a minimum while, on the other hand, binary cross-entropy loss on its own tends to converge faster but gets stuck on the same solutions, local minima, after a number of steps.

Regarding optimizers, the stochastic gradient descent (SGD) adaptive moment estimation optimizer, Adam [12], is known among the scientific community due to its performance. Adam was designed to enhance Adadelta [10] and other similar algorithms [11]. It does so by, like SGD with momentum [13], storing an exponentially decaying average of the past gradients and, like Adadelta, storing an exponentially decaying average of past squared gradients.

B. U-Net

The U-Net architecture was first introduced by Olaf Ronneberger, Philipp Fischer and Thomas Brox in 2015 [8] and achieved the top score on the International Symposium on Biomedical Imaging cell tracking challenge of that same year. The author’s objectives were twofold, retain the advantages of the FCNN [6] while reducing its computational strain and build a model that could quickly learn specific patterns from a limited two-dimensional dataset, ranging from less than one hundred labelled samples to less than a thousand labelled samples, even taking into account augmentation. The network’s architecture consists of a contracting path to capture context and a symmetric expanding path.

The U-Net shares the increased resolution of the output and localization properties of the FCNN and further enhances them through several changes to the original. Two of the major changes are the symmetrical expanding path that enables precise localization, and the large amount of feature channels that allow for context propagation to higher resolution layers.

The authors propose pre-computing each ground-truth weights’ map to facilitate class balancing and the distinction between bordering objects, in their case cells. Each weight map is computed as:

\[ w(x) = w_c(x) + w_o, \exp \left( -\frac{(d_1(x) + d_2(x))^2}{2\sigma^2} \right) \] (1)

where \( w_c(x) \) is the weight map to balance class frequencies, \( d_1 \) the distance to the border of the nearest cell and \( d_2 \) the distance to the border of the second nearest cell. Through empirical testing the authors reached the values \( w_c = 10 \) and \( \sigma \approx 5 \) pixels as the optimal values. To initialize the network’s weights in such a way as to ensure that the feature maps have almost unit variance, the authors use He initialization [15].

According to the authors, data augmentation using elastic deformations was one of the key factors that accounted for the achievement of a well-performant algorithm. This statement is widely accepted in the deep learning community since it is corroborated by empirical data.

The implementation of this network was made in Caffe API so there is no direct translation to the Tensorflow API, but the corresponding functions are very accessible, particularly due to the Tensorflow library entries in Tensorflow.nn, which contains the layers used in this work, such as the two-dimensional convolutional layer, transposed convolutional layer, also known as deconvolutional layer with the deconvolution operation at its core and many other layers.

C. V-Net

The V-Net was first introduced to the academic community in 2016 by Fausto Milletari, Nassir Navab and Seyed-Ahmad Ahmadi [7] and achieved remarkable results for the Medical Image Computing and Computer Assisted Intervention’s PROMISE 2012 challenge. Its authors intended to leverage the CNN’s ability to, autonomously, learn a hierarchical representation of input data and use it to process third-dimensional images in order to segment prostate MRI volumes.

The existing drawbacks of potential alternative solutions, such as lack of context learning, high computational effort requirements and the known failures to successfully perform segmentation on ultrasound volumetric images led the authors to turn to the FCNN which at that point in time had achieved great results with a low memory and computational effort footprints but only for two-dimensional input images which implies slicing volumetric images towards that end when required.

Since it was inspired by the U-Net, the V-Net shares many structural similarities with the former such as the symmetrical contracting and expanding paths, henceforth named the compression and decomposition paths to follow the author’s terminology which originates in the signal processing domain. Also shared is the skipping property where the compression path outputs, at each stage and before down sampling, are propagated to the input of the corresponding decomposition path stage. This process is also known as feature forwarding.

The residual network entails learning a residual function, first proposed in a paper on deep residual learning for image recognition [5]. The authors approach to learning this residual function is to take the input of a stage and add it to the output of that stage. This happens both for down sampling, on the
compression path and for up sampling on the decompression path.

The dice loss layer uses a formulation of the dice coefficient as a basis towards a loss function. This formulation is

\[
\text{Dcoef} = \frac{2 |X \cap Y|}{|X| + |Y|} \quad (2)
\]

which for sums running over \( N \) voxels of the predicted binary segmentation volume \( p_i \in P \) and the ground-truth binary volume \( g_i \in G \) can be rewritten as

\[
\text{Dcoef} = \frac{2 \sum_i p_ig_i}{\sum_i p_i^2 + \sum_i g_i^2} \quad (3)
\]

The authors, and many other researchers on the community such as Fidon, Li, Garcia-Peraza-Herrera, Ekanayake, Kitchen, Ourselin and Vercauteren [14], have empirically tested and verified the improvements brought by using this objective function, in the context of this sort of task, instead of a typical multinomial logistic loss with sample re-weighting. As an additional criterion in favor of this loss the need to attribute weights to foreground objects is eliminated.

To assess the performance of the network architectures two challenges were selected. The first challenge, nuclei segmentation challenge, entails detecting different cell’s nuclei in order to facilitate the introduction of new pharmaceutical drugs to the general public, its data samples are significantly different from each other whether in acquisition means or on the actual data content. The second challenge, ultrasound nerve segmentation challenge, consists of segmenting specific nerve structures in the neck area, the samples are slices of volumes obtained through ultrasounds and there are few slices with foreground objects for a noisy data generating source.

D. Nuclei Segmentation Challenge

The challenge uses a modified version of intersection over union as score metric, the average precision at different intersection over union, IoU, thresholds. Their definition of precision, at each threshold value \( t \), is non-canonical and given by

\[
\text{precision}(t) = \frac{TP(t)}{TP(t) + FP(t) + FN(t)} \quad (4)
\]

To score their metric a sweep over a range of IoU thresholds is made, and at each point an average precision value is calculated. To better reflect this score metric on the performance of the implemented models, the mean intersection over union is monitored. Several changes were made to the original architectures while preserving the core architecture itself. These changes entail the hyperparameters, regularization, optimizers, reception fields, spatial resolution and, for the V-Net, the reduction of one dimension for the inputs and outputs.

All the implemented models share a similar preprocessing stage and postprocessing stage. In the preprocessing stage 70% of the samples are assigned for training and 30% for evaluation, the ground truths are assembled by merging all annotated foreground objects, a morphological binary filter is then used to erode the borders of neighboring cells, Fig. 1, and augmentation is performed. The augmentations used, for spatially disproportional samples, are cropping and mirroring, such that model input samples have either 256x256 or 512x512 pixels. An example of both horizontal and vertical mirroring augmentation can be seen on Fig. 2.

![Fig. 1](image1.png) Original 256 pixels of width by 256 pixels of height image and its corresponding eroded ground-truth and morphological binary filter.

![Fig. 2](image2.png) Symmetrically mirrored augmentation on the sample of Figure 1 to expand it to 512 by 512 pixels.

On the postprocessing stage the probabilistic prediction mask outputted by the models is converted to a binary prediction mask based on those pixels whose probability of belonging to the foreground is equal to or exceeds 50%. Afterwards, the augmented samples predictions masks are merged by reversing the augmentation process and applying binary dilation to attempt to restore information erased by the morphological filter.

All models use the SGD variant Adam with a learning rate of \( 1 \times 10^{-4} \) as the optimizer. Each training sample is a step, and each step is processed with 5 epochs. Early stop is used at 3500 steps.

Either the max pooling operation or convolutions with stride 2 are used for down sampling and at that point the number of feature maps, ranging from 8 to 128, is doubled while the spatial resolution is halved. Up sampling or “deconvolution” operations are used to reverse that process. Finally, the outputs are all submitted to a 1x1 convolutional layer with stride 1
which outputs the logits, which are in turn processed by the sigmoid function to output a probability distribution over each pixel which is then used to define whether or not it belongs to the foreground based on an chosen threshold.

In total 8 models were used, the first two are U-Net variants, the third is a hybrid U-Net/V-Net variant and the remaining five are V-Net variants. A comprehensive layout of the different models can be seen on Tab. 1.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Loss</th>
<th>Activation</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>U3CLR</td>
<td>Cross-entropy</td>
<td>ReLu</td>
<td>Xavier</td>
</tr>
<tr>
<td>U5HLLR</td>
<td>Cross-entropy + Dice</td>
<td>Leaky-ReLu</td>
<td>He</td>
</tr>
<tr>
<td>HUV</td>
<td>Cross-entropy</td>
<td>ReLu</td>
<td>Xavier</td>
</tr>
<tr>
<td>VCR</td>
<td>Cross-entropy</td>
<td>ReLu</td>
<td>Xavier</td>
</tr>
<tr>
<td>VHLLR</td>
<td>Cross-entropy + Dice</td>
<td>Leaky-ReLu</td>
<td>Xavier</td>
</tr>
<tr>
<td>VCLR</td>
<td>Cross-entropy</td>
<td>Leaky-ReLu</td>
<td>Xavier</td>
</tr>
<tr>
<td>VHLLRHe</td>
<td>Cross-entropy + Dice</td>
<td>Leaky-ReLu</td>
<td>He</td>
</tr>
<tr>
<td>VDLRHe</td>
<td>Dice</td>
<td>Leaky-ReLu</td>
<td>He</td>
</tr>
</tbody>
</table>

The first U-Net variant, U3CLR, resembles the original [8] barring the zero padded convolutions and the use of the sigmoid cross-entropy loss function, while the second U-Net variant, U5HLLR has a broader receptive field, 5x5, leaky-ReLu activation functions and an hybrid objective function obtained by adding the dice loss function to the sigmoid cross-entropy loss. The hybrid U-Net/V-Net variant, HUV, is essentially a U-Net implementation with the standard max pooling layers replaced by convolutional layers with 2x2 pixel-wide receptive fields and stride 2 for their kernels, the convolution receptive field of U5HLLR and the remaining characteristics of U3CLR.

All V-Net variants have one spatial dimension less when compared with the original. The first V-Net variant, VCR, differs from the original the most as it uses a ReLu activation function after each convolutional layer, sigmoid cross-entropy loss as the objective function and Xavier initialization. The second V-Net variant, VCLR, is similar to VCR with leaky-ReLu activation functions. The fourth V-Net variant, VHLLRHe, resembles VHLLR but with leaky-ReLu activation and He initialization. The fifth and final V-Net variant, VDLRHe, is essentially VHLLRHe with the dice loss function as its sole objective function.

The binary prediction masks for each of these models, for two arbitrary input samples, can be seen on Fig. 3 and Fig. 4. Furthermore, their performance metrics wise can be seen on Tab. 2 and Tab. 3.

Fig. 3. An input sample ground-truth mask side-by-side with the binary prediction masks outputted by (a) U3CLR, (b) U5HLLR, (c) HUV, (d) VCR, (e) VHLLR, (f) VCLR, (g) VHLLRHe and (h) VDLRHe.

Fig. 4. An input sample ground-truth mask side-by-side with the aggregated binary prediction masks outputted by (a) U3CLR, (b) U5HLLR, (c) HUV, (d) VCR, (e) VHLLR, (f) VCLR, (g) VHLLRHe and (h) VDLRHe.
Tab. 2. Loss and mean IoU model performance.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Loss</th>
<th>Mean IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>U3CLR</td>
<td>0.0745</td>
<td>0.8932</td>
</tr>
<tr>
<td>U5HLLR</td>
<td>0.2392</td>
<td>0.9054</td>
</tr>
<tr>
<td>HUV</td>
<td>0.0747</td>
<td>0.8864</td>
</tr>
<tr>
<td>VCLR</td>
<td>0.0753</td>
<td>0.8929</td>
</tr>
<tr>
<td>VHLLR</td>
<td>0.2084</td>
<td>0.9161</td>
</tr>
<tr>
<td>VHLLRHe</td>
<td>0.2114</td>
<td>0.9173</td>
</tr>
<tr>
<td>VDLRHe</td>
<td>0.2870</td>
<td>0.5255</td>
</tr>
</tbody>
</table>

Tab. 3. Accuracy, precision and recall model performance.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Accuracy [%]</th>
<th>Precision [%]</th>
<th>Recall [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>U3CLR</td>
<td>97.6663</td>
<td>93.5002</td>
<td>86.1195</td>
</tr>
<tr>
<td>U5HLLR</td>
<td>97.8816</td>
<td>91.0258</td>
<td>90.9301</td>
</tr>
<tr>
<td>HUV</td>
<td>97.5772</td>
<td>96.5799</td>
<td>82.2877</td>
</tr>
<tr>
<td>VCR</td>
<td>97.7037</td>
<td>95.9587</td>
<td>83.9880</td>
</tr>
<tr>
<td>VCLR</td>
<td>98.1818</td>
<td>94.6954</td>
<td>89.5369</td>
</tr>
<tr>
<td>VHLLR</td>
<td>98.7904</td>
<td>94.3029</td>
<td>88.0402</td>
</tr>
<tr>
<td>VHLLRHe</td>
<td>98.1960</td>
<td>94.0450</td>
<td>90.3635</td>
</tr>
<tr>
<td>VDLRHe</td>
<td>76.1173</td>
<td>32.4727</td>
<td>95.7107</td>
</tr>
</tbody>
</table>

From the results of Tab. 2 and Tab. 3 we conclude that V-Net architectures outperform U-Net ones under similar conditions. The use of a hybrid objective function also proved itself to be a deciding advantageous trait, allowing for better overall scores across all metrics. The VHLLRHe model achieved the best performance metrics wise as it gathers both characteristics. The evolution over training steps of the loss can be seen on Fig.5 and the mean intersection over union on Fig.6.

Fig. 5. Loss metric progression for the VHLLRHe model.

Fig. 6. Mean IoU metric progression for the VHLLRHe model.

E. Nerve Segmentation Challenge

The challenge uses the Sorensen-Dice coefficient loss as score metric so three of the models being tested use the dice coefficient as the sole objective function. The other two models, From_Prior and No_Prior (Tab. 4), port the architecture and configurations of the model with the best performance in the previous challenge, VHLLRHe.

The samples consist of slices of 49 volumetric ultrasounds of the neck region amounting to a total of 5635 samples. An example of a sample with the target nerve structure can be seen on Fig.7 and an example of a sample without it on Fig.8.

Regarding the models with an identical pipeline as that of VHLLRHe, No_Prior, trained on its own for 30000 steps while From_Prior began its training from the weights graph of VHLLRHe at 3500 steps and then trained up to 30000 steps. This experiment intended to show the impact of using weights from a model already capable of segmenting foreground objects versus a model which trains without this prior. The most relevant score metrics, at different steps can be seen on Tab.4.

For those models that use the dice coefficient as the sole objective function, which were built upon VDLRHe, due to the poor performance of their predecessor a study was made on the impact of changing the learning rate by an order of magnitude upwards and downwards. As such the models Dice_Lr1e3, Dice_Lr1e4 and Dice_Lr1e5 trained for 25000 steps for learning rates, respectively. The results can also be seen on Tab. 4.

Fig. 7. Ultrasound input sample and its denoted ground-truth mask.

Fig. 8. Ultrasound input sample without the target nerve structure.
Fig. 8. Ultrasound input sample and its empty ground-truth.

Tab. 4. Loss and mean IoU model performance.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Loss</th>
<th>Mean IoU</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>From_Prior</td>
<td>0.3024</td>
<td>0.4937</td>
<td>20423</td>
</tr>
<tr>
<td>No_Prior</td>
<td>0.2983</td>
<td>0.4954</td>
<td>20015</td>
</tr>
<tr>
<td>From_Prior</td>
<td>0.2432</td>
<td>0.6038</td>
<td>25321</td>
</tr>
<tr>
<td>No_Prior</td>
<td>0.2537</td>
<td>0.6105</td>
<td>25000</td>
</tr>
<tr>
<td>From_Prior</td>
<td>0.3084</td>
<td>0.4937</td>
<td>30000</td>
</tr>
<tr>
<td>No_Prior</td>
<td>0.2989</td>
<td>0.4940</td>
<td>30000</td>
</tr>
<tr>
<td>Dice_Lr13</td>
<td>0.1386</td>
<td>0.4937</td>
<td>25000</td>
</tr>
<tr>
<td>Dice_Lr14</td>
<td>0.1386</td>
<td>0.4937</td>
<td>25000</td>
</tr>
<tr>
<td>Dice_Lr15</td>
<td>0.9042</td>
<td>0.5825</td>
<td>25000</td>
</tr>
</tbody>
</table>

The results obtained by the models used are overall poor. From the results seen on Tab.4 we conclude that the prior introduced from porting the weights of VHLLRHe at 3500 training steps introduced too much regularization as it became detrimental to the overall performance of the model. We also see that the lower learning rate, $1 \times 10^{-5}$, led to a significant improvement of the learning process while an increase quickly led to a vanishing gradient scenario. Out of all these models, only No_Prior at 25000 steps, From_Prior at 25321 steps and DiceLr1e5 at 25000 steps actually produced a binary segmentation mask which can be seen on Fig.9(a), 9(b) and 9(c), respectively.

Fig. 9. An input sample ground-truth mask side-by-side with the binary prediction masks outputted by (a) No_Prior 25000 steps, (b) From_Prior 25321 steps and (c) DiceLr5 at 25000 steps.

It is interesting to note that the progression of the mean intersection over union metric for each Dice_Lr model is very concordant with the expected behavior. The mean IoU for Dice_Lr3, Fig.10, is constant throughout all evaluation steps which indicated that the learning process was severely hindered right at the early stages of training. As for Dice_Lr4, Fig.11, the model appears to be learning at a steady rate until it inevitably stagnates at a local minimum. Finally, Dice_Lr5, Fig.12, appears to be steadily progressing towards a more suitable minimum at a much slower, albeit stable rate.

Fig. 10. Error metric progression for the Dice_Lr3 model with 50000 steps.

Fig. 11. Error metric progression for the Dice_Lr4 model with 25000 steps.

Fig. 12. Error metric progression for the Dice_Lr5 model with 25000 steps.

III. CONCLUSIONS

Through the results presented in Tab.2, Tab.3 and Tab.4 it is possible to conclude that the V-Net based models outperform the U-Net based models when subjected to similar hyperparameters and architecture constraints.

Furthermore, dropping the pooling layer in favor of a down sampling convolutional layer drastically speeds up the convergence of the models and the residual function learning both speeds up the aforementioned convergence as well as slightly improves the performance score wise, which corroborates the findings of the authors of [7]. Besides all this, the latter also somewhat mitigates the effects of overfitting over subsequent training steps.

It is apparent that combining two different, differentiable, loss functions such as the binary cross-entropy loss and the dice coefficient loss can prove beneficial to a model. The reason lies with the objective improvement of the model’s performance in terms of metric scores under the same conditions of those that used only one loss function, presumably due to the more diverse penalties. Model’s that rely on both loss functions also
consistently scored a higher loss compared with the other ones which translates to a better assessment of the need to improve while avoiding vanishing or exploding gradient issues.

Regarding other design choices, the use of either He or Xavier initialization barely influences the scores of their respective models, although activation was used to introduce non-linearity after each convolution, while the original [7] said non-linearity is introduced only after down sampling and up sampling. Using either ReLu or leaky-ReLu activation functions did not result in a significant discrepancy between results and the binary erosion morphology filter, while effective at separating different foreground objects, proved itself to be excessive. The basic data augmentation performed on the pipeline used for all models already amounts to a great score improvement compared to early versions of the implementations under analysis. Finally, the learning rate used as standard, $1 \times 10^{-4}$, generally allowed for competitive performance metrics wise in cell nuclei segmentation, except for those models which rely solely on the soft-dice coefficient. As for the nerve segmentation challenge, the results overall were poor and a learning rate decrease of an order of magnitude led to significant improvements. On the other hand, an increase of an order of magnitude in the learning rate led to a similar result as the model converged to a local minimum and kept going back to it, which effectively disabled the learning process of the model entirely.

In future work a third-dimensional V-Net should be implemented and used with to tackle the second challenge by parsing the full ultrasound volumes from its slices and using them as samples. Versions of the U-Net and V-Net which strictly abide by the characteristics of the original implementations should also be tested. Improving the preprocessing stage by diversifying the augmentation techniques used, such as noise addition, rotation and dilation, processing samples without caching them and switching to a less abrasive morphological filter such as the watershed filter can also be significant sources of improvements. In terms of hyperparameters, other variants should explore alternatives for optimizers, instead of using just Adam increased network depths, increased numbers of convolutional layers on each stage and the use of more feature maps. Training and validation might benefit from cross-validation, the use of a broader range of learning rates, an improved early-stop criterion based on specific error metric scores and new data from a similar data generating source should be used.

Tensorflow’s API allowed for the successful implementation of the models presented throughout this work in a cohesive and comprehensive manner, possessing great tools for building, fully fledged, deep convolutional neural networks and their respective data processing pipelines. However, since training and validation were made locally and thus subject to physical memory constraints. To take full advantage of the distributed computing abilities provided by this API, for which it was designed, Google Cloud Machine Learning platform or a similar alternative should be used, allowing for larger input samples and faster training and evaluation steps as well as more complex underlying architectures.

IV. REFERENCES