Instruções Para os Autores

Carlos Alberto Ynoguti, Rosanna Mara Rocha Silveira, Phyllipe Lima
Instituto Nacional de Telecomunicações - Inatel
ynoguti@inatel.br, rosannas@inatel.br, phyllipe@inatel.br

Abstract—This document contains information on the preparation of the final version of a paper accepted for publication in the Scientific Congress of Inatel, INCITEL. Please carefully follow the instructions provided to ensure legibility and uniformity of accepted papers.

Index Terms—About four keywords or phrases in alphabetical order, separated by commas.

Resumo—Este documento contém informações para a preparação da versão final de um artigo aceito para publicação no Congresso de Iniciação Científica do Inatel, INCITEL. Por favor siga cuidadosamente as instruções para garantir a legibilidade e uniformidade dos artigos aceitos.

Palavras chave—Aproximadamente quatro palavras chave ou frases em ordem alfabética, separadas por vírgulas.

I. Introdução

O propósito deste documento é fornecer informações para ajudar os autores a produzir artigos com aparência profissional para o Congresso de Iniciação Científica do Inatel, INCITEL.

II. Instruções Gerais

Quando escrever o seu artigo, por favor atente às seguintes instruções:

A. Tamanho e formato do papel

Os trabalhos serão impressos em papel tamanho carta (letter), exatamente como você os submeter. Desta forma, a organização e o esmero são de extrema importância Por favor, faça uma revisão cuidadosa dos erros gramaticais e de digitação antes da submissão. Há um limite máximo de 6 e mínimo de 4 páginas para o artigo. Contamos com o bom senso dos autores neste caso.

Os artigos devem ser preparados em coluna dupla. Defina as margens superior e inferior em 1,78 cm, as margens esquerda e direita em 1,65 cm. As colunas devem ter largura de 8,89 cm e o espaço entre elas devem ser de 0,51 cm. Use espaçamento simples entre as linhas.

B. Resumo e abstract

Os artigos escritos em língua portuguesa devem ter também o resumo e as palavras-chave traduzidos para a língua inglesa, como neste exemplo. Garanta que tanto o abstract quanto o resumo tenham no máximo 150 palavras.

C. Seções e subseções

As seções devem ser numeradas com algarismos romanos e ter o título centralizado. Já as subseções devem ser numeradas com letras maiúsculas e ter o título justificado, caso haja sequência de subtítulos as letras devem ser minúsculas e justificadas.

c.1) Sub-subseção

Exemplo de uma sub-subseção.

D. Figuras e Tabelas

Figuras e Tabelas devem ser incluídas como parte do texto sempre que possível, caso contrário, agrupe-as ao final do texto. As Figuras não devem ter elementos coloridos e seus rótulos devem ser posicionados depois das mesmas, com alinhamento centralizado. A sua numeração deve ser feita com algarismos arábicos. Para as Tabelas, o procedimento é diferente: seus rótulos devem ser posicionados antes das mesmas, centralizados, e a numeração deve ser feita com algarismos romanos. As figuras devem ser referenciadas no texto, da seguinte forma: A Figura 1 apresenta o logo do INCITEL.

Fig. 1. Uma figura. O título deve ser colocado abaixo da mesma.

E. Equações

A numeração das equações deve ser entre parênteses e alinhada à direita, como no exemplo abaixo:

$$\phi_X(s) = E[e^{sx}] \tag{1}$$

Para mais símbolos matemáticos, consulte o LaTeX wiki [1]

F. Fontes

Use fonte do tipo Times New Roman ou similar. Os tamanhos a serem usados são mostrados na Tabela I

TABLE I
Tamanhos e Tipos de Letras

TEXTO	TAMANHO	ESTILO
Título	24pt	Negrito
Nome do autor	11pt	Normal
Afiliação	10pt	Normal
Texto principal	10pt	Normal
Título das seções	10pt	Caixa Alta
Título das subseções	10pt	Itálico
Título do resumo/abstract	9pt	Negrito,Itálico
Resumo/Abstract	9pt	Negrito
Título das figuras	8pt	Normal
Título das tabelas	8pt	Caixa Alta
Texto das tabelas	8pt	Normal
Referências	8pt	Normal

G. Referências bibliográficas

Liste as referências em ordem numérica ao final do artigo. Ao final deste texto tem-se vários exemplos de como listá-las, dependendo do tipo. Denote as citações dentro do texto através de colchetes (por exemplo [2]). Ao referenciar mais de um trabalho, use o mesmo par de colchetes, como exemplo: [2, 3, 4].

Segue um exemplo para citações textuais: “De acordo com Lima et al. [2]”

H. Outras questões

Não use notas de rodapé a menos que sejam estritamente necessárias; neste caso, procure não agrupá-las.

III. Methods

A. U-Net

a.1) Motivation

Here we explain why we choose U-Net as our first and baseline method to experiment with. Published in [5], it has been cited over 10000 times and widely used as a benchmark in medical image segementation. Based on U-Net, many vairations are designed to pursue better performance, as we will see in later parts. Nevertheless, U-Net itself was a big breakthrough at its publish time, and it is very beneficial to study with its netwrok structure.

a.2) Detailed Description

Image segementation has long been a major task in computer vision, where the input is an image and the expected output is the mask of image—where all pixels belonging to the same object are labeled out (See Figure 2).

Fig. 2. An illustration of the goal of iamge segementation.

The first try to combine deep learning methods with this task is Fully Convolutional Networks by [6], which became a milestone. We know that in traditional CNN, the feature dimension is decreased by convolution and pooling operations, and the reception region change from local to global gradually as the image propagate through the network. This makes it difficult to do segementation, as we have to restore the size of image. FCN uses reversed convolution and upsampling operation to restore image size, and conduct pixel-level classification.

However, FCN is not good at image details (see Figure 4). The result is often blurred and smooth, thus not suitable for medical image segementation, where we especailly care about edges and details in the image, in case the doctors make wrong diagnosis. So there came U-Net, which imporve based on FCN.

Figure 5 shows the structure of U-Net. It uses many feature channels to allow features containing more

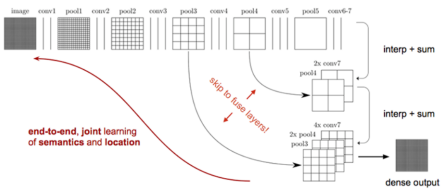


Fig. 3. Structure of Fully Convolutional Network, where it fuses pool4, pool3 and feature map to concate features.

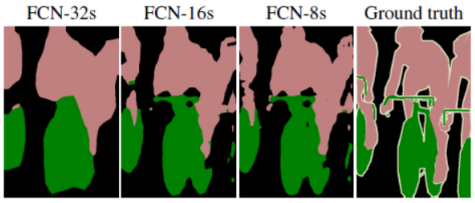


Fig. 4. Some results of FCN compared with the ground truth. Details of the cycling man isn’t very good.

information on the texture of original image to propagate between high-resolution layers. And it is specaily designed for medical tasks, where it is very difficult to separate the connected same-category cells. The authors proposed weighted loss function, where it gives the ground truth of connected cells more consideration. U-Net can perform well on small datasets like ISBI challenge, and does not require demanding GPU memory. Of course, data augmentation needs to be done before training.

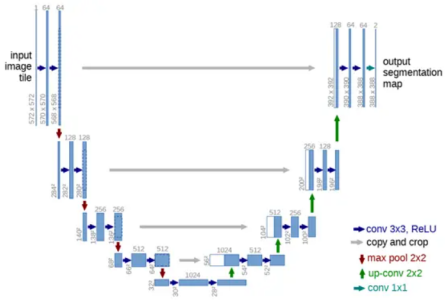


Fig. 5. Sructure of U-Net. It looks like the character ‘U’.

B. UNet++

b.1) Motivation

It’s clear that methods before UNet++[7] can solve general image segmentation problem with decent accuracy, especially when segmenting natural images. However, medical image segmentation require more accuracy in the detail of the segmentation, for example, tumors with ragged edges and more blood vessels on the edge are more likely to be malignant. This puts a higher target for neural networks dedicated to solving medical image segmentation problem, calling for higher-accuracy methods. Also, it’s hard for U-Net users to prune U-Net architecture to gain optimal inference time.

As a result, two propositions have been made:

- Better networks are needed to solve medical image segmentation problems;
- Networks need to be more friendly to production situations where recognition speed is vital.

In order to meet these requirements, UNet++ was designed above U-Net[5] with skip connections replaced with dense convolution blocks. We will discuss detailed designs in the next section.

b.2) Detailed Description

The biggest modification from U-Net into UNet++ was its skip connections. Where were direct skip connections in U-Net was replaced with an array of dense convolutional layers, reforming the network shape as shown in Figure.6.

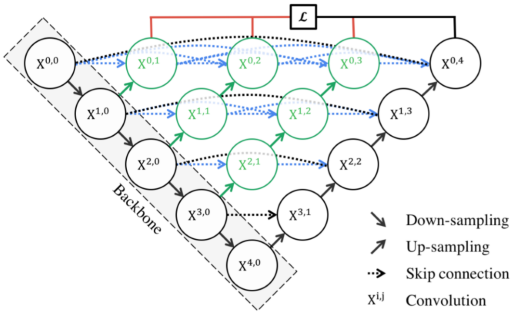


Fig. 6. Structure of Unet++, where black nodes represent basic U-Net, green nodes are dense convolution layers, blue and green arrows are dense connections and up-sampling, and red connections represent deep supervision.

The basic idea of UNet++ was that, with more dense convolution layers in between, the encoder activation domain would become closer to the decoder network activation domain, thus making finding exact segmentation a easier job. From another perspective, the network structure of UNet++ can be separated into several independent networks, as shown in Figure7, there was 4 of them, each having more layers than the last, inheriting the last one's encoder feature map, and adding another layer beyond that. Applying loss at all 4 output positions for the full-fledged network leverages the popular deep supervision method, not only helping the network to converge better, but also enabling users to prune the network during inference time as users can only use the first several layers of output to determine the result. Consequently, UNet++ with its special dense convolution connections could deal with the two requirements that we mentioned at the same time, thus being more strong and efficiency-aware than bare U-Net.

C. UNet+++

c.1) Motivation

After UNet and UNet++, we also try UNet+++[8], which was developed and published in 2020 and maybe the latest UNet-like deep learning network. In the following sections, we first briefly introduce UNet+++ according to the paper, and then introduce our training methods and results.

c.2) Network description

In image segmentation, Combining multi-scale features is one of important factors for accurate segmentation. In recent deep learning network like UNet and UNet++,

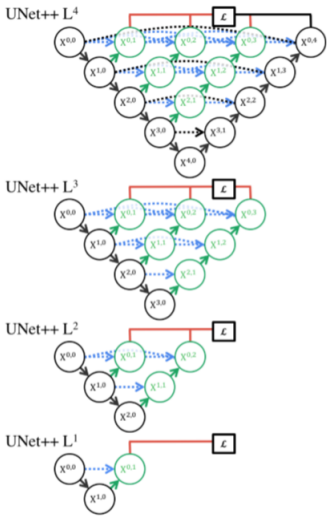


Fig. 7. Pruning structure of UNet++.

feature maps in different scale explore distinctive information. Lowlevel detailed feature maps capture rich spatial information, while high-level semantic feature maps embody position information. Nevertheless, these exquisite signals may be gradually diluted when progressively down- and up-sampling. However, by implementing full-scale skip connections, UNet+++ can make full use of the multi-scale features, incorporating low-level details with high-level semantics from feature maps in different scales. Figure./reffig:unetpppGraph shows simplified overviews of UNet+++. The image is from [8].

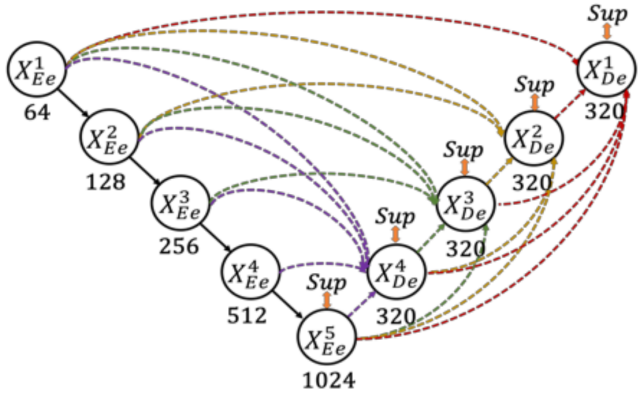


Fig. 8. UNet+++ network structure

IV. Algorithm

A. U-Net

There are mainly 3 tricks to train a U-Net, namely overlap-tile strategy, data augmentation and weighted loss.

a.1) Overlap-tile strategy

This strategy allows the seamless segmentation of arbitrarily large images. See Figure 9 To predict the pixels in the border region of the image, the missing context is extrapolated by mirroring the input image. This tiling strategy is important to apply the network to large images,

since otherwise the resolution would be limited by the GPU memory.

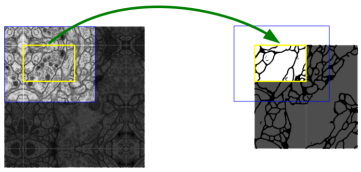


Fig. 9. Overlap-tile strategy for seamless segmentation of arbitrary large images. Prediction of the segmentation in the yellow area, requires image data within the blue area as input. Missing input data is extrapolated by mirroring

a.2) Data augmentation

As for our tasks there is very little training data available, we use excessive data augmentation by applying elastic deformations to the available training images. This allows the network to learn invariance to such deformations, without the need to see these transformations in the annotated image corpus. This is particularly important in biomedical segmentation, since deformation used to be the most common variation in tissue and realistic deformations can be simulated efficiently.

In case of microscopical images we primarily need shift and rotation invariance as well as robustness to deformations and gray value variations. Especially random elastic deformations of the training samples seem to be the key concept to train a segmentation network with very few annotated images. We generate smooth deformations using random displacement vectors on a coarse 3 by 3 grid. The displacements are sampled from a Gaussian distribution with 10 pixels standard deviation. Per-pixel displacements are then computed using bicubic interpolation. Drop-out layers at the end of the contracting path perform further implicit data augmentation.

a.3) Weighted loss

We pre-compute the weight map for each ground truth segmentation to compensate the different frequency of pixels from a certain class in the training data set, and to force the network to learn the small separation borders that we introduce between touching cells.

The separation border is computed using morphological operations. The weight map is then computed as

$$w(\mathbf{x}) = w_c(\mathbf{x}) + w_0 \cdot \exp\left(-\frac{(d_1(\mathbf{x}) - d_2(\mathbf{x}))^2}{2\sigma^2}\right)$$

where $w_c : \Omega \to R$ is the weight map to balance the class frequencies, $d_1 : \Omega \to R$ denotes the distance to the border of the nearest cell and $d_2 : \Omega \to R$ the distance to the border of the second nearest cell.

B. UNet++

There are several unique algorithms and tricks used in UNet++, including deep supervision and its unique loss function. We will discuss all these below in detail.

b.1) Deep supervision

We have mentioned deep supervision in the last part about the description of UNet++, here we will discuss it in detail.

Let's reflect on Figure.6, where all the outputs of the four branches connect to a single loss function. It's worth noticing that, in normal deep supervision procedure, losses at supervision layers decay with time, while in this setup, no loss would decay through time. This was because of the proposition that UNet++ was designed to support pruning during inference time, which can be done through only taking the output at layer $X^{0,t}, t \in \{1, 2, 3, 4\}$. As a result, the deep supervision in UNet++ is not in its standard form, whose effects will be discussed in the experiment section.

b.2) Loss Function of UNet++

In the paper[7], the loss function at arbitrary output point has a uniform formula, which consists of a cross-entropy term and a Dice-coefficient term:

$$L(Y, \hat{Y}) = -\frac{1}{N}\sum_{b=1}^{N}(\frac{1}{2} * Y_b * log\hat{Y}_b + \frac{2 * Y_b * \hat{Y}_b}{Y_b + \hat{Y}_b}) \qquad (2)$$

Where Y_b and $\hat{Y}_b$ denote the flattened probability vector output for the b^{th} image.

Totalling all these things together, we can see that UNet++ has 4 streams of constant gradient flow feeding back into the network during training time.

C. UNet+++

We use Pytorch to train UNet+++, so we use Pytorch's loss function and optimizer.

In details, because the result picture only have two class 0,1 per pixel, we use loss function BCEWithLogitsLoss, it combines a Sigmoid layer and the BCELoss. BCELoss is a criterion that measures the Binary Cross Entropy between the target and the output, the formula is:

$$\mathbf{l}(\mathbf{x}, \mathbf{y}) = \{l_1, \cdots, l_N\}^T,$$
$$l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

Where N is the batch size. BCEWithLogitsLoss is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.

We use Adam algorithm as optimizer. Actually we test Adam, RMSprop, SGD and we find that Adam performs best.

V. Experimental Settings

A. U-Net

Here we briefly descirbe the environment and tools we use to train a U-Net ourservles. We use TensorFlow with Keras API, and the training is done on a NVIDIA GTX 1060 (6GB). We use standard Adam optimizer with

learning rate 1e-4, and train the net for 10 epochs for 5 hours, each epoch containing 2000 steps. The data augmentation is done via Keras ImageGenerator API, with rotation, shift and horizontal flip.

B. UNet++

We will discuss the settings of UNet++ throughout our experiments in this section. The network structure was the same as depicted in Figure.6. Due to the limited capacity of our hardware, the training time batch size was very limited, which is actually only 1 picture per batch. We understand the effect that small batch size hinds convergence, but we had no choice. The optimizer was SGD, with a learning rate of 1e-3, momentum of 0.9 and weight decay of 1e-4. We were only able to train the network for 100 epochs due to the limited memory.

C. UNet+++

We use the given dataset to train and test the network, the learning rate is the default value(0.001) of torch.optim.Adam. Because the network is complex, we can only set batch size to 1, otherwise the GPU memory is not enough.

VI. Result

A. U-Net

To make sure we are on the right way, we just train for 5 epochs before we determine the optimal training and testing settings. We run into a few problems when we start. At first we find the original input size of U-Net is 256 square while the training data size is 512 square. Though the training is super fast and the validation accuracy seems pretty good (as the validation set is also from resized input), when we output the predicted image, use the provided code to test accuracy, the result can be as low as 60 percent. This is beacause that the provided code compare two images pixel-wise, and different dimension difinitely causes big problem.

We try different methods to fix it: (1)resize the predicted image to 512 square; (2) resize the label to 256 square; (3) change the network structure to allow input=512, and retrain the model. TableII shows the result.

TABLE II
3 ways to fix the size issue

Method	resize predtion	resize label	change network
Accuracy	0.6754	0.7545	0.6996

Seems working, but still not so good. And here comes new question: why does resizing label perform better? Theoretically, resizing leads to information loss. To answers these questions, we need to take a closer look at the data. The labels depict the edges of cell stuctures using pure black, and on any other parts of the images, the pixels are pure white. White make up most of the image, so even if you always predict Positive (i.e. no edge, white on image), you get around 60 accuracy. So after shrinking label, it is very possible that the area coverage of white pixels increase, such that the performance is even better than the retrained model.

But what about the 70 percent accuarcy? The validation information during training process says the accuracy is as high as 97 percent? The answer lies in the following two images, on the left is the predicted image and on the right the label, as is shown in Figure 10.

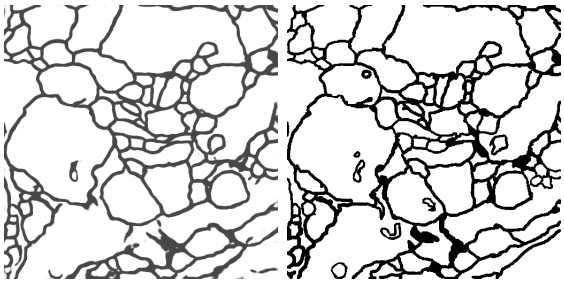


Fig. 10. Please look carefully at the two images. they are pretty the same, but wait, isn't the black in the left image not so black?

Hey! The network is not 100 percent sure about the prediction, so the output is possibility, which after converting to 0-255 uint8 value, is not 0 (pure black) nor 255 (pure white), but something in between. And the evaluation code requires that each pixel value to be the same as the label, to be count as a TP or TN. So if we want to use this code, we must conduct binarization on predictions before we compare pixels. Then we conduct experiment to select the optimal threshold of binarization, whose result in this case, is 87. See Figure 11 for details.

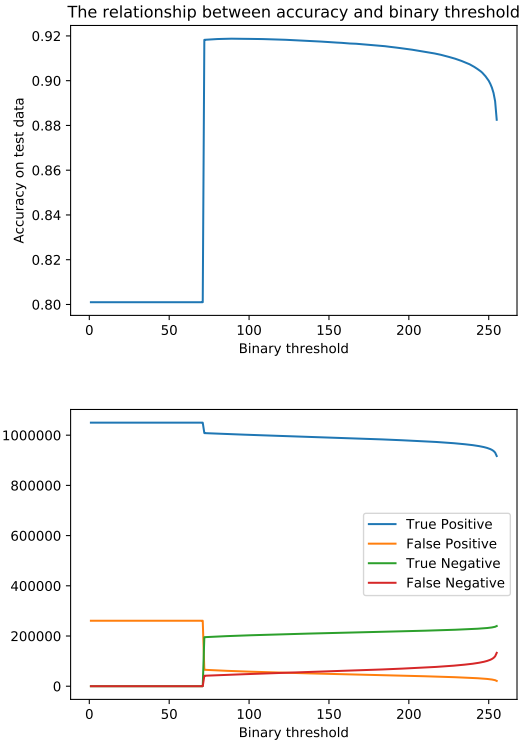


Fig. 11. Selecting optimal binarization threshold based on retrained model.

Then using this threshold, we trained for more 5 epochs and Figure 12 shows the performance and training epochs.

The best accuracy is 0.9189. It seems not improving much while the epochs increase, this may be that there is too many steps (2000) in an epoch, and the work is pretty much done in the first epoch.

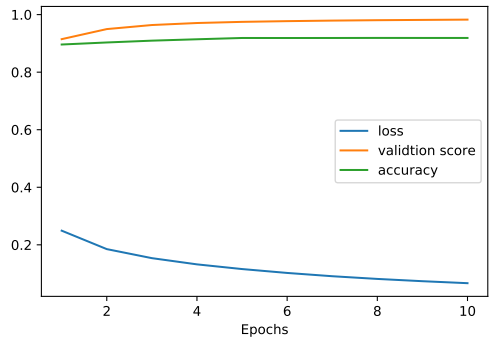


Fig. 12. Training history of U-Net.

B. UNet++

We trained UNet++ with the designated parameters, along with deep supervision and without supervision, which results in the accuracies in Table.III.

TABLE III
UNet++ results

	Out 1	Out 2	Out 3	Out 4
with deep supervision	0.878	0.886	0.891	0.894
without deep supervision	/	/	/	0.917

As shown in Table.III, deep supervised network performance increases with output layer, while still performing worse than the network without deep supervision, where in the UNet++ paper[7], the gap was not that big, and deep-supervised network perform generally better than normal ones. We suppose that was caused by our non-ideal training parameter setting. We tend to think that deep-supervised networks actually converge slower in this setting because of two reasons:

- Non-decaying deep supervision force the network to figure out a good approximation right at the beginning of the network, which may over-exploit the potential of the first layer, leaving less work to do for the following layers, and making the network less competent as a whole. We argue that this setup actually slows down the training process, which will be shown in the next experiment.
- Small training epochs lead to early exit and non-optimal performance during training.

As shown in Figure.13, training accuracy of normal UNet++ are generally higher than deep-supervised UNet++, which is consistent with our former proposition. We can observe that the deep-supervised network, trained upon our poor laptop GPU with few epochs, could reach no better result than normal UNet++. We can conclude that deep supervision really slows down the training process. Although it may perform better in the end, it may take a much longer training time, which is still a problem for those who want to save the training calculation time.

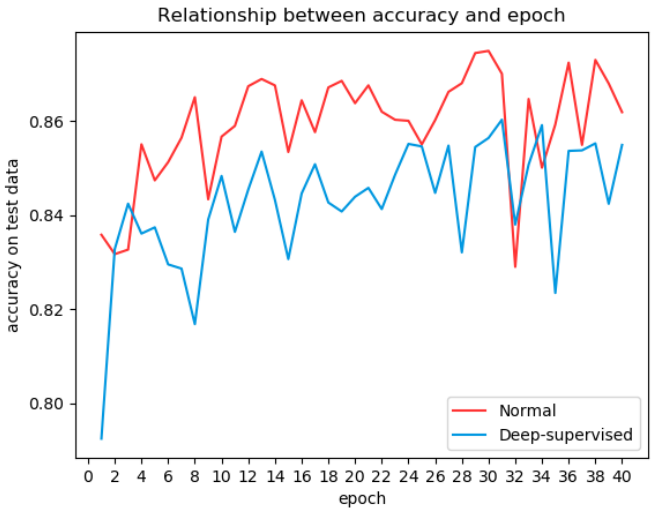


Fig. 13. UNet++ training accuracy-epoch curve.

C. UNet+++

We test the relationship between epoch and the accuracy of test data, in order to find the best epoch value. In the test, the value of epoch range from 1 to 40. In each iteration, we test the current model on test dataset, and record the accuracy. The result is given Figure.14.

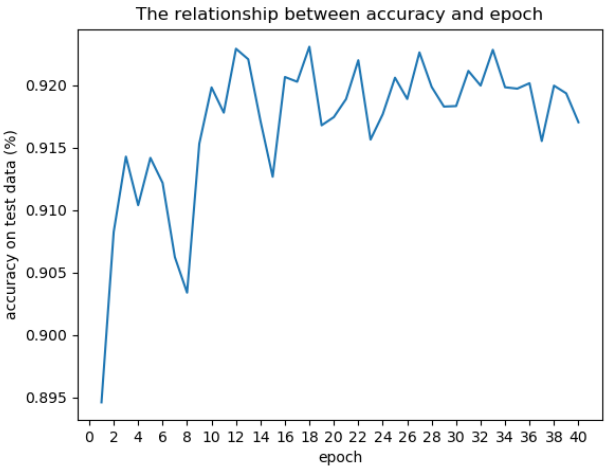


Fig. 14. UNet+++ training accuracy-epoch curve

The curve shows that as the epoch value increase, the accuracy first increase rapidly and than decrease slowly, which means that when epoch is too large, it becomes overfitting. The best accuracy is 0.923, with the epoch value 18

VII. Conclusion

In the project, we study the deep learning network UNet, UNet++ and UNet+++, and implement all three network on the given dataset. The experiments on the three network show lots of information. First, all experiments get relatively high accuracy on test data, which means that UNet and its variants work well in image segmentation. Second, these network can converge quickly. In the training

process, Although the given training dataset is not big(25 images), the accuracy reaches highest when iterate for about 20 times, a relatively small value. Third, the modification for Unet in other two networks improve the network performance, especially UNet+++. The best accuracy of UNet and UNet++ is approximately 0.918, and that of UNet+++ can be 0.923, a impovement of 0.5%.

In conclusion, we learn a lot from the project. We get a better understand of image segmentation and deep learning. However, Because of time limit and device limit, our experiments lack further optimization on some parameter, so the accuracy in some experiments above can be higher. We may refine it in the futrue.

References

[1] WikiBooks. LaTeX/Mathematics. url: en.wikibooks.org / wiki / LaTeX / Mathematics (visited on 07/01/2019).

[2] P. Lima, E. Guerra, P. Meirelles, L. Kanashiro, H. Silva, and F. Silveira. "A Metrics Suite for Code Annotation Assessment". In: Journal of Systems and Software 137 (2018), pp. 163–183. issn: 0164-1212. doi: https://doi.org/10.1016/j.jss.2017.11.024. url: http://www.sciencedirect.com/science/article/pii/S016412121730273X.

[3] J. Schell. The Art of Game Design: A Book of Lenses. 2nd. A. K. Peters, Ltd., 2014. isbn: 1466598646, 9781466598645.

[4] P. Lima, E. Guerra, M. Nardes, A. Mocci, G. Bavota, and M. Lanza. "An Annotation-based API for Supporting Runtime Code Annotation Reading". In: Proceedings of the 2Nd ACM SIGPLAN International Workshop on Meta-Programming Techniques and Reflection. Meta 2017. Vancouver, BC, Canada: ACM, 2017, pp. 6–14. isbn: 978-1-4503-5523-0. doi: 10.1145/3141517.3141856. url: http://doi.acm.org/10.1145/3141517.3141856.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: Cham: Springer International Publishing, 2015, pp. 234–241. isbn: 978-3-319-24574-4.

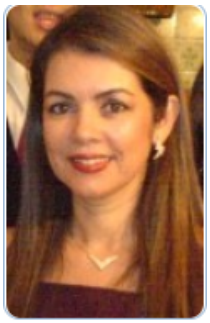
[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. IEEE Computer Society, 2015, pp. 3431–3440. doi: 10.1109/CVPR.2015.7298965. url: https://doi.org/10.1109/CVPR.2015.7298965.

[7] Zhou. Zongwei and Rahman. Siddiquee. Md. Mahfuzur. "UNet++: A Nested U-Net Architecture for Medical Image Segmentation". In: Cham: Springer International Publishing, 2018, pp. 3–11. isbn: 978-3-030-00889-5.

[8] Huimin Huang, Lanfen Lin, Ruofeng Tong, Hongjie Hu, Qiaowei Zhang, Yutaro Iwamoto, Xianhua Han, Yen-Wei Chen, and Jian Wu. "UNet 3+: A Full-Scale Connected UNet for Medical Image Segmentation". In: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE. 2020, pp. 1055–1059.

Autores

Carlos Alberto Ynoguti Possui graduação em Engenharia Elétrica pela Universidade de São Paulo(1991), mestrado em Engenharia Elétrica pela Universidade de São Paulo(1995), doutorado em Engenharia Elétrica pela Universidade Estadual de Campinas(2000) e pós-doutorado pela Universidade Estadual de Campinas(2001). Atualmente é Professor Adjunto do Instituto Nacional de Telecomunicações e Membro de corpo editorial da Telecomunicações (Santa Rita do Sapucaí) (1516-2338). Tem experiência na área de Engenharia Elétrica, com ênfase em Telecomunicações. Atuando principalmente nos seguintes temas:processamento digital de sinais, processamento de voz, reconhecimento de fala.



Rosanna Mara Rocha Silveira Possui graduação (1985) em Ciência da Computação pela UFMG (Universidade Federal de Minas Gerais) e mestrado (2003) em Engenharia Elétrica pela UNICAMP (Universidade Estadual de Campinas). Atualmente é professora adjunta da Fundação Instituto Nacional de Telecomunicações (INATEL). Na área de Engenharia tem atuação nas seguintes subáreas: algoritmos, estruturas de dados, banco de dados, modelagem e simulação, simulação de evento discreto, métodos numéricos, redes de comunicação sem fio, múltiplo acesso, análise e desempenho de redes de comunicação, processamento de sinais, transformadas wavelets e compressão de sinais.

Phyllipe Lima é Doutorando em Computação Aplicada pelo INPE - Instituto Nacional de Pesquisas Espaciais, na área de Engenharia de Software realizando estudos sobre metadados através da análise estática de código fonte e MSR (Mining Software Repositories). Mestre em Ciência da Computação(2016) pela UNIFEI - Universidade Federal de Itajubá. Engenheiro de Telecomunicações(2011) pelo INATEL - Instituto Nacional de Telecomunicações. Técnico em Telecomunicações(2006) pela Escola Técnica de Eletrônica - ETE "FMC". É professor auxiliar do INATEL, atuando nos cursos de Engenharia da Computação e Engenharia de Software.

Tem interesse nas áreas de Engenharia de Software Empírica, Desenvolvimento de Jogos e Computação Gráfica.