Final Project of CS420: Model for the Electron Microscopy Image Segmentation.

BoWen Zhang
517021910797

Xuanrui Hong
517030910227

Shuyuan Fu
517030910124

***Abstract*—Multi-class image segmentation and labeling is one of the most challenging and actively studied problems in computer vision. The goal is to label every pixel in the image with one of several predetermined object categories, thus concurrently performing recognition and segmentation of multiple object classes. In this project, we use the ISBI 2012 EM Segmentation Dataset [1] for image semantic segmentation, and tried various methods such as FCN, Unet, Unet++, CRF, etc. to achieve optimal results. The codes of our experiment can be found in github [1].**

1. Main Ideas

In this project, we implemente several methods for Pixel-level image segmentation, including Fully Convolutional Networks (FCN), Unet and Unet++. For each method, we train the model and tune the hyper-parameters on the train set. We also perform data augmentation on Unet and Unet++ and take DenseCRF as the back end of Unet to achieve optimal results. Based on our experiment, we find that among all methods, FCN performs the worst while Unet performs the best, and DenseCRF will slightly improve the performance. Our codes have been published into github: https://github.com/lxyi2u/CS420-SJTU-Project.

The remainder of this paper starts with the illustration of the image semantic segmentation methods and optimization algorithms of SGD and Adam during training, followed by the statement of the procedures and results of our experiment. We also conduct some analyses and comparisons based on the obtained results. At the end of this paper, we give the conclusion of our experiment.

2. Methods

2.1. Fully Convolutional Networks

Traditionally, the architecture of CNN is a stack of convolutional layers, followed by several fully connected layers, which project the feature map produced by convolutional layers into a fixed length feature vector. Therefore, CNN is suitable for image classification but hard to do semantic segmentation, which is a pixel level classification task. Different from CNN, FCN can accept the images with any size. It replaces the fully connected layers in CNN with 1*1 convolutional layers as Figure 1 shows, and adds a transpose convolutional layer after the last convolutional layer, to recover the size of the original image, in order to do pixel level image classification.

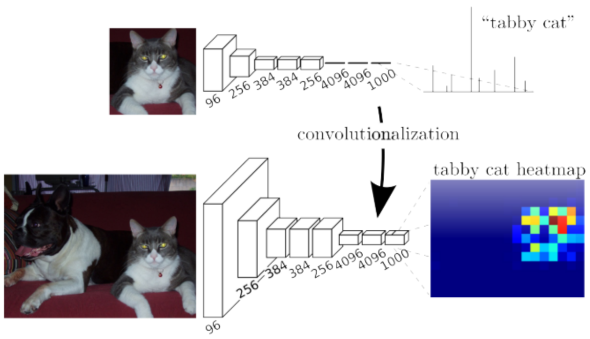


Figure 1: Convert fully connected layer to convolutional layer

Specifically, as Figure 2 shows, an image with size $H*W$ is feed into FCN, and goes through the convolutional layers and pooling layers. Finally, the resolution of the image becomes $\frac{1}{32}$ of the original image while the feature map contains more abstract and higher-order information. Then, using transpose deconvolutional layer to upsample the image to the original size.

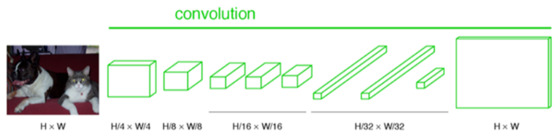


Figure 2: The architecture of FCN

As we mentioned before, FCN will upsample the feature map to enlarge the resolution 32 times. However, since the low resolution feature map loses many detailed information of the original image, it may be not capable of recovering the original image accurately. Therefore, FCN uses skip architecture as Figure 3 shows to solve this problem. It firstly upsamples the $\frac{1}{32}$ feature map 2 times, and adds it to the $\frac{1}{16}$ feature map to form a new $\frac{1}{16}$ feature map, then upsamples the new feature map 2 times and adds it to the $\frac{1}{8}$ feature map to form a new $\frac{1}{8}$ feature map. Finally, it upsamples the new

1. https://github.com/lxyi2u/CS420-SJTU-Project

$\frac{1}{8}$ feature map 8 times to recover the original image. The skip architecture of FCN fully uses the different resolution of feature map and can recover the image more accurately.

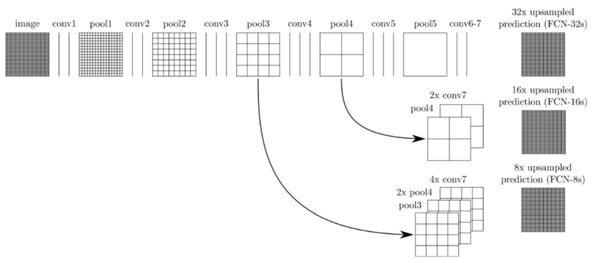


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

Figure 3: The skip architecture

2.2. Unet

The typical usage of convolutional networks is for classification tasks. The output of a classification task to an image is a single category label. However, in many vision tasks, especially in biomedical image processing, the desired output should include localization, that is, each pixel should be assigned a category label. More importantly, in biomedical tasks, thousands of training images are often unavailable (labeling costs are too high). Ciresan et al. segment medical images by predicting the category of each pixel [2]. Although this network has achieved good results, a major drawback is that there is a trade-off between localization accuracy and the use of contextual information. Larger tiles require more maximum pool layers, reducing positioning accuracy while smaller tiles can only allow the network to see very little context.

Unet [3] takes into account the characteristics from multiple layers. This makes the simultaneous use of good positioning and context possible. The main structure of the network is shown in Figure 4.

1) Encoder: the left half is composed of two 3x3 convolutional layers (ReLU) + 2x2 max polling layers (stride=2), and the number of channels doubles after each downsampling;
2) Decoder: The right half consists of a 2x2 upsampling convolution layer (ReLU) + Concatenation + two 3x3 convolution layers (ReLU);
3) The last layer uses a 1x1 convolution to change the number of channels to the desired number of categories.

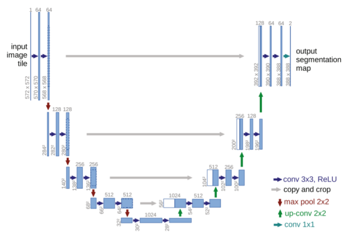


Figure 4: The architecture of Unet

2.3. Unet++

Unet++ [4] makes some improvements based on the original Unet, mainly aimed at the skip connection part of the original structure. The main architecture is shown in Figure 5. Compared with the original Unet network,

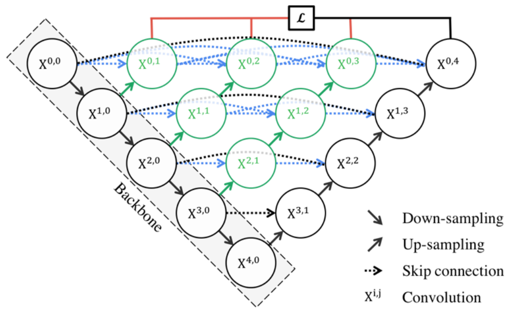


Figure 5: The architecture of Unet++

Unet++ connects all the Unet layers 1 to 4. One advantage of this structure is to let the network learn the importance of features of different depths, so that the network will not miss features that are effective at a certain depth. Another advantage is that it shares a feature extractor, that is, only one encoder is trained, and its different levels of features are restored by different decoder paths. This encoder can still be flexibly replaced with various backbones.

2.4. CRF and Dense CRF

Conditional Random Field (CRF) is a type of discriminative undirected probabilistic graphical model. Its nodes can be divided into exactly two disjoint sets X and Y, the observed and output variables, respectively; the conditional distribution $p(\boldsymbol{Y}|\boldsymbol{X})$ is then modeled.

Multi-class image segmentation and labeling problem can be posed as maximum a posteriori (MAP) inference in a CRF defined over pixels or image patches. The CRF potentials incorporate smoothness terms that maximize label agreement between similar pixels, and can integrate more elaborate terms that model contextual relationships between object classes.

Basic CRF models are composed of unary potentials on individual pixels or image patches and pairwise potentials on neighboring pixels or patches. However, its performance may be limited in its ability to model long-range connections. Some expanded CRF frameworks have also been created, aiming to incorporate hierarchical connectivity and higher-order potentials defined on image regions, but the accuracy will be restricted by unsupervised image segmentation. Fully connected CRF (also known as DenseCRF) model [5] that connects all pairs of individual pixels in the image can refine segmentation and labeling greatly. The details of this model are as follows:

Consider a random field X defined over a set of variables $\{X_1, \ldots, X_N\}$. The domain of each variable is a set of labels $\mathcal{L} = \{l_1, l_2, \ldots, l_k\}$. Consider also a random field I defined over variables $\{I_1, \ldots, I_N\}$. I ranges over possible input images of size N and $\mathbf{X}$ ranges over possible pixel-level image labelings. I_j is the color vector of pixel j and X_j is the label assigned to pixel j.

The Gibbs energy of fully connected pairwise CRF model is

$$E(\mathbf{x}) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j)$$

where i and j range from 1 to N.

The unary potential $\psi_u(x_i)$ is computed independently for each pixel by a classifier that produces a distribution over the label assignment x_i given image features. The pairwise potentials have the form

$$\psi_p(x_i, x_j) = \mu(x_i, x_j) \underbrace{\sum_{m=1}^{K} w^{(m)} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j)}_{k(\mathbf{f}_i, \mathbf{f}_j)}$$

where each $k^{(m)}$ is a Gaussian kernel $k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) = \exp\left(-\frac{1}{2}(\mathbf{f}_i - \mathbf{f}_j)^{\mathrm{T}} \Lambda^{(m)} (\mathbf{f}_i - \mathbf{f}_j)\right)$, the vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ are feature vectors for pixels i and j in an arbitrary feature space, $w^{(m)}$ are linear combination weights, and μ is a label compatibility function. Each kernel $k^{(m)}$ is characterized by a symmetric, positive-definite precision matrix $\Lambda^{(m)}$, which defines its shape.

For multi-class image segmentation and labeling, contrast-sensitive two-kernel potentials are used, defined in terms of the color vectors I_i and I_j and positions p_i and p_j:

$$k(\mathbf{f}_i, \mathbf{f}_j) = \underbrace{w^{(1)} \exp\left(-\frac{|p_i - p_j|^2}{2\theta_\alpha^2} - \frac{|I_i - I_j|^2}{2\theta_\beta^2}\right)}_{\text{apperance kernel}} + \underbrace{w^{(2)} \exp\left(-\frac{|p_i - p_j|^2}{2\theta_\gamma^2}\right)}_{\text{smoothness kernel}}$$

The appearance kernel is inspired by the observation that nearby pixels with similar color are likely to be in the same class. The degrees of nearness and similarity are controlled by parameters θ_α and θ_β. The smoothness kernel removes small isolated regions.

3. Algorithms

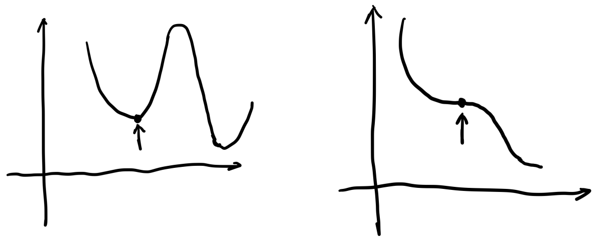
For deep learning model, we always have a loss function, which is our optimization goal. As the deep learning model is very complex, we cannot find the analytical solution for the optimization problem. Therefore, the most popular method for optimizing the deep learning model is gradient decent. However, as the dataset of deep learning task is always very large, calculating the gradient of all data can be computational expensive. Therefore, a general solution is to use the gradient of small batch to roughly represents the gradient of the whole dataset. In our experiment, we mainly use SGD with momentum and Adam algorithm as our optimizers.

3.1. SGD with momentum

SGD is designed to avoid the local minimum and saddle point shown in Figure 6a and Figure 6b, and deal with high curvature, small but direction consistent gradient. As the gradient in the local minimum and saddle point is 0, traditional SGD algorithm will be trapped in such points. However, the motivation of SGD with momentum is that it regards the parameter as a rock rolling down from the hill. When it achieves the local minimum or saddle points, although its accelerated speed is 0, it can rush out of the local minimum and saddle point with its inertia.

The formula of parameter update is shown in Equation 1. SGD with momentum algorithm use v as speed, thus the update steps of parameters are not only depend on the current gradients, but the history of gradients while α stands for the weight of history. Therefore, when there are many consecutive gradients point in the same direction, the step size will be continuously increased, which accelerates the speed of converging. The pseudo code of SGD with momentum algorithm is shown in Algorithm 1.

$$\begin{aligned} v_t &= \alpha v_{t-1} - \epsilon g_t \\ \Delta\theta &= v_t \\ \theta_{t+1} &= \theta t + \Delta\theta \end{aligned} \tag{1}$$



(a) Local Minimum (b) Saddle Point

Algorithm 1: SGD with momentum

```
Input: Learning Rate $\epsilon$, Momentum parameter $\alpha$,
       Initial parameters $\theta$, initial speed $v$
Output: Converged parameters $\theta$
1 while Stop criteria not met do
2     Sample a mini batch $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ ;
3     Calculate gradient:
        $g = \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$ ;
4     Update the speed $v = \alpha v - \epsilon g$ ;
5     Update the parameters: $\theta = \theta + v$ ;
```

3.2. Adam

Adam [6] uses momentum and adaptive learning rates to speed up convergence. The author of the Adam algorithm describes it as a set of advantages of two stochastic gradient descent extensions, namely:

1) The adaptive gradient algorithm (AdaGrad) reserves a learning rate for each parameter to improve performance on sparse gradients (ie, natural language and computer vision problems).
2) Root Mean Square Propagation (RMSProp) is based on the average value of the nearest magnitude of the weight gradient and adaptively retains the learning rate for each parameter. This means that the algorithm has excellent performance on unsteady and online problems.

Adam algorithm both estimates the first moment and the second moment of the gradients, which can help the model adapt to the learning rate automatically. The pseudo code of Adam algorithm is shown in Algorithm 2.

Algorithm 2: Adam

```
Input: Learning Rate $\epsilon$, Exponential decay rate $\beta_1$, $\beta_2$, constant value $\delta$ for numerical stability, initial parameters $\theta$
Output: Converged parameters $\theta$
1  Initialize the first moment and second moment:
     $s = 0$, $r = 0$;
2  Initialize the step: $t = 0$;
3  while Stop criteria not met do
4      Sample a mini batch $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ ;
5      Calculate gradient:
         $g = \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$ ;
6      $t = t + 1$ ;
7      Update the first moment estimation
         $s = \beta_1 s - (1 - \beta_1)g$ ;
8      Update the second moment estimation:
         $r = \beta_2 r + (1 - \beta_2) g \circledcirc g$ ;
9      Correct the first moment: $\hat{s} = \frac{s}{1-\beta_1^t}$;
10     Correct the second moment: $\hat{r} = \frac{r}{1-\beta_2^t}$;
11     Calculate update: $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}+\delta}}$;
12     Update the parameters: $\theta = \theta + \Delta\theta$;
```

4. Experiments and Results

In this section, we explain our experimental process and detail the methods we adopted. We also show the results of different methods and make some analyses and comparisons.

4.1. FCN

Followed by the default settings of the original paper, we choose pretrained VGG16 model without skip architecture as our base net, and train by SGD with momentum of 0.99, weight decay of 0.0005, batch size of 1, epoch of 200, learning rate of 10^{-10}, and the loss function is the cross entropy loss sum over all pixels.

In this default setting, although the final test accuracy achieves 80.10%, we find this result meaningless cause all pixels are predicted to 1. Therefore, a reasonable speculation is that the model is trapped in the local optimum, and we try to increase the learning rate. We find that the model gets the same results when learning rate is 10^{-9}, but we get loss of nan when learning rate is 10^{-8}. The loss of nan indicates that we have encountered the problem of gradient explosion, and we can not increase the learning rate any more. Therefore, we try a smaller learning rate such as 10^{-11}, 10^{-12}, as expected, it not works.

After further thinking, we assumed that the reason why our model predicts all pixels to 1 is due to the category imbalance. In our dataset, white region is much larger than the black region, which indicates the boundary. Therefore, we intend to start from solving the problems caused by unbalanced samples. And the focal loss is designed to solve this problem. The definition of focal loss is as follow:

$$L = -\alpha y^\gamma \log \hat{y} - (1-\alpha)(1-y)^\gamma \log(1-\hat{y})$$

Different from traditional binary cross entropy loss, focal loss add a γ term to focus on the sample that is easily misclassified. As Figure 7 shows, focal loss is equivalent to

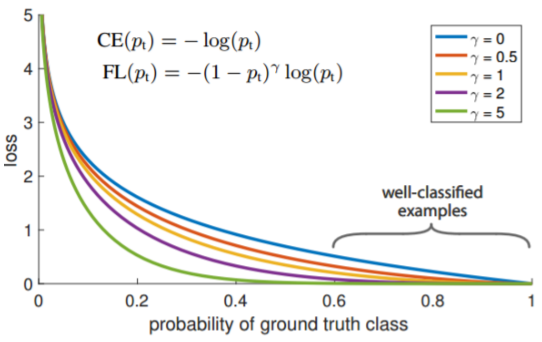


Figure 7: Focal Loss

cross entropy loss when $\gamma = 0$, and the larger γ is, the more model focuses on the misclassified samples. The α term is used to assign different weights for different classes, and a larger weight will be assigned to class with less samples to increase its importance. We firstly fix the $\alpha = 0.5$ and vary γ, the result is shown in Table 1. As our dataset has category imbalanced problem, we not only focus on the total accuracy, but consider the accuracy of each class at the same time, we consider white region as positive samples while black region as negative samples. As we can see, the

Table 1: γ - Test Set Accuracy

γ	**Total Acc**	**Pos Accuracy**	**Neg Acc**
0.5	19.91%	0.01%	99.99%
1	20.01%	0.26%	99.52%
2	33.76%	23.42%	75.39%

larger γ is, the more the model jumps out the local optimal. However, it's difficult to explain why negative accuracy is much higher than positive accuracy, which is contrast to the former experiment. And when we fix the $\gamma = 2$, the model will over inclined largely to positive samples if we choose an α larger than 0.5, and over inclined largely to negative samples. In a word, although focal loss makes the results more reasonable to a certain extend, it can't deal with our problem efficiently. Therefore, we need to do further optimization, and use focal loss with γ of 2, α of 0.5, instead of cross entropy loss in the following.

To a further step, we assume that the image in our dataset is too large, which might makes the task difficult. Therefore, we try to resize the image in our dataset from $512 * 512$ to $256 * 256$. The result is shown in Table 2. We can see

Table 2: γ - Test Set Accuracy

Size	Total Acc	Pos Acc	Neg Acc
512*512	33.76%	23.42%	75.39%
256*256	36.87%	13.87%	84.38%

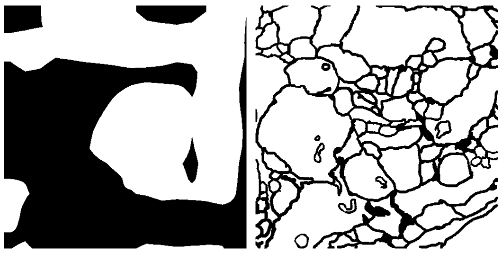
that the performance has a slight improvement, but doesn't fundamentally solve the problem.

Another direction that can be optimized is using a more powerful optimization algorithm like Adam, instead of SGD. As Adam algorithm can adaptively adjust the learning rate, it can deal with more complex problem. So we train the model by Adam with $\beta1$ of 0.9, $\beta2$ with 0.999, and use focal loss with α of 0.5, γ of 2, then tune the learning rate, and the result is shown in Table 3. We find that the model

Table 3: The result of Adam algorithm

Learning Rate	Total Acc	Pos Accuracy	Neg Accuracy
1e-5	27.89%	13.85%	**84.42%**
1e-6	**61.43%**	**68.36%**	33.53%
1e-7	49.86%	49.57%	51.02%

gets a relatively acceptable performance when learning rate is 10^{-6}. And then we visualize the result in the Figure 8. It can be easily find that our prediction is far from the label.



(a) Prediction (b) Label

Figure 8: Visualization of Prediction

Then, we try to apply the skip architecture to see if it can solve the problem, and we train the model **FCN16s** which combines feature maps of two different resolutions, and **FCN8s** which combines feature maps of three different resolutions. The result are shown in Table 4. Obviously, skip architecture can't improve the performance.

Table 4: Skip Architecture

Model	Total Acc	Pos Accuracy	Neg Accuracy
FCN32s	61.43%	68.36%	33.53%
FCN16S	53.90%	55.95%	45.63%
FCN8S	42.29%	39.04%	60.39%

In concluaion, we assume that FCN is not a good model for our task, as our task needs to find out the detailed information like boundary, and the downsampling process of FCN easily loses these information while the upsampling process of FCN is too rough to recover those information.

4.2. Unet

Followed by the default setting of the original paper, we train by Adam with $\beta1$ of 0.9, $\beta2$ of 0.999, batch size of 2, epoch of 1000, and the loss function is the binary cross entropy loss.

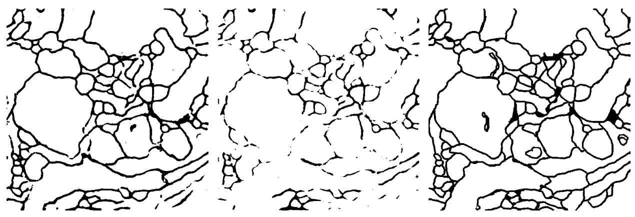
We tune the learning rate of 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, and the result is shown in Table 6. We can find that our model achieves best performance when learning rate is 10^{-5}, which is much better than the result in FCN. In additional, we find that the accuracy of the first test image is very low, and it is because that the first test image and its label doesn't match, and we only need to rotate the test image with 180° to match the image and the label. Then, we calculate the corrected test accuracy which is shown in Table 6.

Then, we visualize the result and find that when learning rate is 10^{-3} and 10^{-4}, the model predicts all the pixels are white, which is meaningless. However, the results of learning rate of 10^{-5} and 10^{-6} shown in Figure 9 predict the meaningful results. We assumed that when learning rate is too large, the model vibrated largely and thus couldn't find the optimal solution. When the learning rate is too small, the model converges slowly and will be easily trapped in the local minimum, and the boundary described by the model shown in Figure 9b is unclear. When the learning rate is 10^{-5}, the model describe the most of the boundary in the image, but still can't perfectly describe some detailed information.

Table 5: Unet: Accuracy - Learning Rate

Learning Rate	Train Acc	Test Acc	Test Acc (corrected)
1e-3	77.11%	80.10%	80.10%
1e-4	83.24%	80.10%	80.10%
1e-5	**96.52%**	**87.17%**	**91.21%**
1e-6	85.90%	84.75%	86.50%

To think further, we assume that our dataset is not large enough, which is an important reason that limits the performance of our model. Therefore, we use data augmentation to solve this problem. Specifically, we randomly rotate, shift, shear, scale the original image and label in training set. The rotate range is $[-10^\circ, 10^\circ]$, and the width shift range, height



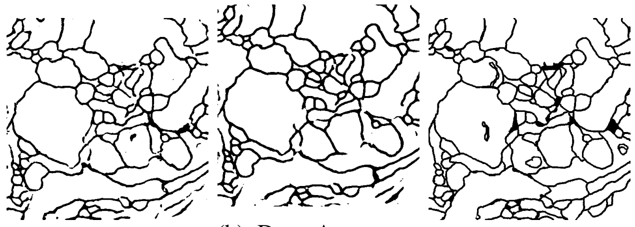
(a) LR=1e-5 (b) LR=1e-6 (c) Label

Figure 9: Visualization of Unet

shift range, shear range, and zoom range are $[-0.05, 0.05]$. Besides, we randomly horizontally flip the input. Then, we run the model with data augmentation and learning rate of 10^{-5}. The result is shown in Table 6 and Figure 10. We find that the performance after applying data augmentation only slightly improves, and there is only slight difference in visualization between original prediction and prediction after applying data augmentation. Therefore, we conclude that the limitation of the size of dataset is not the principle factor that affects the performance.

Table 6: Data Augmentation

Training Set	**Test Acc**
Original	91.21%
Data Augmentation	91.55%



(a) Original (b) Data Augmentation (c) Label

Figure 10: The result of data augmentation.

In conclusion, Unet performs pretty well in our task, and we assumed that it is because Unet fully uses feature maps with different resolution. Besides, the upsampling process is symmetrical to the downsampling process, which makes upsampling process more elaborated.

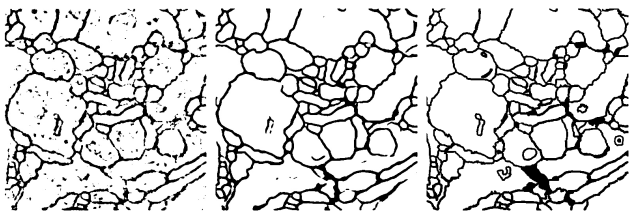
4.3. Unet++

In Unet++, we use SGD as the optimizer with batch size of 5, epoch of 100, learning rate of 10^{-3}, and the loss function is the binary cross entopy loss with Dice loss. We tune the learning rate of 10^{-3}, 10^{-2}, 10^{-1} and the result is shown in Table 7. We can find that our model achieve the best performance when learning rate is 0.1, which is much better than FCN but poorer than Unet. Then, we visualize the result as Figure 11c shows. It can be seen that the prediction structure has basically described the structure, but the details are still missing. Then in order to optimize the model, we conduct experiments with Adam as the optimizer, and the learning rate is 0.001. The results are shown in Table 8. It can be seen from the results that when the other parameters such as the learning rate are the same, Adam has better results than SGD. The reason may be that when Adam is used, the learning rate of each iteration has a clear range, which makes the parameter change very smoothly.

It is worth mentioning that we use data augmentation in the experiment, that is, some image transformation will be performed when loading the data set, which makes each epoch picture different. This makes up for the problem that the data set is too small in this experiment, increases the randomness of the data, and improves the generalization ability of the model.

Table 7: Learning Rate

Learning Rate	**Test Acc**
1e-3	82.46%
1e-2	84.42%
1e-1	**84.70%**



(a) LR=1e-3 (b) LR=1e-1 (c) Label

Figure 11: Visualization of Prediction

Table 8: SGD vs Adam

Optimizer	**Test Acc**
SGD	82.46%
Adam	**84.53%**

4.4. DenseCRF

DenseCRF can be used in the back end to optimize the output of the front end, and finally get the segmentation map. In our experiment, we apply the score map obtained by Unet as the front end and then perform DenseCRF. The method is implemented based on `pydensecrf` pacakage in Python. We first take the negative logarithm of the obtained score map to get the unary potential function. Then, using `d.addPairwiseGaussian` (add the color-independent term) and `d.addPairwiseBilateral` (add the color-dependent term) to add binary potential. After a series of experiments, we set $\theta_\alpha = 5$, $\theta_\beta = 0.01$, and $\theta_\gamma = 1$. The iteration times is set as 10. The segmentation map after DenseCRF is shown in Figure 12b. Comparing it with the standard label, we can get the accuracy 90.35% as shown in Table 9.

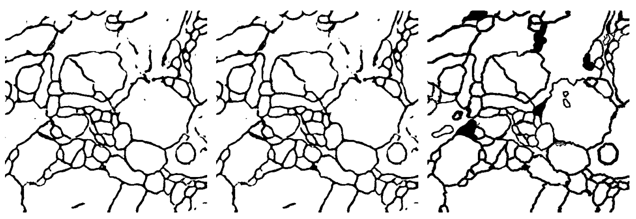

(a) Unet (b) Unet-CRF (c) Label

Figure 12: Unet vs Unet-CRF

Table 9: Unet vs Unet-DenseCRF

Model	**Test Acc**
Unet	90.30%
Unet-DenseCRF	90.35%

Table 9 shows that DenseCRF does not improve the Unet result pretty much. We think this may be because Unet has already performed pretty well, so DenseCRF has little effect. Besides, in order to adapt to the interface in `pydensecrf`, we converted the single-channel image to three channels, maybe using singe channel will get better results. However, since the model of Unet has already been optimized good enough to handle the segmentation problem, it is not much necessary to perform DenseCRF after Unet. We think that combining DenseCRF with FCN would be worth trying, and this is also what usually be taken in segmentation problem. However, due to the time limitation, we haven't implement this yet. We estimate that applying DenseCRF to FCN will improve the performance of FCN to a certain extent.

5. Conclusions

In this project, we try FCN, Unet, and Unet++ on the ISBI 2012 EM Segmentation Dataset to do image semantic segmentation. For each method, we train the model with different optimizer and tune the hyper-parameters to achieve the optimal performance. We also do data augmentation to solve the problem that the size of dataset is small. For Unet, we try to take DenseCRF as its back end to make the results more precisely. In conclusion, we find that FCN shows the worst accuracy, and Unet could perform pretty well. Neither data augmentation nor DenseCRF would greatly improve results.

Acknowledgments

This project is successfully completed under the guidance of Pro. Shikui Tu and TA Wenjing Huang, Peiying Li.

References

[1] Ignacio Arganda-Carreras, Srinivas C. Turaga, Daniel R. Berger, Dan Cirean, Alessandro Giusti, Luca M. Gambardella, Jürgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M. Buhmann, Ting Liu, Mojtaba Seyedhosseini, Tolga Tasdizen, Lee Kamentsky, Radim Burget, Vaclav Uher, Xiao Tan, Changming Sun, Tuan D. Pham, Erhan Bas, Mustafa G. Uzunbas, Albert Cardona, Johannes Schindelin, and H. Sebastian Seung. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9:142, 2015.

[2] Dan Ciresan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[4] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 3–11. Springer, 2018.

[5] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS11, page 109117, Red Hook, NY, USA, 2011. Curran Associates Inc.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.