Contents

1 Introduction **3**

2 Related work **3**
- 2.1 FCN . . . 3
- 2.2 DeepLab . . . 3
- 2.3 SegNet . . . 4
- 2.4 Mask R-CNN . . . 4
- 2.5 CapsNet . . . 4
- 2.6 U-Net . . . 5
- 2.7 Attention U-Net . . . 5
- 2.8 DenseNet . . . 6
- 2.9 Volumetric segmentation . . . 6
- 2.10 Model optimization-Weight pruning . . . 6
- 2.11 Loss functions . . . 7
  - 2.11.1 Cross entropy loss . . . 7
  - 2.11.2 Weighted cross-entropy . . . 7
  - 2.11.3 Focal loss . . . 7
  - 2.11.4 Dice loss . . . 8
  - 2.11.5 Generalised Dice loss . . . 8
  - 2.11.6 Jarccard loss . . . 8
  - 2.11.7 Hausdorff loss . . . 8
  - 2.11.8 Tversky loss . . . 9
- 2.12 Kappa coefficient . . . 9
  - 2.12.1 Definition . . . 9
  - 2.12.2 Kappa coefficient as loss function . . . 10
  - 2.12.3 Derivation . . . 10
- 2.13 Kappa loss and implementation . . . 10
  - 2.13.1 Definition . . . 10
  - 2.13.2 Derivation . . . 11
  - 2.13.3 Code . . . 11
  - 2.13.4 Results . . . 11

3 Method **14**
- 3.1 Dice loss with smooth item . . . 14
- 3.2 Jarccard loss with smooth item . . . 15
- 3.3 Image encoder . . . 15
- 3.4 Activation function . . . 15
  - 3.4.1 Sigmoid function . . . 16
  - 3.4.2 Softmax function . . . 16

4 Experiments **16**
4.1 Data input 16
4.2 Data augmentation 16
4.3 Hyper-parameters 17
4.4 Softmax and Sigmoid activation function 17
4.5 U-Net and DenseNet 18
4.6 Evaluation metrics 19
4.7 Results 19
4.7.1 Data 19
4.7.2 Data augmentation 19
4.7.3 Learning rate and loss functions 21

5 Conclusion **21**

Techinical report on seg CNN and loss

Jing Zhang

April 2019

1 Introduction

Deep Convolutional Neural Networks(DCNNs) and their variants have pushed the performance of computer vision systems to soaring heights on a broad array of high-level problems, including image classification and object detection and image segmentation, where DCNNs trained in an end-to-end manner have delivered strikingly better results than systems relying on hand-crafted features[4].

Good loss functions can make deep neural network converge better. In our work, we compared several different loss functions. Also, we compared two types of neural networks to see their performances.

2 Related work

2.1 FCN

Fully convolutional networks(FCNs)[16] don't have any of the fully-connected layers at the end, which are typically used for classification. Instead, FCNs use convolutional layers to classify each pixel in the image. FCNs use transposed convolution, or essentially backwards convolutions, to upsample the intermediate tensors so that they match the width and height of the original input image. This idea has become a common method in the field of image segmentation.

2.2 DeepLab

DeepLab[4] deals with image segmentation applying the 'atrous convolution'[11] with upsampled filters for dense feature extraction. And they further extend it to atrous spatial pyramid pooling[10], which encodes objects as well as image context at multiple scales. To produce semantically accurate predictions and detailed segmentation maps along object boundaries, they also combine ideas from deep convolutional neural networks and fully-connected conditional random fields[14].

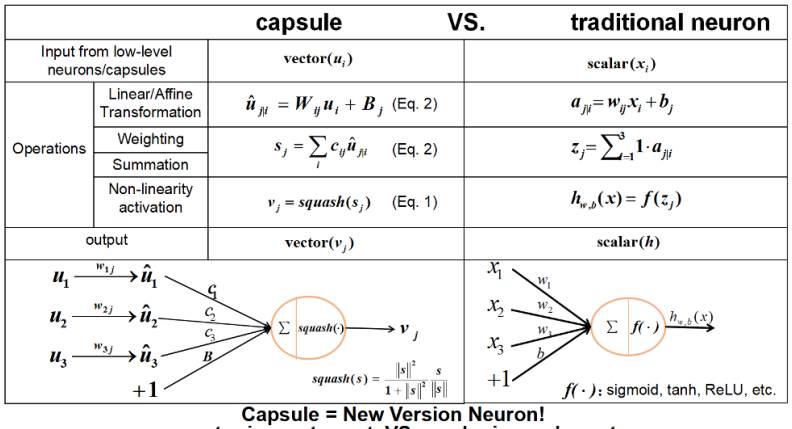


Figure 1: The differences between capsule and traditional neuron

2.3 SegNet

The architecture of SegNet[2] has encoder and decoder two parts. The encoder network in SegNet is topologically identical the convolutional layers in VGG16[25]. The decoder network which consists of a hierarchy of decoders one corresponding to each encoder. And the decoders use the max-pooling indices received from the corresponding encoder to perform non-linear upsampling of their input feature maps, which improves boundary delineation and reduces the number of parameters.

2.4 Mask R-CNN

Mask R-CNN[9] is developed from R-CNN[8], Fast R-CNN[7], Faster R-CNN[20]. Mask R-CNN extends Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest, in parallel with the existing branch for classification and bounding box regression.

2.5 CapsNet

Although neural networks based on CNN have make huge contribution in different areas. Hinton et al.point out the drawback of CNN. That internal data representation of a convolutional neural network does not take into account important spatial hierarchies between simple and complex objects. So they propose a new neuron called capsule which encapsulates all important information about the state of the feature they are detecting in vector form[23] .Capsules encode probability of detection of a feature as the length of their output vector. Figure 1 compares the differences between capsule and neuron[1].

[1]This figure is cited from Naturomics

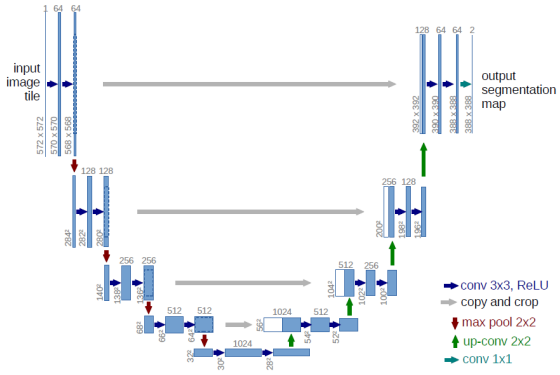


Figure 2: Architecture of U-Net[22]

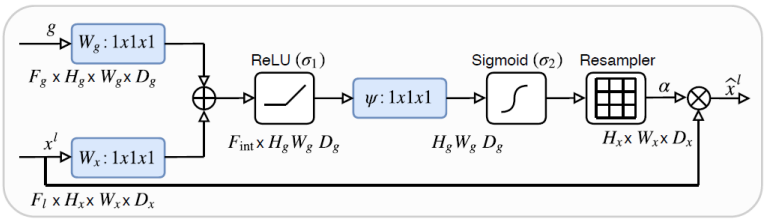


Figure 3: Schematic of additive attention gate[19]

2.6 U-Net

In biomedical image segmentation field, the most representative deep neural network is U-Net[22]. The architecture consists of a contracting path to capture context and symetric expanding path that enables precise localization.

2.7 Attention U-Net

Attention gate(AG) model is a new mechanism that automatically learns to focus on target structures of varying shapes and sizes. Models trained with AGs implicitly learn to suppress irrelevant regions in an input image while highlighting salient features useful for a specific task[19]. Additive attention is formulated as follows:

$$q_{att}^l = \psi^T(\sigma_1(W_x{}^T x_i^l + W_g{}^T g_i^l + b_g)) + b_\psi \tag{1}$$

$$\alpha_i^l = \sigma_2(q_{att}^l(x_i^l, gi; \Theta_{att})) \tag{2}$$

$$\hat{x}_{i,c}^l = x_{i,c}^l \cdot \alpha_i^l, \tag{3}$$

$\alpha_i \in [0, 1]$ is attention coefficient.

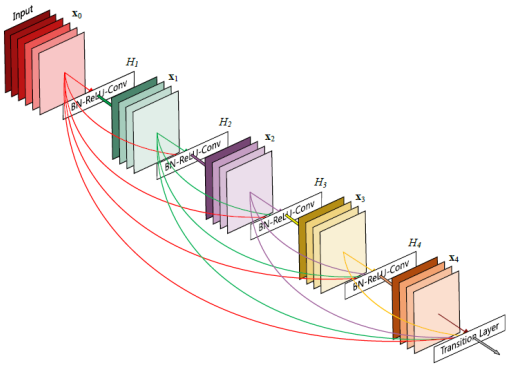


Figure 4: A dense block[12]

2.8 DenseNet

The Dense Convolutional Network(DenseNet)[12] connects each layer to every other layer in a feed-forward way. This deep neural network alleviate the vanishing-gradient problem, strengthen feature propagation, and substantially reduce the number of parameters. The code is from https://github.com/titu1994/Fully-Connected-DenseNets-Semantic-Segmentation

2.9 Volumetric segmentation

3D U-Net[5] and V-Net[17] are dedicated for 3D image segmentation based on a volumetric, fully convolutional neural network, which are both similar to U-Net[22].

2.10 Model optimization-Weight pruning

Weight pruning[2] means literally that: eliminating unnecessary values in the weight tensor. In practice, neural network parameters' values are set to zero to remove low-weight connections between the layers of a neural network.

Its function Tensors with several values set to zero can be considered sparse. This results in important benefits: *Compression.* Sparse tensors are amenable to compression by only keeping the non-zero values and their corresponding coordinates. *Speed.* Sparse tensors allow us to skip otherwise unnecessary computations involving the zero values.

[2]For detail see Tensorflow tutorial.

2.11 Loss functions

2.11.1 Cross entropy loss

Cross entropy describes the distance between two probability distributions. The smaller the cross entropy, the closer the two are.

$$P = Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{4}$$

$$\hat{y} = P(y=1|x), \quad 1-\hat{y} = P(y=0|x) \tag{5}$$

$$P(y|x) = \hat{y}^y * (1-\hat{y})^{1-y} \tag{6}$$

$$logP(y|x) = y * log\hat{y} + (1-y)log(1-\hat{y}) \tag{7}$$

$$BSE = -\frac{1}{N}\sum_{i=1}^{N}[g_i \cdot log(p_i) + (1-g_i) \cdot log(1-p_i)] \tag{8}$$

In this binary Cross Entropy loss, g is the class of ground truth, and p is probability of predicted value.

2.11.2 Weighted cross-entropy

The wighted cross-entropy(WCE) has been used in [22] to address the unbalanced data. The two-class form of WCE can be expressed as

$$WCE = -\frac{1}{N}\sum_{i=1}^{N} wg_i \log(p_i) + (1-g_i)\log(1-p_i), \tag{9}$$

where $w = (N - \sum_{i=1}^{N} p_i)/\sum_{i=1}^{N} p_i$, g_i is ground truth, p_i is predicted value.

2.11.3 Focal loss

Focal loss[15] is the variant of Cross Entropy. It solved the extreme object-background class imbalance problem by adding two coefficients α and γ to balance the weight of one-class examples, and adjust the rate to increase the importance of correcting mis-classified examples.

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{10}$$

If we expand the formula 10, it will be like formula 11.

$$FL = -\frac{1}{N}\sum_{i=1}^{N}[\alpha \cdot g_i^{\gamma} \cdot \log(p_i) - (1-\alpha) \cdot (1-g_i)^{\gamma} \cdot \log(1-p_i)] \tag{11}$$

The source code of focal loss is from Github.

Listing 1: Focal loss

```
def focal_loss(gamma=2, alpha=0.75):
    def focal_loss_fixed(y_true, y_pred):
        eps = 1e-12-
        y_pred=K.clip(y_pred,eps,1.-eps)
        eq1 = tf.equal(y_true, 1)
        eq0 = tf.equal(y_true, 0)
        pt1 = tf.where(eq1, y_pred, tf.ones_like(y_pred))
        pt0 = tf.where(eq0, y_pred, tf.zeros_like(y_pred))
        p = K.pow(1. - pt1, gamma) * K.log(pt1)
        1-p = (1-alpha)*K.pow( pt0, gamma)*K.log(1-pt0)
        return -K.mean(alpha*p)-K.mean((1-alpha)*(1-p))
    return focal_loss_fixed
```

2.11.4 Dice loss

Dice coefficient [6] calculates the overlap between ground truth and predicted images. If the Dice score is 1, which indicates that the predicted image matches perfectly with ground truth data. Here, in order to make loss converge, let the Dice be negative and plus 1.

$$DICE = 1 - \frac{2\sum_{i=1}^{N} p_i g_i}{\sum_{i=1}^{N}(p_i + g_i)} \tag{12}$$

2.11.5 Generalised Dice loss

A known metric Generalized Dice overlap is used as loss function in deep learning[26].

$$GDL = 1 - 2\frac{\sum_{l=1}^{2} w_l \sum_{i=1}^{N} g_i p_i}{\sum_{l=1}^{2} w_l \sum_{i=1}^{N} g_i + p_i}, \tag{13}$$

where $w_l = 1/(\sum_{i=1}^{N} g_{li})^2$, g_i is ground truth, p_i is predicted value.

2.11.6 Jarccard loss

Jarccard index is quite similar with Dice coefficient, they have the same meaning, but a little difference in appearance.

$$Jaccard = 1 - \frac{\sum_{i=1}^{N} p_i g_i}{\sum_{i=1}^{N} p_i + \sum_{i=1}^{N} g_i - \sum_{i=1}^{N} p_i g_i} \tag{14}$$

2.11.7 Hausdorff loss

Hausdorff distance is the maximum distance of a set to the nearest point in the other set. However the Hausdorff distance is not suitable as Hausdorff loss.

Table 1: Counts of agreement and disagreement from ground truth and predicted results on a dichotomous data set

Item	Ground truth			
Predicted result	class	+	-	Total
	+	a	b	a+b(P_+)
	-	c	d	c+d(P_-)
	Total	a+c(G_+)	b+d(G_-)	n

Because it is not derivative[21].

$$Hausdorff = max\{h(P,G), h(G,P)\}, \tag{15}$$

where P and G are the set of predicted and ground truth binary labels, respectively.

2.11.8 Tversky loss

The Dice loss is a harmonic mean of precision and recall thus weighs false positives(FPs)and false negatives(FNs)equally. To address the issue of data imbalance and achieve a better trade off between precision($Precision = TP/(TP + FP)$) and recall($Recall = TP/(TP + FN)$), [24]proposed a loss based on the Tversky similarity[1]. The Tversky index is defined as:

$$S(P,G,\alpha,\beta) = \frac{|PG|}{|PG| + \alpha|P\backslash G| + \beta|G\backslash P|} \tag{16}$$

P and G are the set of predicted and ground truth binary labels, respectively. Where α and β control the magnitude of penaltis for FPs and FNs, respectively.

The Tversky loss function is defined as following formulation:

$$T(\alpha,\beta) = \frac{\sum_{i=1}^{N} p_{0i}g_{0i}}{\sum_{i=1}^{N} p_{0i}g_{0i} + \alpha\sum_{i=1}^{N} p_{0i}g_{1i} + \beta\sum_{i=1}^{N} p_{1i}g_{0i}} \tag{17}$$

Where in the output of the softmax(or sigmoid)layer, the p_{0i} is the probability of element i be a lesion(object) and p_{1i} is the probability of element i be a non-lesion(background). Also, g_{0i} is 1 for a lesion element and 0 for a non-lesion element and vice versa for the g_{1i}.

2.12 Kappa coefficient

2.12.1 Definition

In the dichotomous segmentation, we divide two classes into positive and negative class. Shown as table 1.

Kappa coefficient is a chance-corrected measure of agreement, defined by:

$$\kappa = \frac{observed\ agreement - chance\ agreement}{1 - chance\ agreement} = \frac{p_o - p_c}{1 - p_c} \tag{18}$$

in which p_o is the observed percentage of agreement(the percentage of targets rated the same by all raters), which is also called accuracy.

$$p_o = \frac{a+d}{n} \tag{19}$$

And p_c is defined as follow:

$$p_c = \frac{(a+b)(a+c)+(b+d)(c+d)}{n^2} \tag{20}$$

As we know, $P_- = (c+d)/n, P_+ = (a+b)/n, G_- = (b+d)/n, G_+ = (a+c)/n$, We can rewrite p_c in the way below:

$$p_c = P_+ \cdot G_+ + P_- \cdot G_- \tag{21}$$

From Equation 18, we know that, when κ is 1, the predicted data are perfectly agreed with ground truth; Generally when $\kappa > 0.75$,proves the results are good; If $\kappa < 0.4$, the results are bad.

2.12.2 Kappa coefficient as loss function

If we regard the Kappa coefficient as loss functions, we just change it into $L = 1 - \kappa$, which is:

$$L = \frac{1-p_o}{1-p_c} \tag{22}$$

2.12.3 Derivation

The gradient of the loss in Equation 22 with respect to P_+ and P_- can be calculated as:

$$\frac{\partial L}{\partial P_+} = (1-p_o)\frac{1}{(1-p_c)^2} \cdot \frac{\partial p_c}{\partial P_+} = \frac{(1-p_o)\cdot G_+}{(1-P_+G_+ - P_-G_-)^2} \tag{23}$$

$$\frac{\partial L}{\partial P_-} = (1-p_o)\frac{1}{(1-p_c)^2} \cdot \frac{\partial p_c}{\partial P_-} = \frac{(1-p_o)\cdot G_-}{(1-P_+G_+ - P_-G_-)^2} \tag{24}$$

In the same circumstance, the κ index is more critique than the Dice coefficient. Because:

$$\kappa - DICE = \frac{2(ad-bc)}{(a+b)(b+d)+(a+c)(c+d)} - \frac{2a}{(a+b)+(a+c)} < 0 \tag{25}$$

2.13 Kappa loss and implementation

2.13.1 Definition

$$K = 1 - \frac{2\sum_{i=1}^{N} p_i g_i - \sum_{i=1}^{N} p_i \cdot \sum_{i=1}^{N} g_i/N}{\sum_{i=1}^{N} p_i + \sum_{i=1}^{N} g_i - 2\sum_{i=1}^{N} p_i g_i/N} \tag{26}$$

with smooth item:

$$K = (1 - \frac{2\sum_{i=1}^{N} p_i g_i - \sum_{i=1}^{N} p_i \cdot \sum_{i=1}^{N} g_i/N + smooth}{\sum_{i=1}^{N} p_i + \sum_{i=1}^{N} g_i - 2\sum_{i=1}^{N} p_i g_i/N + smooth}) * smooth \tag{27}$$

Table 2: Experiment environment

Item	Setting
Dataset	melanomal(RGB 256*256)99 for training, 20 for test
If data augmentation	yes
Architecture	U-Net
Parameters	31,031,685
Learning rate	1e-4
Batch size	2
Steps per epoch/epochs	99/50

Table 3: Performance on Kappa loss

Training		Testing	
mean loss	0.846	loss	0.765
std of loss	0.024		
mean accuracy	0.964	accuracy	0.981
std of accuracy	0.044		
mean IOU	0.331	IOU	0.309
std of IOU	0.022		

2.13.2 Derivation

$$\frac{\partial K}{\partial p_i} = 2\frac{[\sum g_i \sum p_i + (1-1/2N)\sum g_i^2 - \sum p_i g_i + \sum p_i g_i^2/N^2 - \sum p_i \sum g_i^2/N^2]}{(\sum p_i + \sum g_i - 2\sum p_i g_i/N)^2} \tag{28}$$

2.13.3 Code

Listing 2: Kappa loss

```
def Kappa_loss(y_true, y_pred):
    Gi = K.flatten(y_true)
    Pi = K.flatten(y_pred)
    N = 256.0*256.0
    numerator = 2*K.sum(Pi * Gi)-K.sum(Pi)*K.sum(Gi)/N
    denominator = K.sum(Pi)+K.sum(Gi)-2*K.sum(Pi*Gi)/N
    Kappa_loss = 1 - numerator/denominator
    return Kappa_loss
```

2.13.4 Results

Table 2 is experiment settings, in which the different learning rates can influence the final results.

Table 3 is the result under Kappa loss. And the Figure 5 shows the performance under Kappa loss.

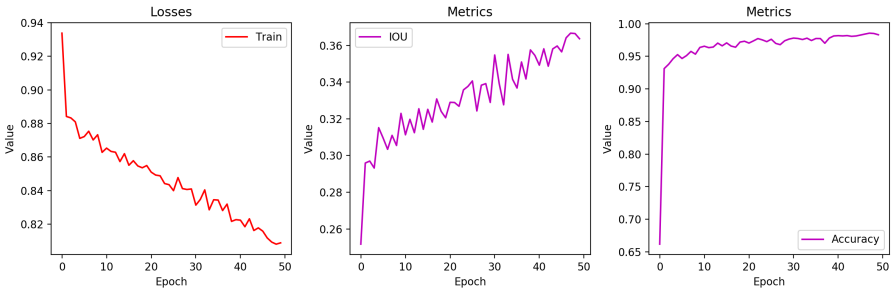


Figure 5: Performance under Kappa loss

Table 4: Performance on Dice loss

Training		Testing	
mean loss	82.000	loss	84.149
std of loss	0.393		
mean accuracy	0.113	accuracy	0.098
std of accuracy	0.002		
mean IOU	0.179	IOU	0.156
std of IOU	0.003		

Under the same experiment condition, the performance of Dice loss is shown as Table 4 and Figure 6. In this case, the training stage is not stable and not learning something due to the learning rate setting.

The result is not satisfying, so another experiment was done under the learning rate is 1e-5. The result was as follows:

We can see from the table 6, the metrics IOU is not good enough. So we add a smooth item just like Dice coefficient.

Listing 3: Kappa loss with smooth item

```
def Kappa_loss(y_true, y_pred, smooth = 100):
```

Table 5: Performance on Dice loss when learning rate is 1e-5

Training		Testing	
mean loss	41.606	loss	37.904
std of loss	4.384		
mean accuracy	0.971	accuracy	0.979
std of accuracy	0.018		
mean IOU	0.575	IOU	0.613
std of IOU	0.043		

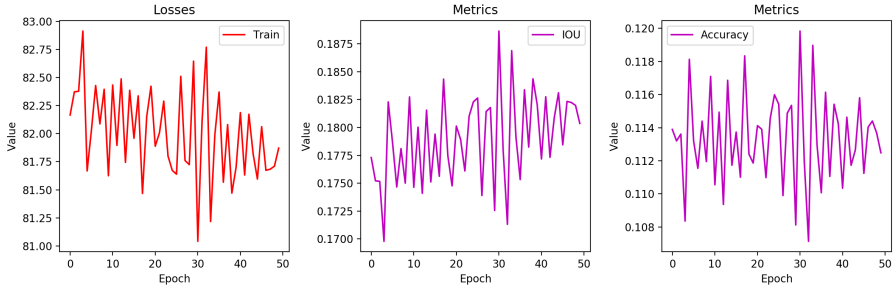


Figure 6: Performance under Dice loss

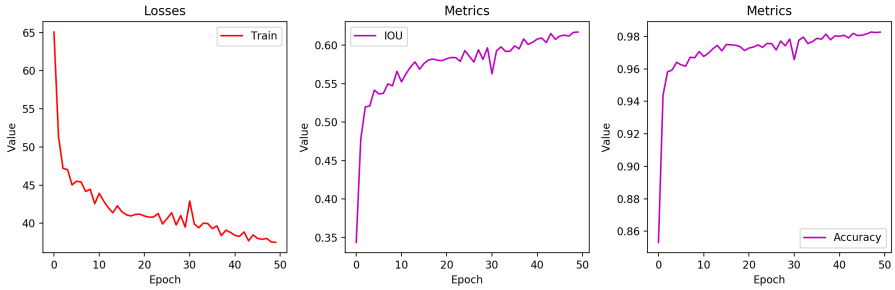


Figure 7: Performance under Dice loss when learning rate is 1e-5

Table 6: Performance on Kappa loss when learning rate is 1e-5

Training		Testing	
mean loss	0.798	loss	0.818
std of loss	0.007		
mean accuracy	0.947	accuracy	0.978
std of accuracy	0.047		
mean IOU	0.289	IOU	0.259
std of IOU	0.007		

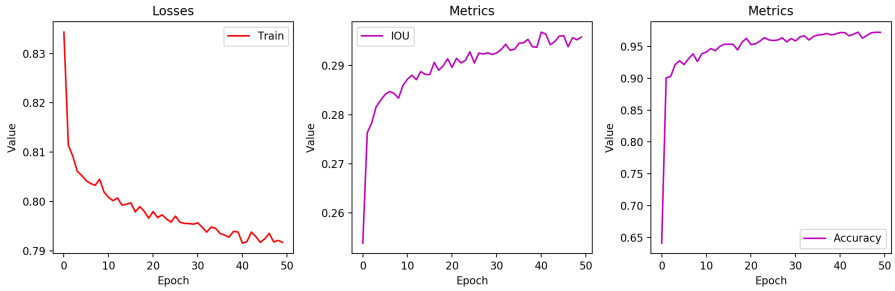


Figure 8: Performance under Kappa loss when learning rate is 1e-5

Table 7: Performance on Kappa loss with smooth item when learning rate is 1e-5

Training		Testing	
mean loss	45.181	loss	41.101
std of loss	3.986		
mean accuracy	0.973	accuracy	0.979
std of accuracy	0.010		
mean IOU	0.576	IOU	0.611
std of IOU	0.039		

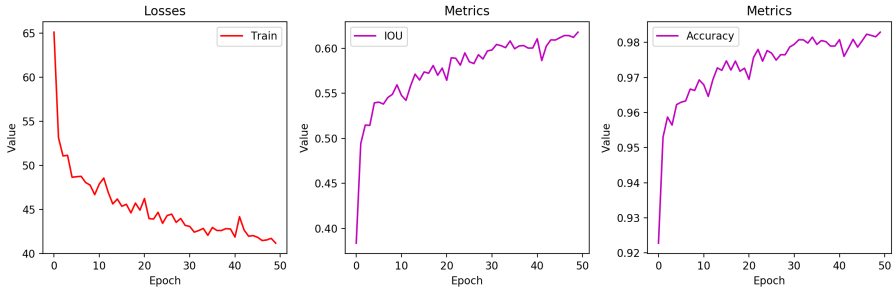


Figure 9: Performance under Kappa loss with smooth item when learning rate is 1e-5

```
Gi = K.flatten(y_true)
Pi = K.flatten(y_pred)
N = 256.0*256.0
numerator = 2*K.sum(Pi * Gi)-K.sum(Pi)*K.sum(Gi)/N
denominator = K.sum(Pi)+K.sum(Gi)-2*K.sum(Pi*Gi)/N
Kappa_loss = (1 - (numerator+smooth)/(denominator+smooth))*smooth
return Kappa_loss
```

And we test the program again, the result is in table 7 and figure 9.

3 Method

3.1 Dice loss with smooth item

We add a smooth item in the Dice loss function.

$$DICE = (1 - \frac{2\sum_{i=1}^{N} p_i g_i + smooth}{\sum_{i=1}^{N}(p_i + g_i) + smooth}) * smooth \tag{29}$$

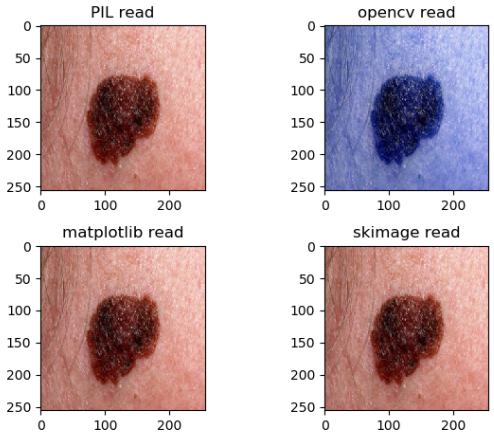


Figure 10: Image mode under different image reading methods

Without smooth on the outside of Dice loss:

$$DICE = 1 - \frac{2\sum_{i=1}^{N} p_i g_i + smooth}{\sum_{i=1}^{N}(p_i + g_i) + smooth} \tag{30}$$

Derivation of smooth is formula 31. So theoretically, the larger smooth is, the Dice loss with smooth item optimizes(converges) more quickly.

$$\frac{dDICE}{dsmooth} = \frac{(P + G - 2PG)(P + G)}{(P + G + smooth)^2} \tag{31}$$

3.2 Jarccard loss with smooth item

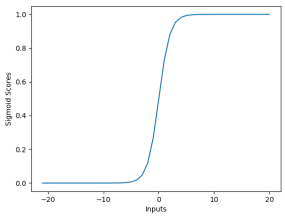
$$J(A, B) = (1 - \frac{\sum_{i=1}^{N} p_i g_i + smooth}{\sum_{i=1}^{N} p_i + \sum_{i=1}^{N} g_i - \sum_{i=1}^{N} p_i g_i + smooth}) * smooth \tag{32}$$

3.3 Image encoder

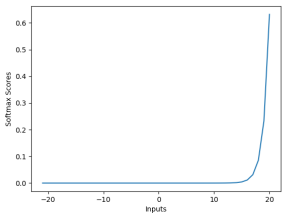
In this work, we compared four image reading methods using python libraries. They are image reading functions from package PIL[28], package cv2[3], package matplotlib[13], package skimage[27].

3.4 Activation function

No matter in classification or segmentation problem, the last layer must output a probability value to show the pixel or image belong to which class. So for the last layer, we need to decide to use softmax function or sigmoid function. While as for the previous layers of U-Net and DenseNet, ReLU[18] is applied as activation functions.



(a) Sigmoid curve



(b) Softmax curve

Figure 11: Sigmoid function and softmax function

3.4.1 Sigmoid function

$$S(t) = \frac{1}{1 + \exp(-t)} \tag{33}$$

3.4.2 Softmax function

$$F(X_i) = \frac{\exp(X_i)}{\sum_{j=0}^{K} \exp(X_j)} \quad for \quad i = 0, \cdots, K \tag{34}$$

Frome the above equations and figures of two functions we can know that Softmax function is used for the multi-classification task. Figure 11 show that the high value will have the high probability. And sigmoid function is used for the binary classification task.

4 Experiments

4.1 Data input

We use public Skin Cancer Detection dataset to do the experiments. The original dataset has 119 RGB images and labels respectively. We split them into 99 images as training set and 20 images as testing set.

Different size of images can have different effects both in speed and segmentation effects. Obviously, larger images takes longer time than small size images. So, our work compared the segmentation accuracy between large and small data sets. We resized our data into 512*512, 256*256, 128*128 3 types of datasets.

Also, we compared different image reading methods during the test stage, we found that different image reading methods have the contrary visual segmented effects.

4.2 Data augmentation

In supervised learning, it requires training images and ground truth images. Especially in deep neural network, it requires more data to learn the features.

Table 8: Performance on Softmax activation fucntion

Training		Testing	
mean loss	80.855	loss	82.951
std of loss	0.563		
mean accuracy	0.113	accuracy	0.098
std of accuracy	0.003		
mean IOU	0.191	IOU	0.170
std of IOU	0.005		

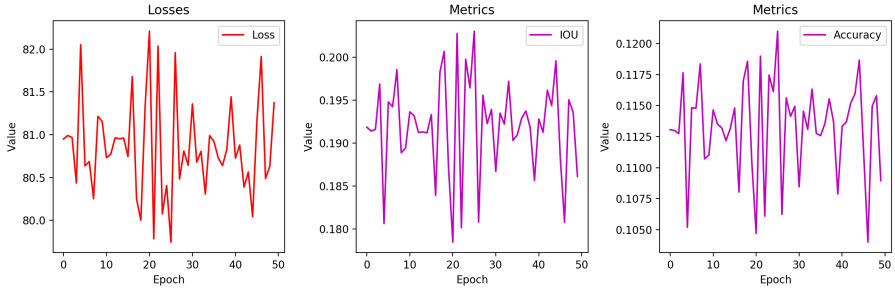


Figure 12: Performance under Softmax activation function

However, usually the original datasets is not enough because the labeling work requires time and expert ability. So, in this work, we do the data augmentation to generate more data to feed the deep neural network. Specifically, we set some parameters like rotation, scaling, flipping to create train images and corresponding ground truth.

4.3 Hyper-parameters

Hyper-parameters are handled by human. In this work, we set learning rate, loss functions as hyper-parameters to observe the different effects.

4.4 Softmax and Sigmoid activation function

We used DenseNet as baseline, and compared softmax and sigmoid activation functions in one classes image segmentation. As we can see from table 8 and figure 12, the DenseNet with softmax did not learn anything from images.

Under the same circumstance(same hyper parameters and networks), table 9 and figure 13 gave the information about the performance with the sigmoid activation function based on DenseNet, which shows that the networks learned very well.

From the above results, we can draw a conclusion that the sigmoid activation function is more preferable in one-class image segmentation than softmax

Table 9: Performance on Sigmoid activation function

Training		Testing	
mean loss	13.728	loss	16.537
std of loss	8.259		
mean accuracy	0.970	accuracy	0.953
std of accuracy	0.027		
mean IOU	0.863	IOU	0.835
std of IOU	0.083		

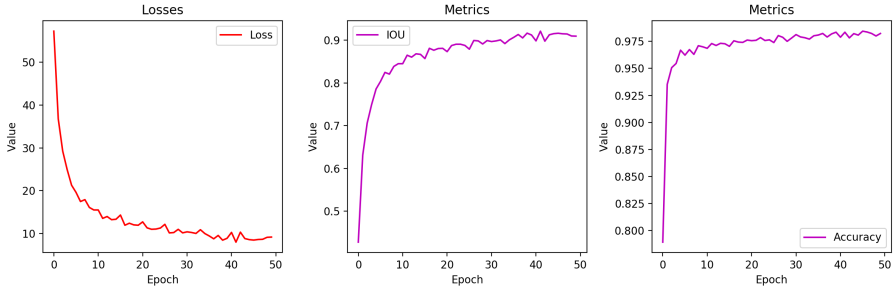


Figure 13: Performance under Sigmoid activation function

function.

4.5 U-Net and DenseNet

We compared two deep neural networks, which are U-Net and DenseNet respectively as table 10. Obviously, the DenseNet surpass U-Net in terms of loss converging and metrics. At the same time, because the number of parameters is less than U-Net, so the computation time is similar with U-Net, although the number of DenseNet's layer is more than 5 times as U-Net.

Table 10: Comparison between U-Net and DenseNet

	U-Net	DenseNet
layers #	38	213
parameters #	31,031,685	2,102,816
loss	45.181/41.101	**19.216/20.380**
accuracy	0.973 /0.979	0.971/0.956
IOU	0.576/0.611	**0.859/0.846**

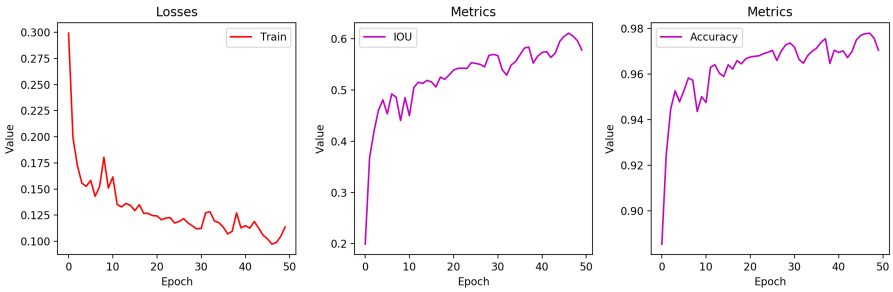


Figure 14: The performance under image size: 128*128

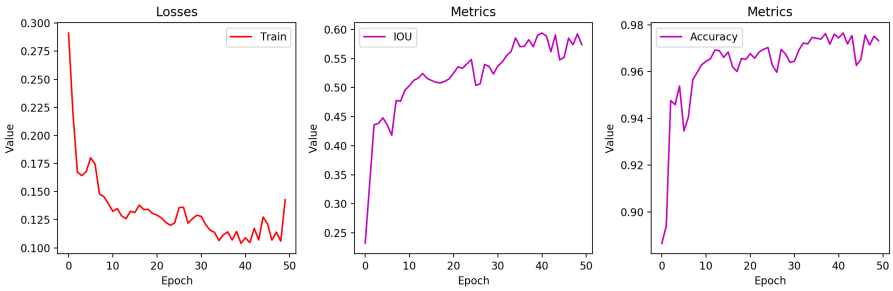


Figure 15: The performance under image size:256*256

4.6 Evaluation metrics

We use two evaluation metrics, one is segmentation accuracy, the other is Dice coefficient.

4.7 Results

4.7.1 Data

We compared 3 different sizes of the same type images. They are 128*128, 256*256, 512*512, as figure 14 shown. In table 11, the performance is the best when the image size is 256*256. So the following experiments are done on that size of image.

4.7.2 Data augmentation

As figure 17 shown, the performance with data augmentation(9801 images) is better than that without data augmentation(99 images).

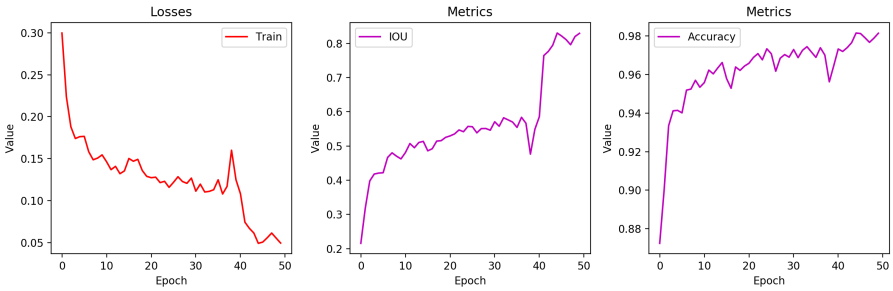
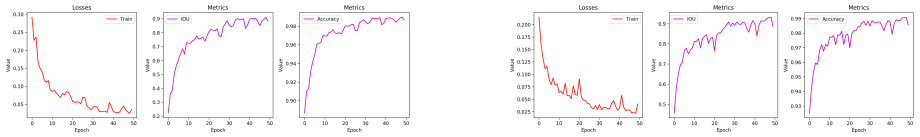


Figure 16: The performance under image size:512*512

Table 11: Performance under different image sizes

Item	Training			Test		
Image size	Loss	Accuracy	IOU	Loss	Accuracy	IOU
128*128	0.131	0.963	0.527	0.107	0.902	0.620
256*256	0.133	0.963	0.520	**0.09**	**0.978**	**0.625**
512*512	0.126	0.962	0.560	0.543	0.907	0.463



(a) Without data augmentation (b) With data augmentation

Figure 17: Sigmoid function and softmax function

Table 12: Performance under different loss functions

Item	Training			Test		
Loss functions	Loss	Accuracy	DICE	Loss	Accuracy	DICE
Cross Entropy	0.5	0.5	0.5	0.5	0.5	0.5
Focal loss	0.5	0.5	0.5	0.5	0.5	0.5
Dice loss	0.5	0.5	0.5	0.5	0.5	0.5
Jaccard loss	0.5	0.5	0.5	0.5	**0.5**	0.5

4.7.3 Learning rate and loss functions

5 Conclusion

References

[1] Tversky Amos. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[5] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. *CoRR*, abs/1606.06650, 2016.

[6] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[7] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[8] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[11] M. Holschneider, R. Kronland-Martinet, J. Morlet, and Ph. Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In Jean-Michel Combes, Alexander Grossmann, and Philippe Tchamitchian, editors, *Wavelets*, pages 286–297, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[12] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[13] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[14] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *CoRR*, abs/1210.5644, 2012.

[15] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[16] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[17] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. *CoRR*, abs/1606.04797, 2016.

[18] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[19] Ozan Oktay, Jo Schlemper, Loïc Le Folgoc, Matthew C. H. Lee, Mattias P. Heinrich, Kazunari Misawa, Kensaku Mori, Steven G. McDonagh, Nils Y. Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention u-net: Learning where to look for the pancreas. *CoRR*, abs/1804.03999, 2018.

[20] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[21] Javier Ribera, David Güera, Yuhao Chen, and Edward J. Delp. Weighted hausdorff distance: A loss function for object localization. *CoRR*, abs/1806.07564, 2018.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[23] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017.

[24] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks. *CoRR*, abs/1706.05721, 2017.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, May 2015.

[26] Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sébastien Ourselin, and M. Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. *CoRR*, abs/1707.03237, 2017.

[27] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.

[28] wiredfool, Alex Clark, Hugo, Andrew Murray, Alexander Karpinsky, Christoph Gohlke, Brian Crowell, David Schmidt, Alastair Houghton, Steve Johnson, Sandro Mani, Josh Ware, David Caro, Steve Kossouho, Eric W. Brown, Antony Lee, Mikhail Korobov, Michał Górny, Esteban Santana Santana, Nicolas Pieuchot, Oliver Tonnhofer, Michael Brown, Benoit Pierre, Joaquín Cuenca Abela, Lars Jørgen Solberg, Felipe Reyes, Alexey Buzanov, Yifu Yu, eliempje, and Fredrik Tolf. Pillow: 3.1.0, January 2016.