Highlights

ADS_UNet: A Nested UNet for Histopathology Image Segmentation

Yilong Yang,Srinandan Dasmahapatra,Sasan Mahmoodi

- We propose ADS_UNet that integrates cascade training and AdaBoost algorithm.
- We supervise layers of UNet to learn useful features in a manner that is learnable.
- The importance of layers varies across the training and contributes differently.
- The ADS_UNet achieves state-of-the-art performance and is computationally efficient.

ADS_UNet: A Nested UNet for Histopathology Image Segmentation

Yilong Yang[a,*], Srinandan Dasmahapatra[a] and Sasan Mahmoodi[a]

[a]School of Electronics and Computer Science, University of Southampton, University Road, Southampton, SO17 1BJ, Hampshire, United Kingdom

ARTICLE INFO

Keywords:
Segmentation
UNet
AdaBoost
Histopathology
Ensemble

ABSTRACT

The UNet model consists of fully convolutional network (FCN) layers arranged as contracting encoder and upsampling decoder maps. Nested arrangements of these encoder and decoder maps give rise to extensions of the UNet model, such as UNete and UNet++. Other refinements include constraining the outputs of the convolutional layers to discriminate between segment labels when trained end to end, a property called deep supervision. This reduces feature diversity in these nested UNet models despite their large parameter space. Furthermore, for texture segmentation, pixel correlations at multiple scales contribute to the classification task; hence, explicit deep supervision of shallower layers is likely to enhance performance. In this paper, we propose ADS UNet, a stage-wise additive training algorithm that incorporates resource-efficient deep supervision in shallower layers and takes performance-weighted combinations of the sub-UNets to create the segmentation model. We provide empirical evidence on three histopathology datasets to support the claim that the proposed ADS UNet reduces correlations between constituent features and demonstrate that ADS_UNet outperforms state-of-the-art Transformer-based models by 1.08 and 0.6 points on CRAG and BCSS datasets, but requires only 37% of GPU consumption and 34% training time that required by Transformers.

1. Introduction

The fully convolutional neural network (FCN) [21], trained end-to-end on per-pixel labels, is considered a milestone in image segmentation using deep networks. It was then extended by Ronneberger et al. [27] to include a large number of up-sampled features concatenated using skip connections with the encoded convolutional features. They named the network a UNet after a geometrical laying out of the network topology in a u-shape. Zhou et al. [34] modified the UNet architecture by adding more nodes and connections to capture low-level correlation of distributed semantic attributes. The resulting architectures, known as UNet^e (*e* denotes ensemble) and UNet++, used class labels to guide the outputs of decoder layers (called deep supervision) to learn highly discriminative features.

Both UNet^e and UNet++ can be classified as ensemble models, in which multiple models are created to obtain better performance than each constituent model alone [25]. A property that is present in a good ensemble is the diversity of the predictions made by contributing models. However, end to end training of deep networks tends to correlate intermediate layers [16], hence the collaborative learning of constituent UNets adopted by UNet^e and UNet++ induces learned features to be correlated. Such learning runs counter to the idea of feature diversity pursued by ensemble models. Moreover, simple averaging performed in UNet^e, disregarding the difference in the performance of each member also restricts the final predictive performance of the ensemble.

Based on the work of UNet^e and UNet++, we pose several questions: 1) can each constituent model be forced to extract decorrelated features during training, to guarantee prediction diversity? 2) can the outputs of constituent models, sensitive to different spatial resolutions, be weighted differently when they are integrated in the final segmentation? 3) can we provide deep supervision for encoders directly rather than by supervising the up-sampled decoders? To address these questions, we propose the Adaboosted Deeply Supervised UNet (ADS_UNet). The key contributions of our work can be summarized as follows:

1) We integrate deep supervision, cascade learning, and AdaBoost into the proposed ADS_UNet, a stage-wise additive training algorithm, in which multiple UNets of varying depths are trained sequentially to enhance the feature diversity of constitutent models. Extensive experiments demonstrate that ADS_UNet is effective in boosting segmentation performance.

2) In our deep supervision scheme, we down-sample the mask to have the same size as feature maps of hidden layers to compute pixel-wise loss, instead of up-sampling features. This modification retains the advantages of deep supervision and yet reduces computation cost and GPU memory consumption.

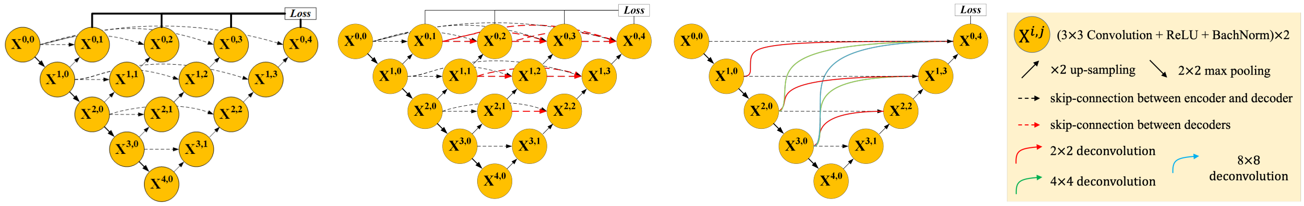
3) Instead of assigning balanced weights to all supervised layers, we introduce a learnable weight for the loss of each supervised layer to characterize the importance of features learned by layers.

4) We conduct a comprehensive ablation study to systematically analyze the performance gain achieved by the ADS_UNet.

2. Related Work

In this section, we review the works related to UNet and its variants, deep supervision and AdaBoost, which are the main components of our architecture.

*Corresponding author
Yilong.Yang@soton.ac.uk (Y. Yang); sd@ecs.soton.ac.uk (S. Dasmahapatra); sm3@ecs.soton.ac.uk (S. Mahmoodi)
ORCID(s): 0000-0002-2595-7883 (Y. Yang)


Loss
Loss
Loss
$X^{i,j}$ (3×3 Convolution + ReLU + BachNorm)×2
×2 up-sampling
2×2 max pooling
skip-connection between encoder and decoder
skip-connection between decoders
2×2 deconvolution
4×4 deconvolution
8×8 deconvolution

(a) UNet^e (b) UNet++ (c) CENet (d) Legend

Figure 1: Comparison of UNet^e (a), UNet++ (b), CENet (c), and proposed UNet (c). UNet++ is constructed from UNet^e by introducing skip-connections (red dashed lines in (b)) between decoder nodes. CENet disregards inner decoder nodes and adopts deconvolution and concatenation to harvest multi-scale context clues between encoder and decoder nodes.

2.1. UNet family

UNet [27] consists of a down-sampling path to capture context, and a symmetric up-sampling path to expand feature maps back to the input size. The down-sampling part has a FCN-like architecture that extracts features with 3×3 convolutions. The up-sampling part uses deconvolution to reduce the number of feature maps while increasing their area. Feature maps from the down-sampling part of the network are copied and concatenated to the up-sampling part to ensure precise localization.

Building on the success of UNet, several variants have been proposed to further improve segmentation performance. Here we describe the networks UNet^e and UNet++ [34], whose simplified architectures are given in Figure 1. UNet^e is an ensemble architecture, which combines UNets of varying depths into one unified structure. Note that deep supervision is required to train UNet^e in an end-to-end fashion. In order to allow deeper UNets to offer a supervision signal to the decoders of the shallower UNets in the ensemble and address the potential loss of information, the UNet++ connects the decoder nodes, to enable dense feature propagation along skip connections and thus more flexible feature fusion at the decoder nodes. The difference between UNet++ and UNet^e is that there are skip-connections between decoder nodes in UNet++ (highlighted in red in Figure 1(b)). Zhou et al. [33] proposed the contextual ensemble network (CENet), where the contextual cues are aggregated via densely up-sampling the features of the encoder layers to the features of the decoder layers. This enables CENet to capture multi-scale context information. While UNet++ and CENet yield higher performance than UNet, it does so by introducing dense skip connections that result in a huge increase of parameters and computational cost.

Most recent, building upon the success of Vision Transformer [7] on image classification tasks, self-attention modules have also been integrated into UNet-like architectures for accurate segmentation. Luo et al. [22] proposed the hybrid ladder transformer (HyLT), in which the authors use bidirectional cross-attention bridges at multiple resolutions for the exchange of local and global features between the CNN- and transformer-encoding paths. The fusion of local and global features renders HyLT robust compared to other CNN-, transformer- and hybrid- methods for image perturbations. Gao et al. [12] presented MedFormer, in which an efficient bidirectional multi-head attention (B-MHA) is proposed to eliminate redundant tokens and reduce the quadratic complexity of conventional self-attention to a linear level. Furthermore, the B-MHA liberates the constraints of model design and enables MedFormer to extract global relations on high-resolution token maps towards the fine-grained boundary modelling. Ma et al. [23] proposed a hierarchical context-attention transformer-based architecture (HT-Net), which introduces an axial attention layer to model pixel dependencies of multi-scale feature maps, followed by a context-attention module that captures context information from adjacent encoder layers.

CRediT authorship contribution statement

Yilong Yang: Conceptualization, Methodology, Software, Formal analysis, Writing-Original Draft. **Srinandan Dasmahapatra:** Supervision, Formal analysis, Writing-Reviewing & Editing. **Sasan Mahmoodi:** Supervision, Formal analysis, Writing-Reviewing & Editing.

2.2. Deep supervision

A deeply supervised network (DSN)[20] introduced classification outputs to hidden layers as well as the last layer output as is the convention. This was shown to increase discriminative power of learned features in shallow layers and robustness to hyper-parameter choice.

Despite the fact that the original DSN was proposed for classification tasks, deep supervision can also be used for image segmentation. Dou et al. [8] introduced deep supervision to combat potential optimization difficulties and concluded that the model acquired a faster convergence rate and greater discriminability. Based on the UNet architecture, [35] introduced a supervision layer to each encoder/decoder block. Their method is very similar to our proposed supervision scheme; the difference lies in how the loss between the larger-sized ground-truth and the smaller-sized output of hidden layers is calculated. Note that the dimension of feature maps of the hidden layers are gradually reduced and become much smaller than that of the ground-truth mask, because of the down-sampling operation. In [8] and [35], deconvolutional layers were used to up-sample feature maps back to the same size as the ground-truth mask. Evidently, the additional deconvolutional layers introduce more parameters and more computational overhead. Although it was pointed out in [21] that one can learn arbitrary interpolation functions, bilinear interpolation was adopted in [32] to up-sample feature maps with no reduction in performance compared to learned deconvolutions. All of the aforementioned literature solve the dimension mismatch problem by up-sampling feature maps. However, in our deep supervision scheme, we perform average pooling to down-sample the ground-truth mask to the same size as feature maps of hidden layers. This reduces the amount of computation and is more GPU memory efficient.

2.3. AdaBoost

AdaBoost (Adaptive Boosting) [10] is a very successful ensemble classifier, which has been widely used in binary classification tasks. The idea of AdaBoost is based on the assumption that a highly accurate prediction rule can be obtained by combining many relatively weak and inaccurate rules. This was re-derived in [11] as a gradient of an exponential loss function of a stage-wise additive model. Such an additive model was extended to the multi-class case by [13], who proposed SAMME (Stage-wise Additive Modeling using a Multi-class Exponential loss function) that naturally extends the original AdaBoost algorithm to the multi-class case without reducing it to multiple two-class problems. The detailed iterative procedure of multi-class AdaBoost is described in Algorithm 2 of [13].

Starting from equally weighted training samples, the AdaBoost trains a classifier f_t ($t \in \{1, 2, ..., T\}$ the iteration index) iteratively, re-weighting the training samples in the process. A misclassified item x_i is assigned higher weight w_i^t so that the next iteration of the training pays more attention to it. After each classifier f_t is trained, it is assigned a weight based on its error rate ϵ_t on the training set. For the integrated output of the classifier ensemble, the more accurate classifier is assigned a larger weight α_t to have more impact on the final outcome. A classifier with $< \frac{1}{C}\%$ accuracy (less than random guessing for C target classes) is discarded. T classifiers will be trained after repeating this procedure for T times. The final labels can be obtained by the weighted majority voting of these T classifiers.

An adaptive algorithm, Adaboost-CNN, which combines multiple CNN models for multi-classification was introduced in [30]. In AdaBoost-CNN, all the weak classifiers are convolutional neural networks and have the same architecture. Instead of training a new CNN from scratch, they transfer the parameters of the prior CNN to the later one and then train the new CNN for only one epoch. This achieves better performance than the single CNN, but at the cost of increasing the number of parameters several fold. Curriculum learning [4] is related to boosting algorithms, in that the training schedule gradually emphasizes the difficult examples. [5] demonstrated that better performance can be achieved by forcing UNet to learn from easy to difficult scenes. However, the difficulty level of training samples is predefined according to the size of the target to be segmented, rather than calculated by the network itself.

3. Method

Ensemble learning is often justified by the heuristic that each base learner might perform well on some data and less accurately on others for some learned features, to enable the ensemble to override common weaknesses. To this end, we seek enhanced segmentation performance of the model by enabling diverse feature maps to be learned. We propose the ADS_UNet algorithm, which adopts a layer-wise cascade training approach [9, 3, 24] but with an added component that re-weights training samples to train each base learner in sequence. We evaluate the role of feature map diversity in section 5.3.

3.1. Computation and Memory Efficient Deep Supervision

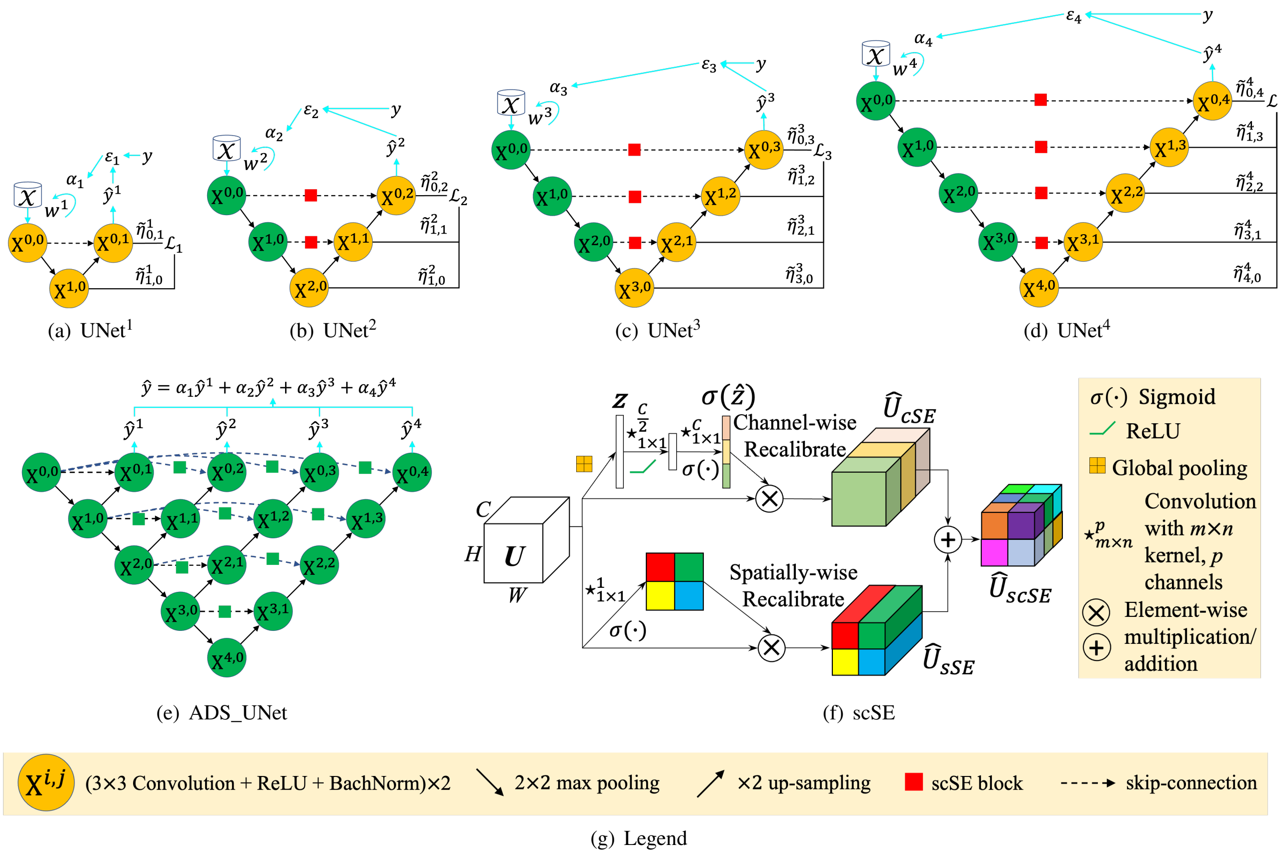
As we mentioned in the introduction section, the UNet^e and UNet++ [34] offer deep supervision to shallower layers by gradually up-sampling feature maps to the size of the mask, which is computation and GPU memory expensive. To ease computational burden, we average-pool the mask to have the same size as feature maps. The advantage of this change is that we no longer need to train deconvolutional weights for intermediate blocks to obtain feature maps with the same dimension as the ground-truth mask. This is of potential benefit for texture segmentation, as relevant textural characteristics occur at multiple length scales, and is not confined to the location of the mask boundary. We adopted UNet^{d}s, whose hidden layers been supervised, as base learners of the proposed ensemble model. Given the input image x and the network, we define the probability map generated at block $\mathrm{X}^{i,j}$ as:

$$\hat{y}^{i,j}(x) = \mathrm{softmax}(\mathrm{X}^{i,j}(x)) \tag{1}$$

The mapping $\mathrm{X}^{i,j}(\cdot) : \boldsymbol{X} \mapsto \mathbb{R}^{N^{i,j} \times C}$ consists of a sequence of convolution, batch normalization, ReLu activation and pooling operations, to transform the input image to a feature representation. Then a *softmax* activation function is used to map the representation to a probability map. Here C is the number of classes, $N^{i,j}$ denotes the number of pixels of the down-sampled mask, (i, j) denotes the index of convolutional blocks. Given mask $y^{i,j} \in \mathbb{R}^{N^{i,j},C}$, the loss function used in the block $\mathrm{X}^{i,j}$ is the pixel-wise cross-entropy loss, which is defined as:

$$\mathcal{L}^{i,j}(y^{i,j}, \hat{y}^{i,j}, N^{i,j}) = -\frac{1}{N^{i,j}} \sum_{n=1}^{N^{i,j}} \sum_{c=1}^{C} y_{n,c}^{i,j} \log(\hat{y}_{n,c}^{i,j}), \tag{2}$$

where $y_{n,c}^{i,j}$ is the ground-truth label of a pixel and $\hat{y}_{n,c}^{i,j}$ is the probability of the pixel being classified as class c. Based on equation 2, the overall loss function for the deep supervised UNet^d is then defined as the weighted sum of the cross



(a) UNet^1 (b) UNet^2 (c) UNet^3 (d) UNet^4

(e) ADS_UNet (f) scSE

(g) Legend

Figure 2: The architecture of the proposed ADS_UNet. Each circular node in the graph represents a convolution block. Specifically, yellow nodes indicate that parameters are trainable during back-propagation, green nodes indicate that parameters are frozen. (a-d) UNets of varying depths. All of UNet^i are trained with the same dataset $\mathcal{X}$, but using different sample weight, W. (e) Ensemble architecture, ADS_UNet, which combines UNets of varying depths into one unified architecture for inference. (f) The scSE block. It contains *left branch* channel squeeze and spatial excitation block (sSE), and *right branch* spatial squeeze and channel excitation block (cSE).

entropy loss from each supervised block $X^{i,j}$:

$$\mathcal{L}_d = \sum_{i,j\geq 0}^{i+j=d} \eta^d_{i,j}\mathcal{L}^{i,j}(y^{i,j}, \hat{y}^{i,j}, N^{i,j}),\ \eta^d_{i,j} \geq 0,\ \sum_{i,j\geq 0}^{i+j=d} \eta^d_{i,j} = 1, \tag{3}$$

where $\eta^d_{i,j}$ is a weighting factor that assigned to the convolutional block $X^{i,j}$ to characterize the relative importance between blocks. d denotes the depth of the UNet. In contrast to the previous work [8, 35, 34] that use equal weights $\eta^d_{i,j} = \frac{1}{d+1}$, we initialize $\eta^d_{i,j}$ to $\frac{1}{d+1}$ and allow the $\eta^d_{i,j}$ to be trainable, and use the *softmax* function to normalize $\eta^d_{i,j}$ to guarantee $\sum \eta^d_{i,j} = 1$. However, the feature learning of a block will be restricted if its $\eta^d_{i,j}$ decreases to 0, during training. In order to guard against this competition exclusion phenomenon and encourage all supervised blocks to contribute to the segmentation, we add a constant $\frac{1}{d+1}$ to $\eta^d_{i,j}$ to raise its lower limit:

$$\tilde{\eta}^d_{i,j} = \frac{\eta^d_{i,j} + \frac{1}{d+1}}{\sum_{i,j\geq 0}^{i+j=d}(\eta^d_{i,j} + \frac{1}{d+1})} = \frac{\eta^d_{i,j}}{2} + \frac{1}{2(d+1)}, \tag{4}$$

Since $\lim\limits_{\eta^d_{i,j}\to 0} \frac{\eta^d_{i,j}}{2} + \frac{1}{2(d+1)} = \frac{1}{2(d+1)}$ and $\lim\limits_{\eta^d_{i,j}\to 1} \frac{\eta^d_{i,j}}{2} + \frac{1}{2(d+1)} = \frac{d+2}{2(d+1)}$, $\tilde{\eta}^d_{i,j}$ is bounded in $[\frac{1}{2(d+1)}, \frac{d+2}{2(d+1)}]$. Then equation (3) is re-written as follows to train each constitute model UNet^d:

$$\begin{gathered}\mathcal{L}_d = \sum_{i,j\geq 0}^{i+j=d} \tilde{\eta}^d_{i,j}\mathcal{L}^{i,j}(y^{i,j}, \hat{y}^{i,j}, N^{i,j}),\\ \frac{1}{2(d+1)} \leq \tilde{\eta}^d_{i,j} \leq \frac{d+2}{2(d+1)},\ \sum_{i,j\geq 0}^{i+j=d} \tilde{\eta}^d_{i,j} = 1.\end{gathered} \tag{5}$$

Once the UNet^d is trained, the final probability map generated by UNet^d is calculated by:

$$\hat{y}^d(x) = \sum_{i,j\geq 0}^{i+j=d} \tilde{\eta}^d_{i,j}\hat{y}^{i,j}(x), \tag{6}$$

with $\hat{y}^{i,j}(x)$ and $\eta^d_{i,j}$ defined in equations (1) and (4). $\hat{y}^d(x)$ denotes the combined prediction of model UNet^d. We conduct ablation studies in section 5.2 to show the benefits of imposing range constraint on $\tilde{\eta}^d_{i,j}$. Moreover, we demonstrate that generating the final prediction by using the weighted summation of multi-scale outputs yields better segmentation performance.

3.2. Stage-wise Additive Training

The stage-wise additive training process of the ADS_UNet is described in Algorithm 1 and visually illustrated in Figure 2. The main components of the iterative training procedure are 1) updating sample weights, 2) assigning weighting factors to base learners, and 3) freezing trained encoders while training decoders. We will elaborate on these as follows.

Firstly, given the training images $\mathcal{X}$=$\{x_1, ..., x_m\}$ and associated masks $\mathcal{Y}$=$\{y_1, ..., y_m\}$, we assign a weight w_k to each sample x_k. These weights are initialized to $w_k^1 = \frac{1}{m}$ (line 1 in Algorithm 1). Then, in the first iteration (d=1), the parameters of the encoder block (X^{0,0}) of the first base learner UNet^1 are initialized (line 2). In the first iteration of the sequential learning approach, parameters of the bottleneck node X^{1,0} and decoder nodes X^{0,1}) of the UNet^1 are initialized randomly (lines 4-6). Line 7 initializes the weighting factors $\tilde{\eta}^d_{i,j}$ of supervised blocks. The UNet^1 is then trained on all training samples with the same weight of $\frac{1}{m}$ (line 8). After the UNet^1 is trained, the training set will be used to evaluate it and to determine its error rate ϵ_1 (lines 9-11). In contrast to AdaBoost, we use mean Intersection over Union (mIoU) error (lines 10) to measure segmentation performance rather than using mis-classification rate. In detail, given one-hot mask $\mathbf{y}_{i,c}$=$[k_1, \cdots, k_n]$, k_j∈$\{0, 1\}$ for an image's c^{th} class and the corresponding one-hot prediction $\hat{\mathbf{y}}^d_{i,c}$=$[\hat{k}_1, \cdots, \hat{k}_n]$, $\hat{k}_j$∈$\{0, 1\}$ generated by UNet^d, the mIoU score s_i^d is calculated by:

$$s_i^d = \text{mIoU}(y_i, \hat{y}_i^d) = \frac{1}{C}\sum_{c=1}^{C}\frac{\mathbf{y}_{i,c}\cdot\hat{\mathbf{y}}^d_{i,c}}{\mathbf{y}_{i,c}\cdot\mathbf{y}_{i,c}+\hat{\mathbf{y}}^d_{i,c}\cdot\hat{\mathbf{y}}^d_{i,c}-\mathbf{y}_{i,c}\cdot\hat{\mathbf{y}}^d_{i,c}}, \tag{7}$$

where i is the index of training images, c is the index of class labels, d is the index of iteration and also denotes the depth of the constituent UNet. If the error rate ϵ_1 of the UNet^1 is less than 1-$\frac{1}{C}$ (line 12), then UNet^1 will be preserved for the ensemble, otherwise, it will be disregarded by setting its weighting factor to 0 (lines 18-19). In the case that $\epsilon_1 < 1 - \frac{1}{C}$, the equation shown in line 13 is used to calculate model weight α_d for the ensemble. So far we have obtained the first base learner UNet^1, and its weighting factor α_1.

We then update sample weights based on mIoU scores (line 14) for the training of the next iteration:

$$w_k^d = w_k^{d-1}e^{(1-s_k^{d-1})}, \quad k = 1, 2, ..., m, \tag{8a}$$

$$w_k^d \leftarrow \frac{w_k^d}{\sum_{i=1}^m w_i^d}, \quad k = 1, 2, ..., m, \tag{8b}$$

Algorithm 1: ADS_UNet. The $\tilde{\eta}^t_{i,j}$ term in line 9 is discussed in the context of equation (4); the UNet^t are described in Figure 2.

```
   Input: Number of class: $C$; Training images:
          $\mathcal{X} = \{x_1, ..., x_m\}$; Training masks:
          $\mathcal{Y} = \{y_1, ..., y_m\}$; Number of iteration: $T$.
1  $\boldsymbol{w}^1 = \{w_k^1 | w_k^1 = \frac{1}{m}, k = 1, 2, ..., m\}$ ;
2  Initialising convolutional block $X^{0,0}$ ;
3  for $d = 1, 2, ..., T$ do
4      for $j = 0, 1, ..., d$ do
5          Initializing convolutional block $X^{d-j,j}$;
6      end
7      $\tilde{\eta}^d_{i,j} = \frac{1}{d+1}, \ i, j \geq 0, \ i + j = d$ ;
8      Train UNet$^d(\mathcal{X}, \mathcal{Y}, \boldsymbol{w}^d)$ ;
9      $\hat{y}_k^d = \sum_{i,j\geq 0}^{i+j=d} \tilde{\eta}^d_{i,j}\hat{y}^{i,j}(x_k)$ ;   // see equation (6)
10     $s_k^d = \text{mIoU}(\hat{y}_k^d, y_k)$ ;             // see equation (7)
11     $\epsilon_d = \sum_{k=1}^m w_k^d(1 - s_k^d)$ ;
12     if $\epsilon_d < 1 - \frac{1}{C}$ then
13         $\alpha_d = \frac{1}{2}\ln(\frac{1-\epsilon_d}{\epsilon_d}) + \ln(C - 1)$ ;
14         Updating sample weight $w_k^d$ using
             equation (8a) and (8b);
15         for $j = 0, 1, ..., d$ do
16             Freeze convolution block $X^{j,0}$ ;
17         end
18     else
19         $\alpha_d = 0$ ;
20     end
21 end
   Output: ADS_UNet = $\arg\max_C(\sum_{d=1}^T \alpha_d \hat{y}^d)$
```

Equation (8a) assigns greater weight to images that cannot be accurately segmented by UNet^{d-1}, encouraging UNet^d to focus more on their segmentation. Equation (8b) normalizes sample weights to guarantee that $\sum_{k=1}^m w_k^d = 1$.

Before the start of the second iteration, it is necessary to freeze the encoder nodes (X^{0,0} and X^{1,0}) of the UNet^1 (lines 15-17), otherwise the process of training UNet^2 would update UNet^1's encoder parameters as well; this would reduce the learned association between the encoder and decoder paths of UNet^1, and furthermore, {UNet$^{\text{d}}$ | d ≥ 2} would acquire correlated features. The code block in lines 4-20 are run for T iterations to obtain T base learners, as well as a weighting factor α_d for each of them. Note that all parameters of UNet^1 are trained as a whole but UNet^2 reuses encoder weights of UNet^1 and only its decoder parameters are trained (if $\epsilon_1 \leq 1 - \frac{1}{C}$). The purpose of using the updated sample weights w_k^2 to train UNet^2 is to force the decoder layers of UNet^2 (because the connection between X^{i,0} and X$^{i+1,0}$ only involves max pooling) to learn features dissimilar to those learned by UNet^1.

3.3. Feature Re-calibration

The concurrent spatial and channel Squeeze & Excitation (scSE) block [28] is used to re-calibrate feature maps learned from encoder blocks of UNet^d, to better adapt to features learned from decoder blocks of deeper UNet^{d+a}, $a \geq 1$ layers. For example, features learned by the encoder block $X^{0,0}$ and the decoder block $X^{1,0}$ of the UNet^1 can cooperate well to perform segmentation since their weights are updated in a coordinated end-to-end back-propagation process. In UNet^2, however, features produced by $X^{1,1}$ and $X^{1,0}$ (in the same depth) can be very different, since the gradient flow is truncated between block X^{1,0} and $X^{2,0}$. Therefore, although features produced by $X^{0,0}$ used to cooperate well with that of $X^{1,0}$, it is not guaranteed that it can adapt well to that of $X^{1,1}$. Based on this analysis, the scSE block is used to re-weight features before concatenating. We evaluate the role of feature re-calibration in section 4.4. The detailed process of scSE is illustrated in Figure 2(f).

Given an input feature map $\boldsymbol{U} \in \mathbb{R}^{H \times W \times C_{in}}$, The channel squeeze operation generates a projection matrix $\boldsymbol{q} \in \mathbb{R}^{H \times W}$ with matrix element $q_{i,j} = \sum_{k=1}^{C_{in}} W_k U_{i,j,k}$. This matrix is then re-scaled by passing it through a sigmoid function $\sigma(\cdot)$, which re-weights the input feature map $\boldsymbol{U}$ spatially,

$$\hat{U}^{sSE}_{i,j,k} = \sigma(q_{i,j}) U_{i,j,k}, \tag{9}$$

The global average pooling of the feature map over all pixels produces $\boldsymbol{z}$ with components z_k,

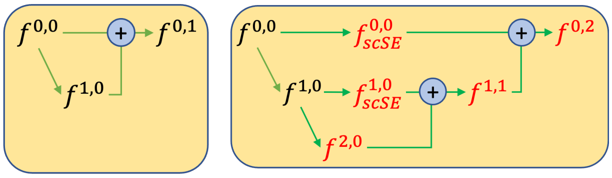
$$z_k = \frac{1}{H \times W} \sum_{i}^{H} \sum_{j}^{W} U_{i,j,k}, \quad k = 1, 2, \ldots, C \tag{10}$$

This vector, $\boldsymbol{z}$, is transformed to $\hat{\boldsymbol{z}} = \boldsymbol{W_1}(\delta(\boldsymbol{W_2 z}))$, with $\boldsymbol{W}_1 \in \mathbb{R}^{C \times \frac{C}{2}}$, $\boldsymbol{W}_2 \in \mathbb{R}^{\frac{C}{2} \times C}$ being weights of two fully connected layers and $\delta(\cdot)$ is the ReLU function. The range of the activations of $\hat{\boldsymbol{z}}$ are brought to the interval [0, 1], by passing it through a sigmoid function $\sigma(\cdot)$. The input feature map $\boldsymbol{U}$ is then re-weighted by the re-scaled vector, with its k^{th} channel

$$\hat{\boldsymbol{U}}^{cSE}_{k} = \sigma(\hat{z}_k) \boldsymbol{U}_k, \boldsymbol{U}_k \in \mathbb{R}^{H \times W}, \tag{11}$$

In the channel re-calibrated feature maps $\hat{\boldsymbol{U}}^{cSE}$, the channels that are less important are suppressed and the important ones are emphasized. Finally, after concurrent spatial and channel squeeze and excitation (scSE), a location (i, j, c) of the input feature map $\boldsymbol{U}$ is then given higher activation when it gets high importance from both, channel re-scaling and spatial re-scaling.

It is worth mentioning that sample re-weighting and feature re-calibration are utilized in the ADS_UNet for different purposes and are not in conflict with each other. Taking UNet^1 and UNet^2 as an example, sample re-weighting aims at achieving feature diversity between final outputs ($f^{0,1}$ and $f^{0,2}$ in Figure 3) of two base learners, so that the ensemble of UNet^1 and UNet^2 can compensate for each other's false prediction, thus leading to better segmentation.



(a) Info flow of UNet^1

(b) Info flow of UNet^2

Figure 3: Information flow diagram of base learners. $f^{i,j}$ denotes the output feature maps of block X^{i,j}, $f^{i,j}_{scSE}$ denotes the re-calibrated version of $f^{i,j}$, a circle with plus sign denotes feature map concatenation operation, connecting lines with an arrow denote the flow of features. Features in red represent that they can be updated during the training of UNet^2 (after UNet^1 is trained), while others are fixed.

While feature re-calibration aims at feature alignment of two concatenated features (re-calibrate $f^{1,0} \rightarrow f^{1,0}_{scSE}$ to be aligned with $f^{2,0}$, and re-calibrate $f^{0,0} \rightarrow f^{0,0}_{scSE}$ to be aligned with $f^{1,1}$, see figure Figure 3(b)).

3.4. Difference between ADS_UNet and UNet++

In section 3.1- 3.3, we introduced the components and training scheme of the ADS_UNet. For inference, the final probability map for an image $x \in \mathbb{R}^{C \times H \times W}$ can be generated by weighted average:

$$\hat{y}(x) = \text{ADS\_UNet}(x) = \sum_{d=1}^{T} \alpha_d \hat{y}^d(x) \tag{12}$$

Here C is the number of classes. $\hat{y}^d(x)$ is the probability map generated by UNet^d, as defined in equation (6) and shown in Figure 2(e).

The proposed ensemble structure differs from the UNet++ in two ways: one differs in the training method, and the other in the way decisions are made and incorporated into learning. 1) *Embedded vs. isolated training*. The UNet++ is trained in an embedded training fashion where the full UNet++ model is trained as a whole, with deep supervision on the last decoder block $X^{0,i}$ of branch i. In the ADS_UNet, however, each UNet^d is trained by isolating features acquired by the deeper encoder and decoder blocks. Moreover, deep supervision is added to each decoder block of each branch by down-scaling the label masks, rather than solely on the last decoder node of each branch. 2) *Average vs. weighted average voting*. In the ensemble mode of the UNet++, the segmentation results from all branches are collected and then averaged to produce the final prediction. UNet++$(x) = \arg\max_{c \in C}(\frac{1}{T} \sum_{d=1}^{T} \text{UNet}^d(x))$, with UNet$^d(x) = \hat{y}^{0,d}$. T is the number of branches of the UNet++. However, the ADS_UNet takes performance-weighted combinations of the component UNets to create the final segmentation map: ADS_UNet$(x) = \arg\max_{c \in C}(\sum_{d=1}^{T} \alpha_d \hat{y}^d)$, with $\hat{y}^d$ =UNet$^d(x)$ is calculated from equation (6). α_d reflects the importance of the UNet^d in the ensemble.

4. Experiments and Results

Three histopathology datasets are used to check the effectiveness of the proposed methods.

4.1. Datasets

CRAG dataset. The colorectal adenocarcinoma gland (CRAG) dataset was originally used in [2]; it contains a total of 213 Hematoxylin and Eosin images taken from 38 WSIs scanned with an Omnyx VL120 scanner under 20× objective magnification). All images are mostly of size 1512×1516 pixels. The dataset is split into 173 training images and 40 test images. We resize each image to a resolution of 1024×1024 and then crop it into four patches with a resolution of 512×512 for all our experiments.

BCSS dataset. The Breast Cancer Semantic Segmentation dataset [1] consists of 151 H&E stained whole-slide images and ground truth masks corresponding to 151 histologically confirmed breast cancer cases. A representative region of interest (ROI) was selected within each slide by the study coordinator, a medical doctor, and approved by a senior pathologist. ROIs were selected to be representative of predominant region classes and textures within each slide. Tissue types of the BCSS dataset consists of 5 classes (i)tumour, (ii)stroma, (iii)inflammatory infiltration, (iv)necrosis and (v)others. We set aside slides from 7 institutions to create our test set and used the remaining images for training. Shift and crop data augmentation, random horizontal and vertical flip were adopted to enrich training samples. Finally, 3154 and 1222 pixel tiles of size 512×512 were cropped for training and testing, respectively. Weighted categorical cross-entropy loss was used to mitigate class imbalance, with the weight associated with each class determined by $W_c = 1 - \frac{N_c}{N}$, where N is the number of pixels in the training dataset and N_c is the number of pixels belonging to class c.

MoNuSeg dataset. The MoNuSeg dataset [19] is a multi-organ nucleus segmentation dataset. The training set includes 30 images of size 1000×1000 from 4 different organs (lung, prostate, kidney, and breast). The test set contains 14 images with more than 7000 nucleus boundary annotations. A 400× 400 window slides through the images with a stride of 200 pixels to separate each image into 16 tiles for training and testing.

4.2. Baselines and Implementation

Since our work is mainly based on UNet, UNet^e, and UNet++, we re-implement these three models, as well as CENet, to compare with our proposed methods. We also compare the proposed ADS_UNet with two transformer-based UNet variants, HyLT [22] and MedFormer [12], using the implementation provided by the authors. For a fair comparison, the configuration of the outermost convolutional blocks ($X^{i,0}$, $i \in \{0, 1, 2, 3\}$ and $X^{i,j}$, $i, j \geq 0$, $i+j = 4$) of all compared methods are exactly the same as in the original UNet (both the number and size of filters). All inner decoder nodes of UNet^e, UNet++ and ADS_UNet are also exactly the same, and all models have the same hyper-parameters. It is noted that scSE block is not used in UNet, UNet^e, UNet++ and CENet, but it is used in the skip-connections of ADS_UNet. The models are implemented in Pytorch [26] and trained on one NVIDIA RTX 8000 GPU using the Adam optimizer [17] with weight decay of 10^{-7} and learning rate initialized at 0.001 and then changed according to the 1cycle learning rate policy [29]. The cross-entropy loss is used to train all compared models, and ADS_UNet is trained with the linear combination of loss functions using equation (5). On models with a depth of 4, the number of filters at each level are 64, 128, 256, 512, and 1024, on the CRAG and the BCSS dataset. This setting is consistent with the standard UNet [27]. However, we change the number of filters to 16, 32, 64, 128, 256 for all models, when trained on the MoNuSeg dataset, as our experimental results show that increasing the number of filters leads to inferior performance. The colour normalization method proposed in [31] is used to remove stain color variation, before training. We also compare our methods with the state-of-the-art nnU-Net [15]. Note that the nnUNet automatically decides the depth of the architecture based on its characterization of properties of the datasets. In our experiments, the nnUNet generated for the BCSS dataset and the CRAG dataset is of a depth of 7, while it is 6 for the MoNuSeg dataset. The officially released nnUNet source code is used in our experiments.

4.3. Results

Some image patches and their corresponding segmentation maps are depicted in Appendix **??** Figure 4.3. Table 1 summarizes the segmentation performance achieved by all compared methods. The performance of the baseline method (VGG-16, FCN-8) used in [1] is also included for comparison. The number of parameters and computational efficiency of various UNet variants is also included in the table. Statistical analysis of the results (Table 1) is performed with the help of the Autorank package [14]. The non-parametric Friedman test and the *post hoc* Nemenyi test [6] at the significance level $\alpha = 0.05$ are applied to determine if there are significant differences between the predictions generated by models and to find out which differences are significant. As shown in Table 4.2, out of 21 pairwise comparisons on 3 datasets, 17 models show significant differences to the ADS_UNet.

Among the different networks evaluated, the ADS_UNet outperforms all of the other state-of-the-art approaches on CRAG and BCSS dataset, and achieves competitive performance on MoNuSeg dataset. UNet++ achieves 1.17, 0.44 and 1.17 higher mIoU scores than UNet by performing 2.35 times more computation and consuming 1.77 times more GPU memory. In contrast, ADS UNet performs the best, but requires at most 59.51% of the GPU memory and 42.55% of the floating-point operations required by UNet++ for training. CENet surpasses ADS_UNet on the MoNuSeg dataset, but at a cost of requiring 2.15 times of more computation and 1.19 times of more GPU memory. nnUNet consumes the least amount of GPU memory and the number of operations, but at the cost of a small decrease in segmentation accuracy.

Table 1
The comparison of models in terms of number of parameters, computational complexity (measured by FLOPs), required GPU memory, training time (seconds) per epoch, and segmentation performance (measured by mIoU). The FLOPs and GPU consumption are computed with 512×512 inputs. The GPU memory consumption is measured by *nvidia-smi* command (batch size=2). In ADS_UNet, base learners require a different amount of GPU memory, since they vary in depth and the number of parameters (The total number of trainable parameters of the ADS_UNet is 35.41 million). The mIoU score of the FCN-8 is computed from the confusion matrix provided in the supplementary material of [1].

Net	Year	Params(M)	FLOPs(G)	GPU(GB)	Time (s)	CRAG	BCSS	MoNuSeg
FCN-8 [1]	2015	–	–	–	–	–	60.55	–
UNet [27]	2015	31.04	218.9	5.54	771	86.87	59.41	80.12
UNet^e [34]	2019	34.92	445.2	9.80	1071	86.75	58.73	81.08
UNet++ [34]	2019	36.17	514.8	9.31	1303	88.04	59.85	81.29
nnUNet [15]	2021	41.27	65.6	2.92	442	88.45	60.96	80.79
CENet [33]	2022	35.17	471.55	5.99	713	86.85	59.45	81.67
HyLT [22]	2022	42.20	329.11	16.06	1500	87.70	60.45	81.69
MedFormer [12]	2022	99.54	325.76	15.48	1337	87.92	60.26	**81.84**
ADS_UNet	2023	0.41→1.63→ 6.65→26.72	62.61→114.80→ 166.93→219.04	4.00→ 4.92→ 5.40→5.71	453	**89.04**	**61.05**	81.43

Table 2
The difference of the mean rank between ADS_UNet and compared methods. For each tested dataset, the differences in performance are statistically significant if the difference in the mean rank is greater than the critical distance. Values lower than the critical distance are highlighted in bold, indicating there is not significant differences between those models and ADS_UNet.

Dataset	Critical Distance	UNet	UNet^e	UNet++	nnUNet	CENet	HyLT	MedFormer
CRAG	0.830	1.941	2.109	1.813	**0.316**	2.025	0.831	0.972
BCSS	0.300	0.474	**0.082**	0.336	0.426	**0.239**	0.524	0.801
Kumar	0.700	1.662	2.387	1.978	2.32	1.622	**0.453**	2.218

The design choices (pipeline fingerprint) of nnUNet are not fixed across datasets, but are configured on the fly according to the 'data fingerprint' (dataset properties such as image size, image spacing, number of classes, etc.). The data-dependent 'rule-based parameters' (patch size, batch size, network depth, etc.) of the pipeline is determined by a set of heuristic rules that models parameter inter-dependencies. As shown in Table 1, nnUNet outperforms all models on CRAG and BCSS datasets, except for ADS_UNet. But it demonstrates inferior performance on MoNuSeg dataset. This can be explained by the characteristics of datasets and the receptive field size of models. Firstly, the nnUNet is deeper (the depth of the nnUNet is 6 or 7, as mentioned in section 4.2), which means that the convolutional kernels of the bottleneck layer (the deepest encoder layer) have a larger receptive field, enabling the model to extract information from a larger region. This is especially beneficial when the task is to recognize large objects, e.g. tissue types or glands, since a larger receptive field can cover the whole object. In models with a depth of 4, the size of the receptive field of the bottleneck layer is limited. This difference in the depth of models may explain why nnUNet outperforms shallower models when trained for segmenting tissues and glands. In contrast, the size of the cell nucleus in the MoNuSeg dataset is much smaller than tissue and gland. The receptive field of the bottleneck layer of shallow models is large enough to capture the entire nucleus. Further increasing the depth of the network compresses features leading to information loss rather than enhancing the features learnt. The nnUNet improves segmentation performance by enlarging receptive field size, while ADS_UNet achieves so by ensembling. Image and mask patches presented in Figure 5 show the size difference of target objects between the nucleus segmentation dataset and the gland segmentation dataset.

Both transformer-based architectures, HyLT and MedFormer, demonstrate inferior performance on the CRAG dataset, but achieve competitive performance on the BCSS dataset, and outperform the ADS UNet on the MoNuSeg datasets. However, it is worth noting that the HyLT and the MedFormer have 1.19 times and 2.81 times parameters than the ADS_UNet does and require 2.81 more times and 2.71 times of more GPU memory than the ADS_UNet does for training. The high demand for GPU memory in the HyLT and MedFormer is not surprising, as the attention blocks introduce extra intermediate feature maps that should be kept in the GPU memory for back-propagation.

The amount of computation (FLOPs) and GPU memory requirement are the main constraints of training speed. Among all compared methods, ADS_UNet shows a clear advantage in training speed, because the lower GPU memory requirement of ADS_UNet allows us to use a larger batch size for faster training. The training speed of nnUNet is close to ADS_UNet, due to the same reason. The transformer-based models (MedFormer and HyLT) are the slowest ones

Table 3
Comparison between the original UNet (without deep supervision) and UNet$^{\uparrow}$/UNet$^{\downarrow}$ with deep supervision using up-sampled feature maps/average pooled masks.

Net	Params (M)	FLOPs (G)	GPU (GB)	CRAG	BCSS	MoNuSeg
UNet	**31.04**	**218.9**	**5.54**	86.87	**59.41**	80.12
UNet$^{\uparrow}$	31.17	260.27	14.03	**88.84**	58.40	**81.40**
UNet$^{\downarrow}$	31.06	219.08	5.61	88.33	**59.41**	81.24

Table 4
Ablation study. Performance measured by mIoU (highest score highlighted in bold). "SCSE" denotes spatial and channel squeeze & excitation used in skip-connections. "Re-weight" denote training sample re-weighting. "ens (avg)"/"ens (α)" denote that segmentation results from all branches are collected and then averaged, or summed by α weights.

Model Name	Deep supervision	SCSE	Re-weight	UNet^1	UNet^2	UNet^3	UNet^4	ens(avg)	ens(α)
model_0	✗	✗	✗	43.41	53.66	57.59	58.40	57.92	58.22
model_1	✓	✗	✗	43.89	53.14	57.21	58.07	58.79	58.82
model_2	✗	✓	✗	46.67	55.87	58.70	59.92	59.92	60.20
model_3	✗	✗	✓	43.58	53.93	57.34	58.43	58.59	58.82
model_4	✗	✓	✓	**47.97**	56.47	59.51	59.55	60.51	60.76
model_5	✓	✗	✓	44.04	53.32	55.87	57.99	58.23	58.25
model_6	✓	✓	✗	46.93	56.60	58.62	60.26	60.57	60.63
ADS_UNet	✓	✓	✓	46.93	**56.91**	**60.11**	**60.26**	**61.04**	**61.05**

since they have the highest GPU memory demand and relatively high computation cost.

4.4. Ablation Studies

4.4.1. *Down-sampling masks vs. up-sampling feature maps*

We build UNet$^{\uparrow}$ and UNet$^{\downarrow}$ as UNet's counterparts to demonstrate the advantage of using down-sampled masks for deep supervision. The subscript $^{\uparrow}$ and $^{\downarrow}$ denotes that either feature maps of the UNet are bilinear interpolated to fit the size of the original mask or the masks are down-sampled to fit the size of feature maps. As shown in Table 3, UNet$^{\downarrow}$ with average pooled masks outperforms UNet by 1.46 and 1.12 mIoU on the CRAG and MoNuSeg datasets. This is achieved with only 0.06% more parameters, 1.26% more GPU memory consumption and 0.08% more FLOPs. This small increase comes from a 1×1 convolution layer appended to supervised blocks. We attribute this performance gain to back-propagation through deep layers enforcing shallow layers to learn discriminative features. UNet$^{\uparrow}$ yields 0.51 and 0.16 higher mIoU than UNet$^{\downarrow}$ on CRAG and MoNuSeg dataset, but performs worse than UNet by 1.01 points on BCSS dataset. The 18.80% more computation of the UNet$^{\uparrow}$ (compared with UNet$^{\downarrow}$) originates from bilinear interpolation operations when up-sampling feature maps. The GPU memory required in the training process of UNet$^{\uparrow}$ is 2.50 times that of UNet$^{\downarrow}$. The reason is that during back-propagation the output of all layers is cached during forward propagation, and the size of the feature map of the supervision layer in UNet$^{\uparrow}$ is 4 to 256 times the size of the corresponding one in UNet$^{\downarrow}$. Therefore, beyond a small performance improvement, UNet$^{\downarrow}$ saves more than 1.50× GPU consumption thus enabling us to use a larger batch size and save training time.

4.4.2. *Tracing the origin of the performance gain of ADS_UNet.*

To gain insight into the reason why ADS_UNet demonstrate superior performance on segmentation, we construct eight models and evaluate them on the BCSS dataset, with each of them being a combination of deep supervision, SCSE feature re-calibration blocks and sample re-weighting. The configuration of models, the performance of each constitute UNet^d and their ensemble performance is summarized in Table 4. To see whether weighted average voting of base learners is better than simple average voting or not, we also compare these two ensemble strategies and report results in the Table.

As seen in Table 4, when compared with model_0 (none of three components is used), model_1, model_2 and model_3 demonstrate the effectiveness of incorporating deep supervision, SCSE feature re-calibration and sample re-weighting into the training of each constitute UNet^d, respectively. Moreover, all constitute UNet^{d}s of model_2 (with SCSE) surpass the counterparts of model_0, model_1 and model_3. This supports the claim we made in section 3.3, i.e., features from the encoder block of UNet^{d-1} should be re-calibrated before concatenating with features from the decoder block of UNet^d.

Removing deep supervision from the ADS_UNet drops the mIoU score by 0.29 points (compared with model_4). Further analysis is provided in section 5.2 to reveal the

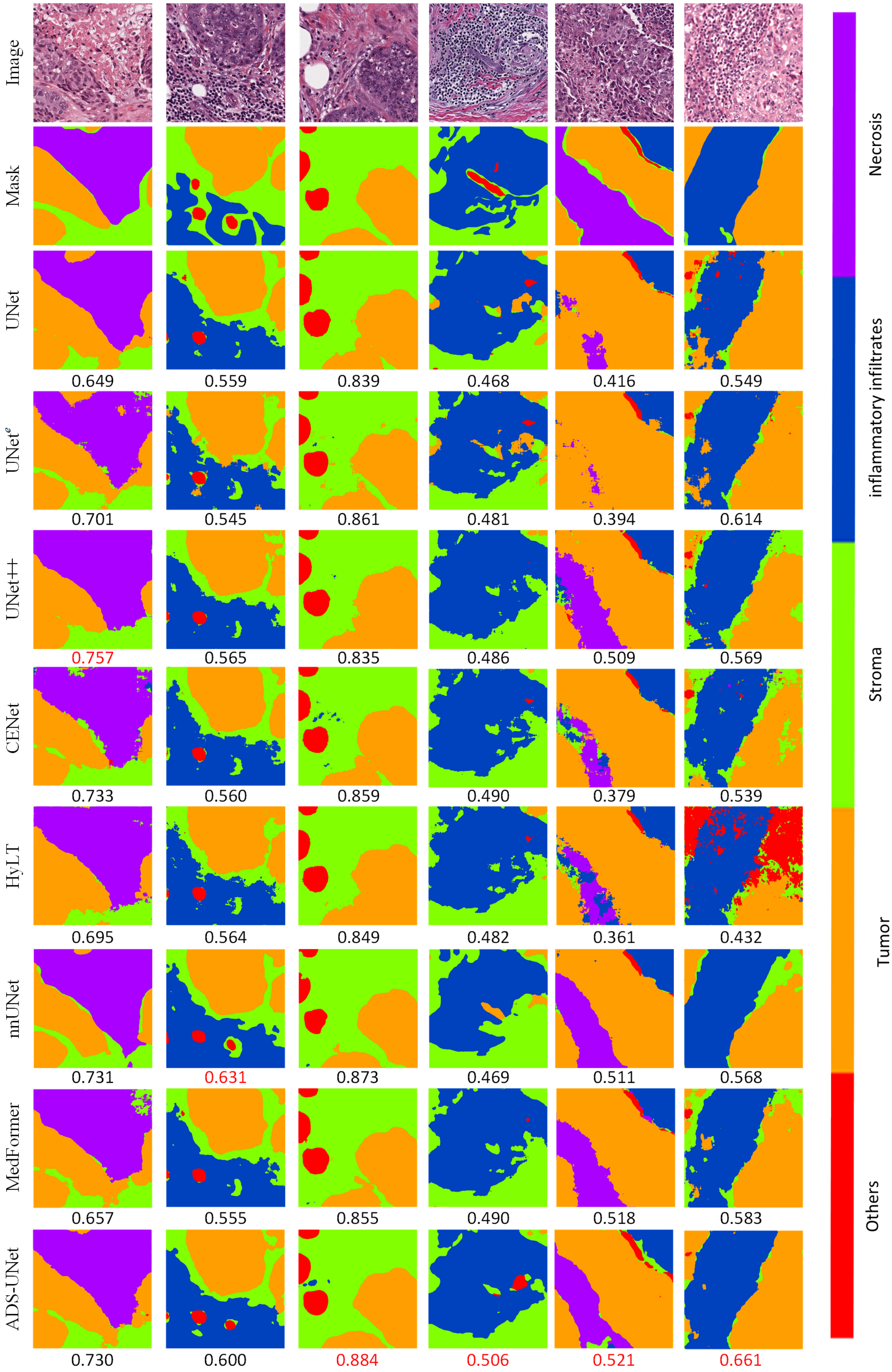


Figure 4: Visual comparison of segmentation maps. The mIoU score of each prediction is reported below the prediction.

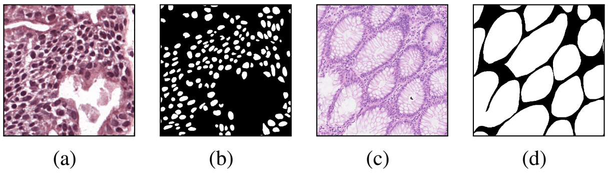


Figure 5: (a)-(b) Image-mask patch from the MoNuSeg dataset (nucleus segmentation). (c)-(d) Image-mask patch from the CRAG dataset (gland segmentation). All patches are of size 512×512.

Table 5
The proportion of incorrect labels in different scaled masks. The ($X^{i,j}$) under the down-scale factor indicates which layers the mask down-sampled by this down-scale factor is used to supervise.

↓ Data	2 $X^{1,0}$, $X^{1,3}$	4 $X^{2,0}$, $X^{2,2}$	8 $X^{3,0}$, $X^{3,1}$	16 $X^{4,0}$
CRAG	1.02	2.78	5.72	9.94
BCSS	1.51	4.32	10.04	19.75
MoNuSeg	5.49	16.21	36.69	60.05

reason why introducing explicit deep supervision leads to better performance.

In model_4, model_5 and model_6, we either remove deep supervision or SCSE block or sample re-weight from the ADS_UNet, respectively, to show the importance of each component in the composition of the ADS_UNet. As seen in Table 4, removing any one of them would lead to lower segmentation performance.

The experiment conducted on model_5 demonstrates that truncating the gradient flow between encoder blocks of UNet^d and decoder blocks of UNet^{d+1} to the detriment of the final segmentation performance (compared with the ADS_UNet). By introducing feature re-calibration in skip-connections, features learnt in encoder blocks are re-weighted to adapt to the ones of decoder blocks, thereby leading to better performance. The importance of SCSE feature re-calibration is also reflected in comparisons of model_0 vs. model_2 (1.98↑), model_1 vs. model_6 (1.81↑), and models_3 vs. model_4 (1.94↑).

In terms of sample re-weighting, the ensemble (ens(α)) of ADS_UNet surpasses the one of model_6 by 0.42 points. We attribute this to sample weight updating, which allows UNet^d to pay more attention to images which are hard to be segmented by UNet^{d-1}. The benefits of sample re-weighting is also reflected in comparisons of model_0 vs. model_3 (0.6↑) and model_2 vs. model_4 (0.56↑).

When comparing ensemble strategies, we find both average voting and weighted voting improve segmentation performance compared with UNet^4; but the improvement due to weighted voting is higher than from average voting. Moreover, the segmentation performance of the model_6 with α weighting is better than that of average weighting, although training samples are not re-weighted in its iterative training process. This, too, supports the view that integrating multiple models by weighting each as per its segmenting ability improves the overall performance of the ensemble.

5. Analysis

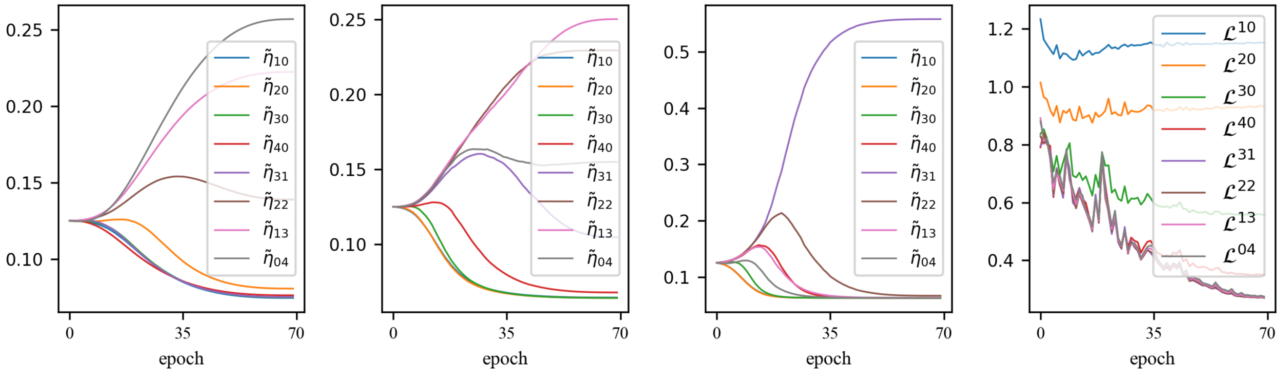
5.1. Incorrect labeling information can be evaded by adjusting $\tilde{\eta}^d_{i,j}$.

It is true that down-sampling the ground-truth mask eliminates small objects and leads to incorrect labels for pixels located on the class boundaries. We quantify the ratio of incorrect labels of down-sampled masks and present the statistics in Table 5. It can be observed that the proportion of incorrect labels rises as the down-scaling factor becomes larger. Incorrect labels in the ×16 down-scaled mask in the CRAG and BCSS datasets account for 9.94% and 19.75% of the total labels, respectively. This figure soars up to 60.05% in the MoNuSeg dataset. However, it is noteworthy that when these reduced masks are used to supervise the training of layers, there is a trainable weight $\tilde{\eta}^d_{i,j}$ (defined in equation (4)) that dynamically adjusts the strength of each layer being supervised. Figure 6(a)-6(c) shows how the network adjusts $\tilde{\eta}_{i,j}$ during training to assign weightings to layers and scales that contribute most to the segmentation task. As seen, at the end of training, the largest $\tilde{\eta}_{i,j}$ values of the MoNuSeg, CRAG and BCSS datasets come from $\tilde{\eta}_{0,4}$, $\tilde{\eta}_{1,3}$ and $\tilde{\eta}_{3,1}$, respectively. That means the UNet$^{\downarrow}$ benefits most from the original mask and the mask down-scaled by the factor of 2, 8, when trained on the MoNuSeg, CRAG and BCSS dataset. The ×2 and ×8 down-scaled masks carry 1.02% and 10.04% incorrect label information, respectively. Therefore, even though the down-scaled masks introduce wrong labelling information, the UNet$^{\downarrow}$ is able to evade this wrong information to a certain extent and puts attention on the informative mask by adjusting $\tilde{\eta}_{i,j}$. Despite the (apparently significant) labelling errors introduced by down-sampling, the overall result (as shown in Table 3) is not adversely affected.

5.2. Deep Supervision in UNet$^{\downarrow}$ and ADS_UNet

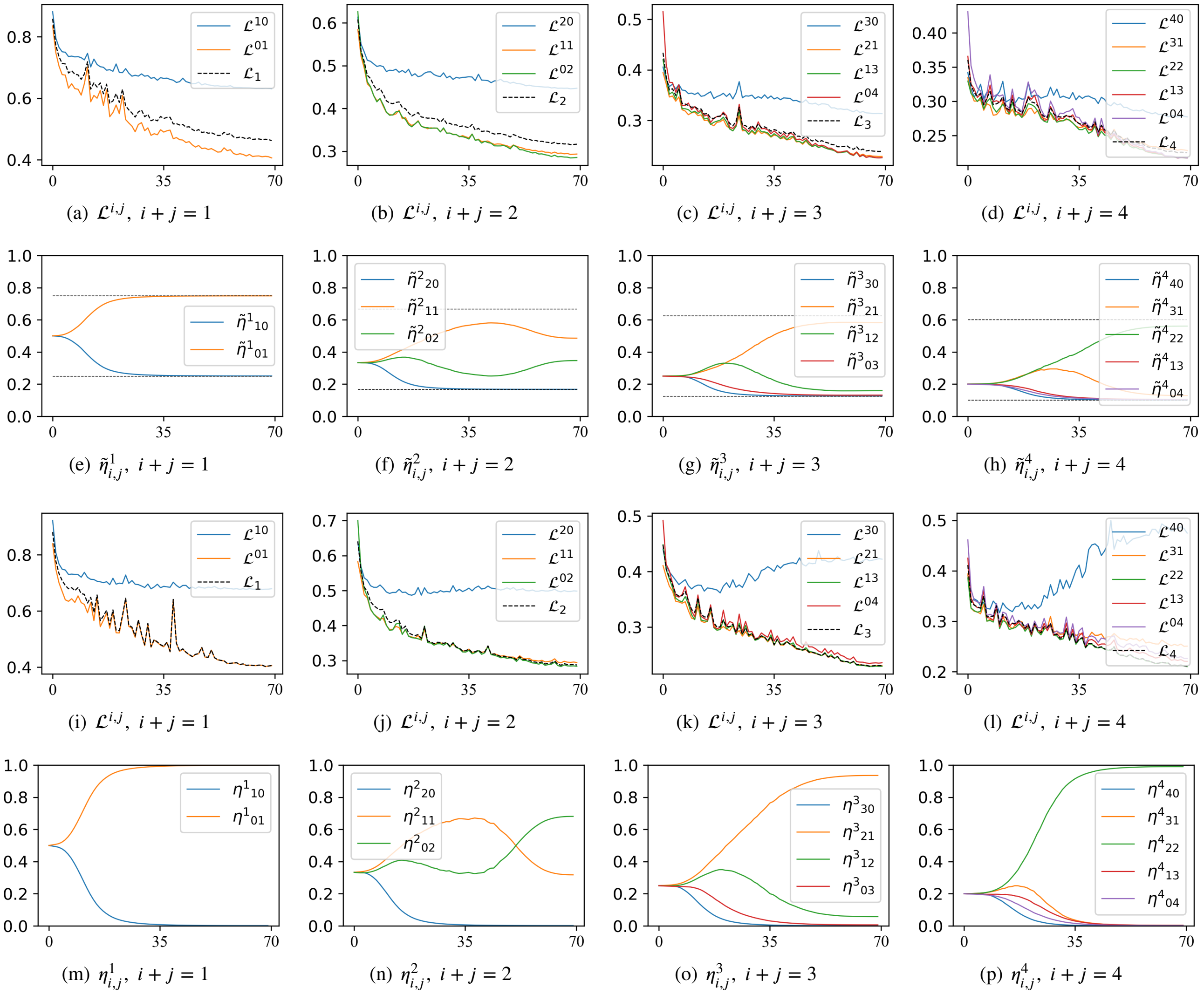
5.2.1. Different layers contribute differently at different time stamps.

In UNet^e and UNet++, all losses have the same weight in the back-propagation process, while in UNet$^{\downarrow}$ and ADS_UNet, $\tilde{\eta}_{i,j}$ is trainable. The purpose of this design is to check whether all layers in the summand of the training loss in equation (5) contribute equally. Taking Figure 6(c) as an example, the importance of decoder nodes $X^{3,1}$ and $X^{2,2}$ is ranked in the top two. This means features learned by these 2 layers contribute more than others, with changes in their importance throughout the training process. From the perspective of back-propagation, this means that parameters of layers which have larger $\eta_{i,j}$ values, will have relatively large changes when they are updated using gradient descent. This fact, therefore, indicates the importance of the features derived at that length scale to the separability of texture labels. The segmentation performance achieved by layer $X^{3,1}$, $X^{2,2}$ and $X^{0,4}$(the last layer) are: 59.57%, 59.55% and



(a) $\tilde{\eta}_{i,j}$ values (MoNuSeg). (b) $\tilde{\eta}_{i,j}$ values (CRAG). (c) $\tilde{\eta}_{i,j}$ values (BCSS). (d) Training losses (BCSS).

Figure 6: (a)-(c) figures show how $\eta^d_{i,j}$ changes when the UNet$^{\downarrow}$ is trained on the MoNuSeg, CRAG and BCSS datasets. The changing of $\eta_{i,j}$ varies from dataset to dataset. (e) The training losses of supervision layers (trained on the BCSS dataset).



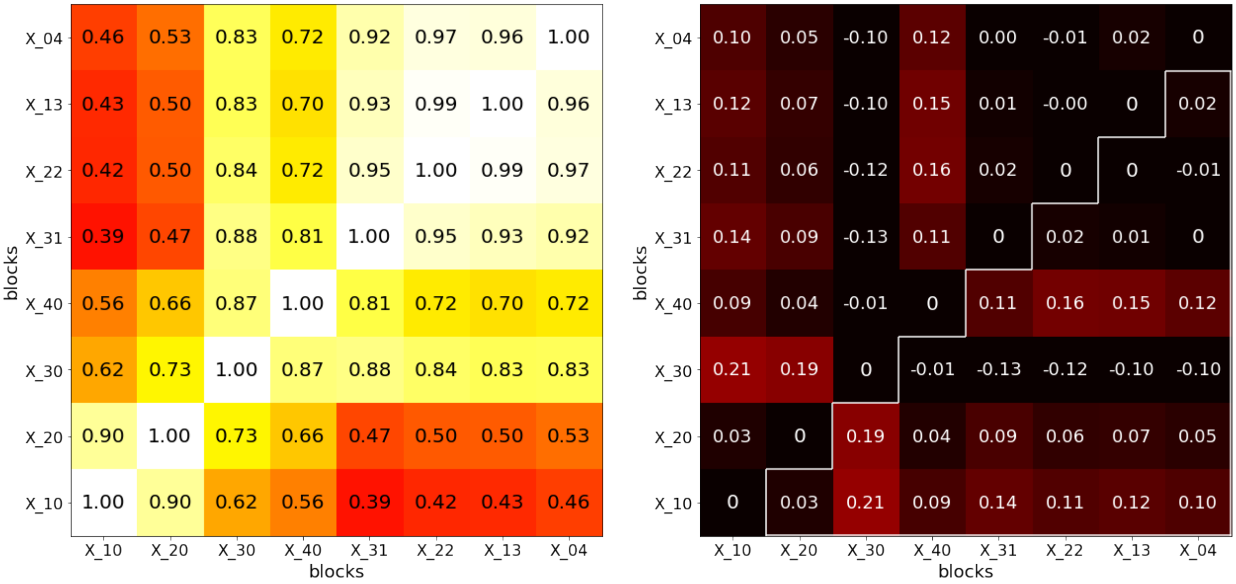
(a) $\mathcal{L}^{i,j}$, $i+j=1$ (b) $\mathcal{L}^{i,j}$, $i+j=2$ (c) $\mathcal{L}^{i,j}$, $i+j=3$ (d) $\mathcal{L}^{i,j}$, $i+j=4$

(e) $\tilde{\eta}^1_{i,j}$, $i+j=1$ (f) $\tilde{\eta}^2_{i,j}$, $i+j=2$ (g) $\tilde{\eta}^3_{i,j}$, $i+j=3$ (h) $\tilde{\eta}^4_{i,j}$, $i+j=4$

(i) $\mathcal{L}^{i,j}$, $i+j=1$ (j) $\mathcal{L}^{i,j}$, $i+j=2$ (k) $\mathcal{L}^{i,j}$, $i+j=3$ (l) $\mathcal{L}^{i,j}$, $i+j=4$

(m) $\eta^1_{i,j}$, $i+j=1$ (n) $\eta^2_{i,j}$, $i+j=2$ (o) $\eta^3_{i,j}$, $i+j=3$ (p) $\eta^4_{i,j}$, $i+j=4$

Figure 7: (a)-(d) Cross-entropy losses of supervision layers during the UNet^d training process (Equation (4) is imposed to constraint the range of $\tilde{\eta}^d_{i,j}$). $\mathcal{L}_d$ is calculated from equation (5). (e)-(h) The corresponding weights of supervision layers. $\tilde{\eta}^d_{ij}$ reflects the importance of node X^{i,j} while computing the overall loss. (i)-(p) The loss and the $\eta^d_{i,j}$ values of supervision layers of UNet^d, in which $\eta^d_{i,j}$ is trained without constraints. $\mathcal{L}_d$ shown in (i)-(l) is calculated from equation (3). For all plots, the x-axis indexes the training epoch. These plots are based on the BCSS dataset.



(a) UNet

(b) diff(UNet$^{\downarrow}$, UNet)

Figure 8: (a) Feature similarity of layers for UNet. (b) The difference in feature similarity of layers between UNet$^{\downarrow}$ and UNet. In (a), each entry shows the CKA similarity between the two layers. In (b), we calculate the feature similarity matrix for UNet$^{\downarrow}$, then take the difference between UNet$^{\downarrow}$ and UNet. These plots are based on BCSS dataset.

Table 6
mIoU score of ADS_UNet trained in 3 modes based on the BCSS dataset. Each UNet^d is trained for 70 epochs.

	UNet^1	UNet^2	UNet^3	UNet^4	ens(α)
$\boldsymbol{\eta}^d$	**47.77**	56.34	58.37	59.20	59.64
$\tilde{\boldsymbol{\eta}}^d$	47.14	55.42	58.52	58.95	60.10
$\tilde{\boldsymbol{\eta}}^d$(sum)	46.93	**56.91**	**60.11**	**60.26**	**61.05**

59.41%, respectively. This is consistent with $\tilde{\eta}_{i,j}$ values (see Figure 6(c)), $\eta_{3,1} > \eta_{2,2} > \eta_{0,4}$. A similar trend in the changes to $\tilde{\eta}^d_{i,j}$ in the iterative training process of ADS_UNet is also observed in Figure 7(e)- 7(h).

Figure 6(c) and Figure 7(e)- 7(h) not only show us how the parameters of different layers change during the training process, but also indicate that: 1) the importance of parameter varies from layer to layer; 2) the significance of parameters also vary throughout the training process. This is the effect of normalization of the weights $\tilde{\eta}^d_{i,j}$, which introduces competition between the layers. And also, 3) the competition between the layers will continue until equilibrium is reached.

5.2.2. *Preventing $\eta^d_{i,j}$ from vanishing leads to higher segmentation performance.*

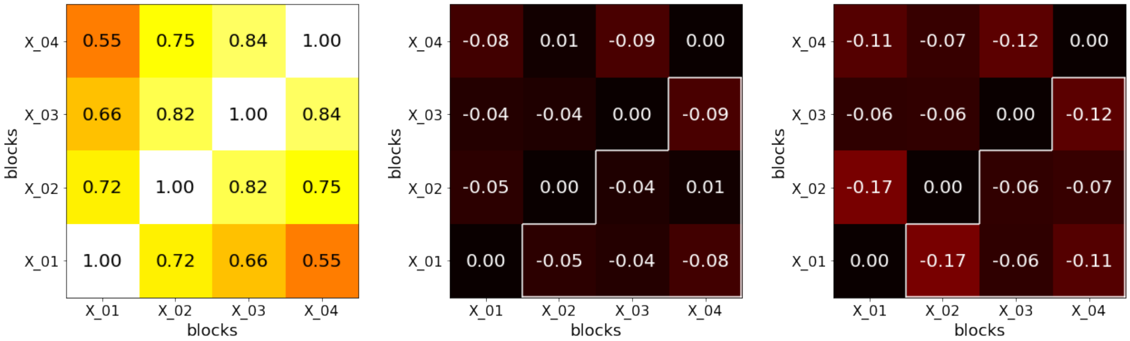
In equation (4), we redefine $\eta^d_{i,j}$ as $\tilde{\eta}^d_{i,j}$ to enforce all layers to learn features that are directly discriminative for classifying textures. We then sum the probability maps produced by these layers based on their importance factors $\tilde{\eta}^d_{i,j}$ to generate the segmentation map of UNet^d (defined in equation (6)). To verify if this constraint range and the weighted combination yields better performance or not, we run experiments on BCSS dataset, in which ADS_UNet is trained in 3 modes:

1) $\boldsymbol{\eta}^d$: with its element $\eta^d_{i,j}$ being trained without range constraint. After UNet^d is trained, the output of the layer which has the largest $\eta^d_{i,j}$ value is selected to generate the final segmentation map. i.e., let $(i', j') = \arg\max_{(i,j)}(\eta^d_{i,j})$, the final probability map is obtained by $\hat{y}^d = \hat{y}^{i',j'}$, with $\hat{y}^{i,j}$ defined in equation (1). Then $\hat{y}^d$ is used to compare with the ground truth to calculate the α^d (the weight of UNet^d).

2) $\tilde{\boldsymbol{\eta}}^d$: $\tilde{\eta}^d_{i,j}$ is bounded in $[\frac{1}{2(d+1)}\frac{d+2}{2(d+1)}]$, according to equation (4). The final segmentation map generation and α^d calculation criteria are the same as 1).

3) $\tilde{\boldsymbol{\eta}}^d$ (sum): training criteria is the same as 2). While the segmentation map produced by model UNet^d is the weighted summation of multi-scale prediction (using equation (6)), which is then used to calculate the α^d.

The results of training ADS_UNet in 3 different modes are reported in Table 6, where ADS_UNet with bounded $\tilde{\boldsymbol{\eta}}^d$ is seem to slightly surpass the unbounded one. After combining the probability maps produced by supervision layers based on the layer importance factors $\tilde{\eta}^d_{i,j}$, the mIoU score on the BCSS dataset is further improved by 0.95 points. To explain the results of Table 6, the loss, $\eta^d_{i,j}$ and $\tilde{\eta}^d_{i,j}$ of the ADS_UNet (trained in mode 1 and mode 3) are tracked and visualized in Figure 7. As observed from Figure 7(i)-7(p), when there is no range constraint on $\eta^d_{i,j}$, only one specific layer's loss dominates the learning process and



(a) ADS_UNet (b) diff(ADS_UNet,UNet^e) (c) diff(ADS_UNet,UNet++)

Figure 9: (a) Feature similarity of the output layers of ADS_UNet. (b) and (c) We calculate the feature similarity matrix for UNet^e and UNet++, then take the difference between ADS_UNet and UNet^e, UNet++. These plots are based on BCSS dataset.

the loss of other layers is almost negligible ($\eta^d_{i,j}$ close to 0), after training for a few epochs. But the loss increases ($\mathcal{L}^{3,0}$ in Figure 7(k) and $\mathcal{L}^{4,0}$ in Figure 7(l)), so there is reduced discriminability at the intermediate layers ($X^{3,0}$, $X^{4,0}$) still. However, this phenomenon is eliminated after the range constraint is imposed, to suppress the weight of the dominant layer and to enable those of the others to grow, as shown in Figure 7(a)- 7(h). That means, by retaining the information from previous layers, the range of features that are being learned are increased, therefore leading to better performance. Note that $\mathcal{L}^{3,0}$ in Figure 7(c) and $\mathcal{L}^{4,0}$ in Figure 7(d) keep decreasing, differs from that of Figure 7(k) and Figure 7(l).

5.3. Feature Similarity of Hidden Layers

Since deep supervision provides features of intermediate blocks with a direct contribution to the overall loss, the similarity of features learned by these blocks will be higher than those of the original UNet. Centered Kernel Alignment (CKA) has been developed [18] as a tool for comparing feature representations of neural networks. Here we use CKA to characterize the similarity of feature representations learned by different convolutional blocks in UNet$^{\downarrow}$. As shown in Figure 8(b), the similarity of features extracted by blocks in UNet$^{\downarrow}$ is mostly higher than in their counterparts in UNet (although 6 of similarity entries in UNet have lower values than that of UNet), which is consistent with our expectation (the 20 positive values add up to 1.89 vs. the 6 negative values add up to -0.47).

5.4. Feature Diversity of Output Layers

Ensemble-based learning methods, such as AdaBoost, rely on the independence of features exploited by classifiers in its ensemble [10]. If base learners produce independent outputs, then the segmentation accuracy of the ensemble can be enhanced by majority weighting. Figure 9(a) characterize the feature similarity of output layers of ADS_UNet. Figure 9(b) and 9(c) shows that features learned by the output layers of ADS_UNet are less similar than those in UNet^e (the values add up to -0.29) and UNet++ (the values add up to -0.59). Our interpretation is that this can be attributed to the stage-wise additive learning, followed by the sample weight updating rule of ADS_UNet, and may explain why ADS_UNet outperforms UNet^e and UNet++.

6. Conclusion

In this paper, we propose a novel stage-wise additive training algorithm, ADS_UNet, that incorporates the AdaBoost algorithm and greedy layer-wise training strategy into the iterative learning progress of an ensemble model. The proposed method has the following advantages: 1) The stage-wise training strategy with re-weighted training samples empowers base learners to learn discriminative and diverse feature representations. These are eventually combined in a performance-weighted manner to produce the final prediction, leading to higher accuracy than those achieved by other UNet-like architectures; 2) In the configuration of base learners, intermediate layers are supervised directly to learn discriminative features, without the need for learning extra up-sampling blocks. This, therefore, diminishes memory consumption and computational burden; 3) By introducing layer competition, we observe that the importance of feature maps produced by layers varies from epoch to epoch at the training stage, and different layers contribute differently in a manner that is learnable; 4) ADS_UNet is more computationally efficient (fewer requirements on GPU memory and training time) than UNet^e, UNet++, CENet and transformer-based UNet variants, due to its cascade training regimen.

However, the ADS_UNet has the following limitation that we would like to address in future work: currently, the sample re-weighting training criteria restricts the ADS_UNet to only update the weights of samples at a relatively coarse granularity. In future work, more fine-grained re-weighting criteria will be explored to guide successive base learners to pay more attention to regions/pixels that are difficult to distinguish. It would also be promising to integrate the AdaBoost and stage-wise training with a Transformer-like architecture to further improve segmentation performance.

Acknowledgement

The authors acknowledge the use of the IRIDIS High-Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work. Yilong Yang is supported by China Scholarship Council under Grant No. 201906310150.

References

[1] Amgad, M., Elfandy, H., Hussein, H., Atteya, L.A., Elsebaie, M.A., Abo Elnasr, L.S., Sakr, R.A., Salem, H.S., Ismail, A.F., Saad, A.M., et al., 2019. Structured crowdsourcing enables convolutional segmentation of histology images. Bioinformatics 35, 3461–3467.

[2] Awan, R., Sirinukunwattana, K., Epstein, D., Jefferyes, S., Qidwai, U., Aftab, Z., Mujeeb, I., Snead, D., Rajpoot, N., 2017. Glandular morphometrics for objective grading of colorectal adenocarcinoma histology images. Scientific reports 7, 1–12.

[3] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al., 2007. Greedy layer-wise training of deep networks. Advances in neural information processing systems 19, 153.

[4] Bengio, Y., Louradour, J., Collobert, R., Weston, J., 2009. Curriculum learning, in: Proceedings of the 26th annual international conference on machine learning, pp. 41–48.

[5] Cui, H., Jiang, L., Yuwen, C., Xia, Y., Zhang, Y., 2022. Deep u-net architecture with curriculum learning for myocardial pathology segmentation in multi-sequence cardiac magnetic resonance images. Knowledge-Based Systems , 108942.

[6] Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. The Journal of Machine learning research 7, 1–30.

[7] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., 2021. An image is worth 16x16 words: Transformers for image recognition at scale, in: 9th International Conference on Learning Representations.

[8] Dou, Q., Chen, H., Jin, Y., Yu, L., Qin, J., Heng, P.A., 2016. 3d deeply supervised network for automatic liver segmentation from ct volumes, in: International conference on medical image computing and computer-assisted intervention, Springer. pp. 149–157.

[9] Fahlman, S., Lebiere, C., 1989. The cascade-correlation learning architecture. Advances in neural information processing systems 2.

[10] Freund, Y., Schapire, R.E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences 55, 119–139.

[11] Friedman, J.H., 2001. Greedy function approximation: a gradient boosting machine. Annals of statistics , 1189–1232.

[12] Gao, Y., Zhou, M., Liu, D., Yan, Z., Zhang, S., Metaxas, D.N., 2022. A data-scalable transformer for medical image segmentation: architecture, model efficiency, and benchmark. arXiv preprint arXiv:2203.00131 .

[13] Hastie, T., Rosset, S., Zhu, J., Zou, H., 2009. Multi-class adaboost. Statistics and its Interface 2, 349–360.

[14] Herbold, S., 2020. Autorank: A python package for automated ranking of classifiers. Journal of Open Source Software 5, 2173.

[15] Isensee, F., Jaeger, P.F., Kohl, S.A., Petersen, J., Maier-Hein, K.H., 2021. nnu-net: a self-configuring method for deep learning-based biomedical image segmentation. Nature methods 18, 203–211.

[16] Ji, Z., Telgarsky, M., 2020. Directional convergence and alignment in deep learning. Advances in Neural Information Processing Systems 33, 17176–17186.

[17] Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

[18] Kornblith, S., Norouzi, M., Lee, H., Hinton, G., 2019. Similarity of neural network representations revisited, in: International Conference on Machine Learning, PMLR. pp. 3519–3529.

[19] Kumar, N., Verma, R., Anand, D., Zhou, Y., Onder, O.F., Tsougenis, E., Chen, H., Heng, P.A., Li, J., Hu, Z., et al., 2019. A multi-organ nucleus segmentation challenge. IEEE transactions on medical imaging 39, 1380–1391.

[20] Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z., 2015. Deeply-supervised nets, in: Artificial intelligence and statistics, pp. 562–570.

[21] Long, J., Shelhamer, E., Darrell, T., 2015. Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3431–3440.

[22] Luo, H., Changdong, Y., Selvan, R., 2022. Hybrid ladder transformers with efficient parallel-cross attention for medical image segmentation, in: Medical Imaging with Deep Learning.

[23] Ma, M., Xia, H., Tan, Y., Li, H., Song, S., 2022. Ht-net: hierarchical context-attention transformer network for medical ct image segmentation. Applied Intelligence , 1–14.

[24] Marquez, E.S., Hare, J.S., Niranjan, M., 2018. Deep cascade learning. IEEE transactions on neural networks and learning systems 29, 5475–5485.

[25] Opitz, D., Maclin, R., 1999. Popular ensemble methods: An empirical study. Journal of artificial intelligence research 11, 169–198.

[26] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. Pytorch: An imperative style, high-performance deep learning library, in: Advances in neural information processing systems, pp. 8026–8037.

[27] Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, Springer. pp. 234–241.

[28] Roy, A.G., Navab, N., Wachinger, C., 2018. Concurrent spatial and channel 'squeeze & excitation'in fully convolutional networks, in: International conference on medical image computing and computer-assisted intervention, Springer. pp. 421–429.

[29] Smith, L.N., Topin, N., 2019. Super-convergence: Very fast training of neural networks using large learning rates, in: Artificial intelligence and machine learning for multi-domain operations applications, International Society for Optics and Photonics. p. 1100612.

[30] Taherkhani, A., Cosma, G., McGinnity, T.M., 2020. Adaboost-cnn: An adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning. Neurocomputing 404, 351–366.

[31] Vahadane, A., Peng, T., Sethi, A., Albarqouni, S., Wang, L., Baust, M., Steiger, K., Schlitter, A.M., Esposito, I., Navab, N., 2016. Structure-preserving color normalization and sparse stain separation for histological images. IEEE transactions on medical imaging 35, 1962–1971.

[32] Xie, S., Tu, Z., 2015. Holistically-nested edge detection, in: Proceedings of the IEEE international conference on computer vision, pp. 1395–1403.

[33] Zhou, Q., Wu, X., Zhang, S., Kang, B., Ge, Z., Latecki, L.J., 2022. Contextual ensemble network for semantic segmentation. Pattern Recognition 122, 108290.

[34] Zhou, Z., Siddiquee, M.M.R., Tajbakhsh, N., Liang, J., 2019. Unet++: Redesigning skip connections to exploit multiscale features in image segmentation. IEEE transactions on medical imaging 39, 1856–1867.

[35] Zhu, Q., Du, B., Turkbey, B., Choyke, P.L., Yan, P., 2017. Deeply-supervised cnn for prostate segmentation, in: 2017 International Joint Conference on Neural Networks (Ijcnn), IEEE. pp. 178–184.

Yilong Yang received the master degree in software engineering from Xiamen University, Xiamen, China, in 2019. He is currently a Ph.D candidate with Vision, Learning and Control Research Group, University of Southampton, United Kingdom. His research interests include computer vision and geometric deep learning.

Srinandan Dasmahapatra received the Ph.D. degree in physics from the State University of New York, Stony Brook, NY, USA, in 1992. He is currently an Associate Professor in the School of Electronics and Computer Science, University of Southampton, Southampton, U.K. His research interests include artificial intelligence and pattern recognition.

Sasan Mahmoodi received the Ph.D degree from the University of Newcastle, Newcastle upon Tyne, U.K., in 1998. He is currently an Associate Professor in the School of Electronics and Computer Science, University of Southampton, Southampton, U.K. His research interests include medical image processing, computer vision, and modeling of the biological vision.

A. Python code for counting incorrect labels in down-scaled masks

```
import numpy as np
def avg_pool(mask, mask_shape, window_size):
    err_count_per_mask = 0
    row, col = mask_shape/window_size
    for i in range(row):
        for j in range(col):
            window = img[i:i+window_size,j:j+window_size]
            if len(np.unique(window))>1:
                err_count_per_mask+=1
    return err_count_per_mask
def get_incorrect_ratio(mask_shape, mask_list):
    down_factor = [2, 4, 8, 16]
    index       = [0, 1, 2, 3]
    err_count_per_scale = [0, 0, 0, 0]
    for mask in mask_list:
        for idx,f in zip(index, down_factor):
            err_count_per_scale[idx] += avg_pool(mask,
                   mask_shape, f)
    pixel_count = [mask_shape/f for f in
          down_factor]**2*len(mask_list)
    return np.array(err_count_per_scale)/np.array(pixel_count)
```