Hidy Gábor

Transferability metrics and medical image segmentation with neural networks

Master's thesis

Supervisor:

András Lukács

Department of Computer Science
Eötvös Loránd University, Faculty of Science
Institute of Mathematics

Eötvös Loránd University
Faculty of Science
2024

Acknowledgments

This thesis is a culmination of work I have been doing mainly in the past three years (although some of it has started even before that). As such, there have been many people along the way who have helped nudge me forward. I appreciate every encouraging word, every idea seed I have received. Here, I would like to take the opportunity to thank a few people by name.

First and foremost, my supervisor, András Lukács, who introduced me to the world of medical imaging, and without whose vast support none of my research or accomplishments would have been possible. Bence Bakos comes at a close second. He has been my research partner in multiple projects for several years now, and is also the co-author of the paper that would publish the results in Chapter 4 of this thesis. In addition to Bence, Bálint Csanády and Thomas Kolb also contributed to the code base used for the experiments presented here. And, last but not least, I thank Vivien Mikes, my mother, for her help with creating Figures 1.2–1.4.

Contents

Introduction

Within the field of computer vision and deep learning, medical image processing offers some unique challenges, which in turn lead to unique solutions in terms of architectures, loss functions, training methods, and evaluation metrics. One of these problems is a lack of labeled training data. Medical image segmentation, in particular, suffers from this problem, since labeling is time intensive, and requires medical experts. For this reason, neural networks for medical image segmentation are often first pretrained on large datasets containing natural images. However, this approach is not without its own challenges.

This work aims to explore the challenges arising when pretraining medical imaging models on ImageNet, and to offer a possible solution. It will show that depending on the model, no or a short ImageNet pretraining might be beneficial. How long a pretraining is necessary can be predicted using a novel transferability metric, that estimates how well a given pretrained model will transfer to another dataset.

The rest of this thesis is organized as follows. Chapter 1 provides an overview of some of the most popular deep learning methods used in medical image segmentation. Chapter 2 presents a set of over three hundred experiments carried out using these methods. Chapter 3 introduces the notion of transferability metrics and provides a literature overview of existing ones.

Chapter 4 introduces a novel transferability metric, and demonstrates its effectiveness on the experiments introduced in Chapter 2. The contents of this chapter are based on a research project I conducted with Bence Bakos, under the supervision of András Lukács. Our findings also led to a paper that is, at the submission of this thesis, under review for the 2024 European Conference on Computer Vision.

Chapter 1

Medical image segmentation with neural networks

This chapter aims to provide an overview of the field of medical image segmentation. It should be noted that the methods described here can be applied to any segmentation task. However, most of them originate from, and are mostly used for medical imaging problems.

We define the task of **semantic segmentation** as a task which aims to map an input $x \in \mathbb{R}^{\Lambda \times C_{\text{in}}}$ to an output $y \in [0,1]^{\Lambda \times C_{\text{out}}}$, where at every position κ, the entries of y_κ sum to 1. Λ can contain any number of leading dimensions, that must be present both in the inputs and outputs. In image processing, $\Lambda = H \times W$, where H is the height, W is the width of the image. The input channel size C_{in} in this case is usually either 1 (grayscale images) or 3 (RGB images). Thus, image segmentation tasks ascribe a classification label for each pixel. Common examples in medical imaging include localizing certain abnormal (e.g., infected or cancerous) areas of the input, or segmenting different organs on an X-ray or MRI image.

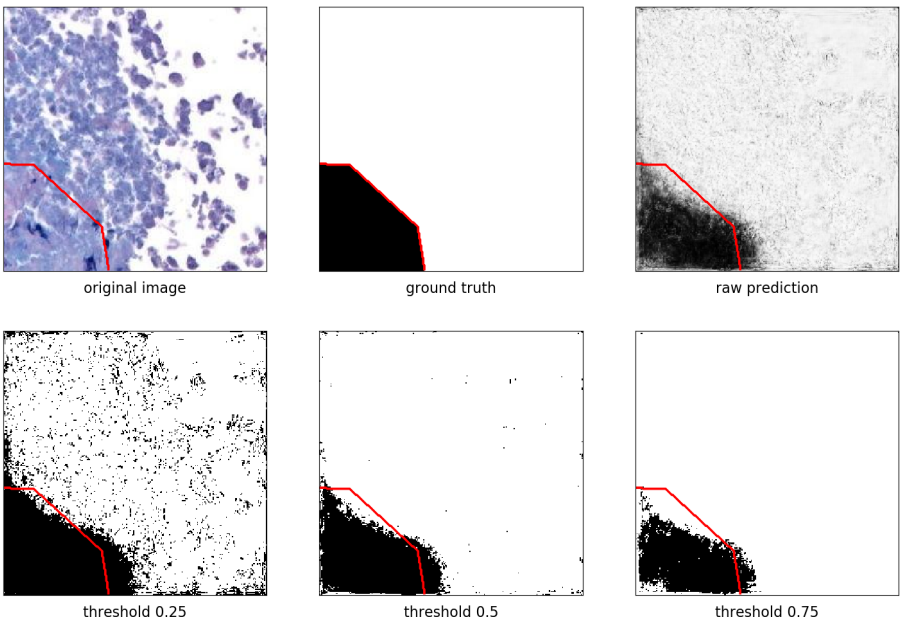


Figure 1.1. Example of an image, its corresponding segmentation mask (here denoting a cancerous region of a histopathology slide), the continuous output of a segmentation model, and its hard predictions at different thresholds. Positive pixels are colored black.

The ground truth label, or **mask**, y is a $\{0,1\}$-tensor, where $y_{\kappa,c} = 1$ if the pixel at position κ belongs to class c. Predictions are provided as continuous distributions for each pixel. The hard prediction is taken as the class with maximal probability. If there are two classes (i.e., the problem is binary), the ground truth mask might be represented as a $\{0,1\}^{\Lambda}$-tensor, and predictions might be given as $[0,1]^{\Lambda}$-tensors. In that case, positions where the ground truth mask is 1 are referred to as **positive**, while the others as **negative**. Hard prediction is calculated by counting each prediction above a certain threshold as positive. Throughout this work, this threshold will always be 0.5.

1.1 Architectures for segmentation

Classical image processing networks were designed for classification, i.e., mapping a variable sized input to a fix sized output. In contrast, segmentation requires the output size to follow the input size, which necessitates modifications in the architecture.

The most naïve approach is to obtain the prediction for each pixel separately, using a classification network in a sliding-window setup. Earliest deep segmentation networks worked this way [3]. This method is computationally inefficient, since it requires running a deep network repeatedly for each pixel.

Fully convolutional networks [6] address this problem by providing pixelwise predictions for the whole image in one pass. Original FCNs converted a CNN for image classification – at the time of the original paper, the state-of-the-art models were AlexNet [2], VGG16 [5], and GoogleNet [7] – to a segmentation model by adding a few additional upsampling blocks, which could provide local predictions.

This approach allows for predictions for all pixels in just one call of the model, thus providing a considerable improvement in time complexity compared to sliding-window networks. However, it creates another problem: the data representation reaches a bottleneck toward the middle of the architecture, which might cause problems since later the model needs to be able to reconstruct local information to provide the segmentation masks. The solution to this problem is "U-shaped" networks, that add new connections to fully convolutional architectures, to provide local, position-sensitive information to later layers.

1.1.1 U-shaped networks

U-Net [8] is a fully convolutional network, building on the original idea proposed in [6]. It is an encoder–decoder style structure, with additional connections introduced between the two parts. Figure 1.2 provides an illustration for the U-Net architecture.

The **encoder** part is a traditional image processing backbone. It consists of several **levels**, where the representation dimensions are constant within one level, and change in between levels: spatial dimensions are halved, and the channel size is doubled. In the original U-Net, each level consists of two 3×3 convolutions, with batch normalization and ReLU layers in between. Downsampling is done by max pooling.

The **decoder** is symmetrical to the encoder, typically with the same structure, except downsampling operations are replaced by upsampling. U-Net solves upsampling via **up-convolutions**, which are a bilinear interpolation followed by a 2×2 convolution. The input of each decoder level is not simply the output of the previous level, but also the output of the corresponding encoder level, through a **lateral skip connection**. The original implementation in [8] had the lateral skip connection slightly crop the output,

to account for the fact that they did not use padding for the convolutional layers. Since then, most implementations (including ours) pad their layers, thus the skip connections are identical.

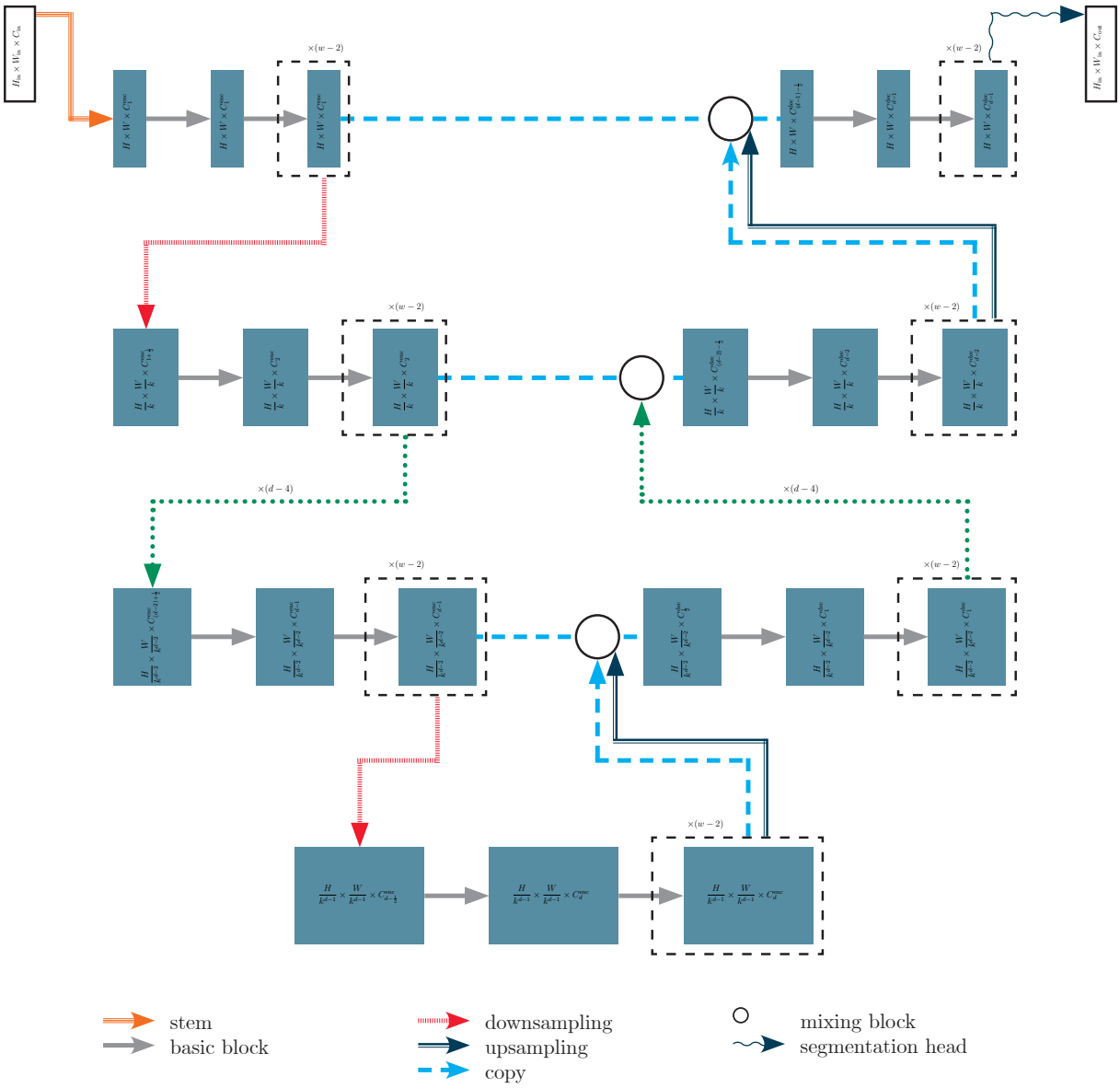


Figure 1.2. High-level architecture of U-shaped models. In the case of U-Net, basic blocks are 3×3 convolutions, downsampling is done by max pooling, upsampling is done by up-convolution blocks, and the mixing block concatenates the lateral signal to the upsampled signal.

1.1.2 Convolutional U-Net variants

As the leading model architecture for medical image segmentation, many variants of the original U-Net have appeared. The basic block may be enhanced by residual connections [21], or replacing them with an Inception [7] block [33]. The encoder part is often replaced by a more specialized image processing network, such as a ResNet50 [12], or even a hybrid CNN–transformer model [39].

Another direction in which the basic U-Net architecture can be modified is by replacing lateral skip connections. Attention U-Net [25] does this with additive attention gates.

Attention gate modules implement attention as

$$\alpha(Q, K, V) = \hat{\sigma}\Big(\Psi\big(\phi(Q + K)\big)\Big) \otimes V, \tag{1.1}$$

where $\hat{\sigma}$ is a sigmoid operation followed by upsampling, Ψ is a linear projection (1×1 convolution), ϕ is the ReLU operation, and $\otimes$ is elementwise multiplication. (Biases are omitted for the sake of simplicity.) In the case of Attention U-Net, $V = x$ is the signal propagated through the lateral connection, K is calculated from x via a 1×1 convolution and downsampling, and Q is a linear transformation of the signal coming from the previous up-block, as illustrated by Figure 1.3.

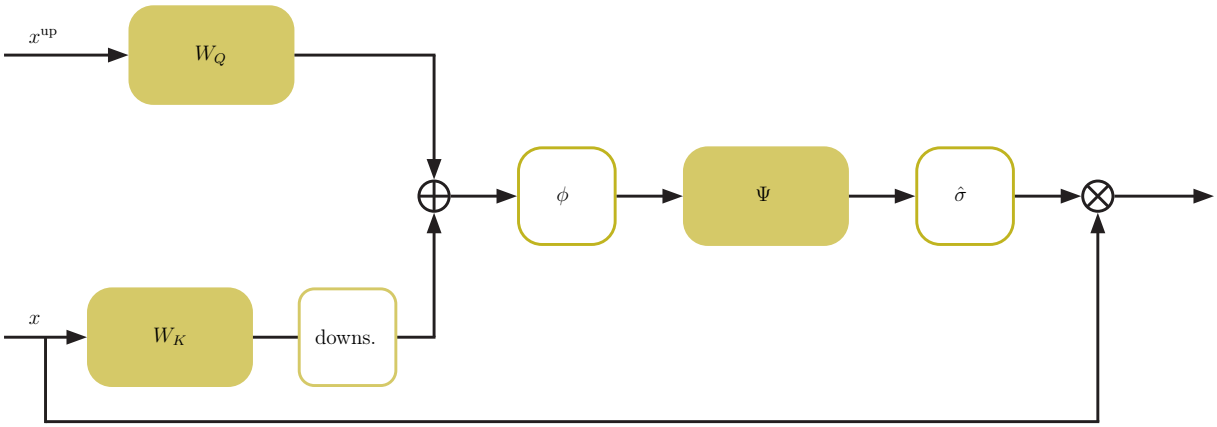


Figure 1.3. The additive attention gate module used in Attention U-Net. x is the signal propagated through the lateral connection, and x^{up} is the signal coming from the previous up-block. W_Q, W_K, and Ψ are 1×1 convolutions.

1.1.3 Swin U-Net

In recent years, transformer [18] architectures started appearing in computer vision tasks too. Vision Transformers [40] were the first to do so, which used a standard transformer encoder, with a flattened image as its input. While ViT produced good results, it had the downside of high computational complexity, since the attention modules have quadratic complexity in the input length.

Swin Transformer [45] is an image transformer that builds on the ideas of ViT, but introduces local attention to lower time complexity, and a hierarchical structure, like that of convolutional networks, where the spatial size of the representation decreases, while its feature size increases through consecutive levels of the model. This architecture lends itself well as an encoder to a U-Net-like model. Swin U-Net [48] builds on this, adding a decoder consisting of similar Swin Transformer blocks, and upsampling operations.

The rest of this subsection will go over the architectural details of Swin Transformer and Swin U-Net. Figure 1.4 illustrates the basic Swin Transformer block, which is used as the basic block in Swin U-Net.

Shifted window-based self attention

Instead of global self attention layers, Swin Transformer blocks use shifted window based self attention. In the first attention module of the block, the input tensor is divided into windows of size $M \times M$, and then attention is only calculated within these windows.

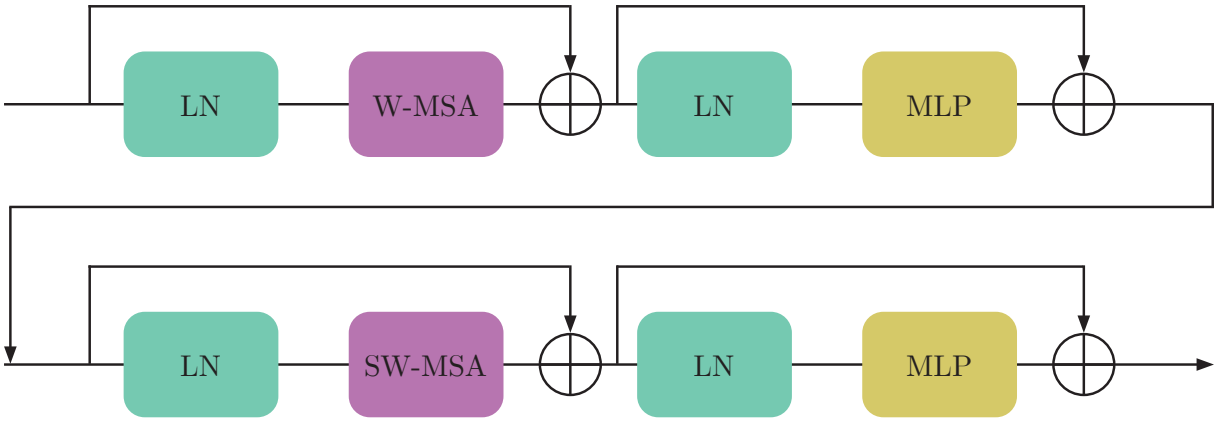
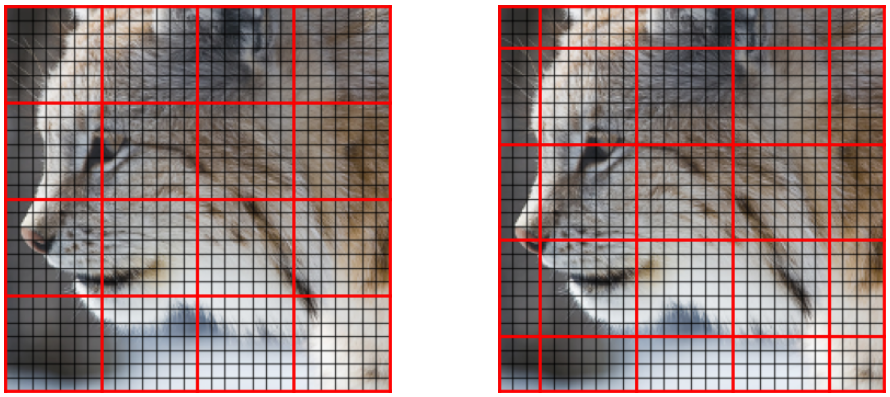


Figure 1.4. A Swin Transformer block. "LN" stands for layer normalization [11], "MLP" is a multilayer perceptron, and "W-MSA" and "SW-MSA" stand for windowed multihead self attention and shifted window multihead self attention respectively

This reduces the quadratic time complexity of global attention to a linear one. (If M is constant.)

The second attention module is also replaced by a local one, which works similarly to the first, but it is a shifted window attention, meaning the original window partition is shifted by $\left\lfloor \frac{M}{2} \right\rfloor \times \left\lfloor \frac{M}{2} \right\rfloor$. (The name Swin Transformer itself comes from the abbreviation for **s**hifted **win**dow.) This allows the information to mix between windows. Figure 1.5 illustrates window and shifted window partitions.



(a) Window partition (b) Shifted window partition

Figure 1.5. Window partitions for windowed and shifted window self attention, with a window size of $M = 7$. Bold red lines indicate window borders. In the shifted window case, there appear extra windows at the edge of the image that are less than $M \times M$ in size.

Relative positional bias

Instead of absolute positional embedding, each attention module within the Swin Transformer adds a position-based bias term to the attention weights, calculating attention as

$$\alpha(Q, K, V) = \sigma\left(\frac{QK^\top}{\sqrt{d}} + B\right)V, \tag{1.2}$$

where Q, K, and V are the query, key, and value matrices respectively, σ is the softmax function, d is the query dimension, and $B \in \mathbb{R}^{M^2 \times M^2}$ is a matrix whose entries depend

on the relative position of the key and query. The values of B are taken from a trainable $\hat{B} \in \mathbb{R}^{(2M-1)\times(2M-1)}$ matrix, since the relative position of two patches within one window always lies in the $[-M+1, M-1]$ range for both axes.

Patch partition, patch merging, and patch expanding

Following ViT, Swin Transformer first partitions the input pixels into patches, then applies a linear embedding to those patches.

Unlike ViT, Swin Transformer is hierarchical, therefore it needs downsampling layers. It uses **patch merging** layers to achieve this, which first reshape an $H \times W \times C$ signal to $\frac{H}{2} \times \frac{W}{2} \times 4C$, then use a pointwise linear transformation to transform the feature space to $2C$ dimensions. (This is equivalent to a 2×2 convolution of stride 2 applied to the $H \times W \times C$ signal, with output channel size $2C$.)

In the decoder part of Swin U-Net, upsampling is achieved by **patch expanding** layers, which function as the inverse of patch merging. They take a signal of shape $\frac{H}{2} \times \frac{W}{2} \times 2C$, apply a pointwise linear transformation to transform the feature space to $4C$ dimensions, then reshape the signal to $H \times W \times C$. (This is equivalent to a 2×2 transposed convolution [19] of stride 2 applied to the $\frac{H}{2} \times \frac{W}{2} \times 2C$ input, with output channel size C.)

1.2 Metrics and losses

While metrics such as accuracy or area under the ROC curve can be used to evaluate the performance of a segmentation model, they do not provide an accurate assessment in the case of imbalanced data, which is common in medical image segmentation tasks. For this reason, other metrics, better equipped for this type of data, are needed. This section will list some of the most commonly used metrics. Note that each metric is defined for a binary segmentation task. For multiclass segmentation, corresponding metrics are obtained by first calculating the binary metric for each class separately (regarding pixels from all other classes as negative), then taking their average (usually excluding the background).

Dice index, also known as Dice similarity coefficient or F_1 score, is calculated as

$$\frac{2 \cdot \mathrm{TP}}{2 \cdot \mathrm{TP} + \mathrm{FP} + \mathrm{FN}}, \tag{1.3}$$

where TP, FP, and FN are the number of true positive, false positive, and false negative predictions (pixels), in order. Since there is no term corresponding to the number of true negative predictions, models with a large false negative rate will have low Dice index.

A similar metric is the **Jaccard index**, or IoU (intersection over union) score, calculated as

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}. \tag{1.4}$$

Matthew's correlation coefficient is another, less widely used metric based on the confusion matrix. It is given by the formula

$$\frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}. \tag{1.5}$$

Mean average precision (mAP), the area under the precision–recall curve is sometimes used as an imbalance-friendly alternative to the area under the ROC curve.

So far, these metrics have all been classification metrics, that disregard the geometry of image segmentation masks and predictions. **Average Hausdorff distance**, on the other hand, is a metric defined between two sets in the plane – e.g.that of the positive ground truth pixels and positive predictions. It is calculated as

$$d_H(X,Y) = \frac{1}{2}\left(\frac{1}{|X|}\sum_{x\in X} d(Y,x) + \frac{1}{|Y|}\sum_{y\in Y} d(X,y)\right), \tag{1.6}$$

where $d(S,p)$ is the Hausdorff distance of set S and point p (infimum distance between p and points of S). A commonly used variant is the **modified Hausdorff distance**, abbreviated as HD95, where $d(S,p)$ is not the distance between p and the point in S closest to it, but rather the 5^{th} percentile of the distances between p and S.

Note that the Dice and Jaccard indices can also be understood as similarity measures between two sets. Specifically, the Dice index defined by Equation 1.3 is equivalent to

$$\frac{2|X\cap Y|}{|X|+|Y|}, \tag{1.7}$$

and the Jaccard index defined by Equation 1.4 is equivalent to

$$\frac{|X\cap Y|}{|X\cup Y|} \tag{1.8}$$

(hence the name intersection over union), where X is the set of positive pixels, and Y is the set of pixels with positive prediction.

Despite their differing definitions, these metrics seem to measure the performance of segmentation models equally well. Figure 1.6 shows the Spearman correlation of the mentioned metrics, as well as other metrics used mostly for (balanced) classification problems. Datapoints come from training and evaluating several models over two datasets (see Section 2.4 for details). Dice index, Jaccard index, and MCC show perfect correlation, while mAP and HD95 (replaced in the figure by its inverse, so all metrics work by assigning a higher score to better performing models) still stay within 95% correlation of them.

Another question is what loss function to use to deal with imbalanced data. Following the idea of the Dice index, the Dice loss [17] is based on the soft version of the Dice index. There are natural two ways to define the Dice index as a continuous function of the (flattened) ground truth values y and predictions $\hat{y}$, in a way that when $\hat{y}$ is a $\{0,1\}$-vector, the definition gives back the original Dice index.

One is based on Equation 1.3, where TP is substituted with $y\hat{y}$, FP with $(\mathbb{1}-y)\hat{y}$, and FN with $y(\mathbb{1}-\hat{y})$, resulting in

$$\mathcal{D}(y,\hat{y}) = 1 - \frac{2y\hat{y}+\varepsilon}{2y\hat{y}+(\mathbb{1}-y)\hat{y}+y(\mathbb{1}-\hat{y})+\varepsilon}. \tag{1.9}$$

(ε is a small smoothing term to avoid dividing by 0.)

The other one is based on Equation 1.7, where we replace the intersection of two sets with the scalar product of their incidence vectors, and the cardinality of a set with the square of its length, giving us

$$\mathcal{D}(y,\hat{y}) = 1 - \frac{2y\hat{y}+\varepsilon}{\|y\|_2^2+\|\hat{y}\|_2^2+\varepsilon}. \tag{1.10}$$

It is easy to see that the two definitions are not equivalent: Equation 1.9 would be equivalent to using $\|\cdot\|_1$ instead of $\|\cdot\|_2^2$ in Equation 1.10.

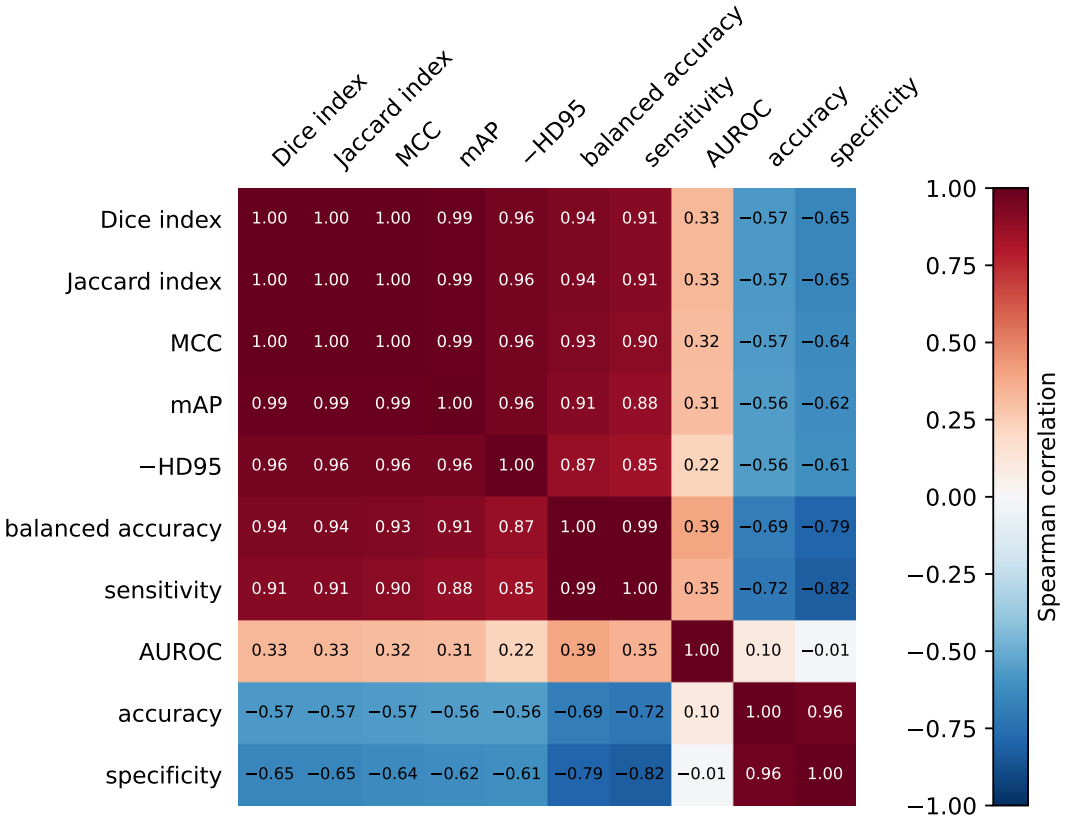


Figure 1.6. Spearman's rank correlation coefficients between frequently used segmentation metrics. (−HD95 indicates the inverse of the modified Hausdorff distance, which is used so that for all metrics a bigger score indicates a better performance.)

1.3 Pretraining models

Medical image processing, especially semantic segmentation tasks, tend to suffer from a lack of labeled training data. For this reason, pretraining on a large dataset is often beneficiary.

Early fully convolutional networks often started from a classification backbone pretrained on a large dataset – most frequently ImageNet [10]. ImageNet, and models pretrained on ImageNet are easily accessible and widely known, therefore it is an attractive dataset for pretraining. However, it has been shown that in several fields, including object detection [29] and medical image segmentation [44], ImageNet pretraining offers no real performance benefits. While it does tend to speed up convergence, that is only an advantage if we do not count the pretraining as part of the training time. Otherwise, any model pretrained on ImageNet is much slower to converge than a model trained from scratch.

With U-shaped segmentation networks, ImageNet pretraining is only done on the subset of the weights. The encoder is complemented with a dense head, and trained for image classification on ImageNet. The decoder is then initialized from scratch for the downstream task. During downstream training, encoder weights might be used as a frozen feature extractor, or they might be further updated based on the downstream loss function.

While it is clear that ImageNet pretraining is ineffective and unnecessary in certain

contexts, it is still widely used, mostly due to its accessibility. Furthermore, in certain contexts – see for example Table 2.10 and Subfigure 3.1a in later chapters – ImageNet pretraining leads to better performance than no pretraining. Our work, explained in Chapter 4, gives a solution that can prevent performance loss, or potentially even unnecessary pretraining time.

Chapter 2

Experiments

We have conducted six ImageNet trainings, and over 300 downstream segmentation experiments. This chapter describes these experiments in detail. Everything was implemented using PyTorch [50].* neptune.ai [49] was used for experiment tracking.

2.1 Model architectures

This section details the three architectures used in our segmentation experiments. Figure 1.2 provides a general architectural setup for U-shaped networks. Table 2.1 describes each of the three models using that language. More detailed explanations for each architecture are provided below.

	basic U-Net	**R50 Atn. U-Net**	**Swin U-Net**
depth	5	5	4
width	2	2	2
C_1^{enc}	64	64	96
stem	conv. 3×3	conv. 7×7	patch embedding
basic block	conv. 3×3	conv. 3×3	Swin T. block
downsampling	max pooling	max pooling	patch merging
upsamling	up convolution	up convolution	patch expanding
head	conv. 1×1	conv. 1×1	patch expanding
mixing block	concatenation	attention gate	concatenation

Table 2.1. Architectures of the models used, as described via the blocks in Figure 1.2

Basic U-Net

Our implementation followed the original [8] closely. The first layer is a 3×3 convolutional stem that transforms the input to 64 channels. Then five encoder and four decoder levels follow, with two blocks each. One block consists of a 3×3 convolution, followed by a batch normalization layer and ReLU. The channel size is doubled between each level by the first block, so that the bottom level has 1024 channels. Padding is applied to all convolutional layers, so the full model preserves the spatial size of its input. After each

*Code is available at `https://github.com/aielte-research/MedSegPretrainImageNet`.

level, a 2×2 max pooling operation halves the spatial sizes. After the last decoder block, a 1×1 convolution provides the segmentation head.

ResNet50 Attention U-Net

For this model, the encoder and decoder were not symmetric. The decoder was a five-level basic U-Net decoder, with levels of 256, 128, 64, 32, and 16 consisting of two blocks of 3×3 convolutions, followed by batch norm and ReLU, with up-convolution between the levels.

The encoder was a ResNet50 backbone, following the original architecture [12]. It started with a 7×7 convolution, followed by batch normalization and ReLU, transforming the input to 64 channels. Then a 3×3 max pooling operation, with stride 2, was used to downsample the input. After that, four levels followed, with channel sizes 256, 512, 1024, and 2048, and widths 3, 4, 6, and 3 respectively. (Width here indicates the number of residual blocks within a level.) Bias was omitted from its convolutional layers.

Lateral skip connections were applied after the output of each encoder level (except for the last), including the 7×7 convolution. Since that only accounted for four decoder levels, the final decoder level did not receive a lateral skip connection. The mixing block concatenated an attention gated signal – as illustrated in Figure 1.3 – to the output of the up-convolution block.

Swin U-Net

Our implementation of Swin U-Net followed the original architecture [48] closely. The first layer was a patch partition followed by linear embedding, downsampling the spatial dimensions by 4, and transforming the channel size to 96. Then four levels followed, each consisting of two Swin Transformer blocks – as illustrated by Figure 1.4 –, and a patch merging operation. The decoder part also consisted of Swin Transformer blocks, with patch expanding layers for upsampling. The final segmentation head was also a patch expanding layer, upsampling by 4, and a linear projection (1×1 convolution) to transform the input to C_{out} channels.

Weight initialization

The weights of U-Net and ResNet50 Attention U-Net were initialized in the same manner. Weights of convolutional layers were initialized according to He's initialization scheme [9], that is, from $\mathcal{N}(0, \sigma^2)$, where

$$\sigma = \sqrt{\frac{\gamma}{\delta}}, \tag{2.1}$$

where δ is the number of input neurons, and $\gamma = 2$.

In the case of Swin U-Net, the patch embedding and final patch expanding layers were initialized similarly, but with $\gamma = \frac{1}{3}$. The rest of the layers in this model were initialized from a truncated normal distribution with 0 mean, 0.02 standard deviation, and values not from $[-2, 2]$ resampled.

Biases were initialized to 0, and the bias and variance parameters of normalization layers were initialized to 0 and 1 respectively.

2.2 Pretraining the encoders

2.2.1 Overview of the methods used

Learning rate scheduling

In deep learning, the learning rate parameter of the optimization algorithm should usually not be kept constant. Specifically, in most cases it should be annealed near the end of the training, so once a local optimum has been found, the model weights do not change drastically from there. Annealing is usually done by setting the learning rate at the i^{th} iteration η_i to some monotonically decreasing function of i. Commonly used examples are the exponential decay

$$\eta_i = \gamma^i \eta_0, \tag{2.2}$$

and polynomial decay

$$\eta_i = \left(1 - \frac{i}{N}\right)^\gamma \eta_0, \tag{2.3}$$

where N is the total number of iterations. (γ in both cases is a chosen hyperparameter.)

Both of these drop from the initial learning rate η_0 quite quickly. In contrast, cosine annealing decay [16] anneals the learning rate according to the cosine curve

$$\eta_i = \frac{\eta_0}{2}\left(1 + \cos\left(\frac{i}{N}\pi\right)\right), \tag{2.4}$$

which has a flat slope near the start of the training.

All of these methods start with a large initial learning rate. However, deeper models can suffer from that, so it is common to use a linear warmup [22], where during the first few epochs, the learning rate is linearly increased from near-0 to the base learning rate η_0.

Decoupled weight decay

The term **weight decay** is often used to refer to two distinct modes of regularization. In its original sense, it describes taking an algorithm that calculates the model weights at iteration i as

$$\vartheta_i = \vartheta_{i-1} - m_i, \tag{2.5}$$

where m_i is given by the optimization method (for example m_i is the gradient of ϑ_{i-1} scaled by the current learning rate η_i in the case of standard gradient descent), and modifies it as

$$\vartheta_i = (1 - \lambda\eta_i)\vartheta_{i-1} - m_i. \tag{2.6}$$

However, instead of the weight decay described above, original deep learning models often used L^2 regularization, which modified the loss function $\mathcal{L}(y, \hat{y})$ as

$$\tilde{\mathcal{L}}(y, \hat{y}) = \mathcal{L}(y, \hat{y}) + \lambda\|\vartheta_{i-1}\|_2^2. \tag{2.7}$$

It is easy to see that in the case of standard stochastic gradient descent, Equations 2.6 and 2.7 are equivalent. However, in the optimizers used in practice, such as SGD with momentum, or Adam [4], the two are distinct. **Decoupled weight decay** [26] refers to using weight decay as per its original definition of Equation 2.6.

Stochastic depth

Training deep neural networks often proves a challenging task. Problems of exploding or vanishing gradients can arise, and tuning so many parameters can be slow. Training networks with **stochastic depth** [15] aims to circumvent this problem, by essentially training a set of shallower networks, then using a deep network at inference time. This is implemented by replacing their residual units of form $x \mapsto x + f(x)$ to

$$x \mapsto x + \delta\big(f(x)\big), \tag{2.8}$$

where $\delta(\cdot)$ is 0 with probability p, and identical with probability $1-p$. At inference, these modules are replaced with

$$x \mapsto x + (1-p)f(x), \tag{2.9}$$

which is the expected value of the module in Equation 2.8. This is essentially equivalent to taking an ensemble of both possible values of δ, with weights following their probability.

The value of p is generally set to 0 for the first block, and linearly increases throughout the network until it reaches some value $\bar{p}$, referred to as the **stochastic depth rate** of the network. With this setup, if there are n blocks, then during training only $\bar{p} \cdot \frac{n-1}{2}$ of them will be used on average, and inference will be equivalent to taking the weighted average of a 2^{n-1}-large ensemble network.

Label smoothing

Deep neural networks trained for classification tend to be overconfident, i.e., they tend to predict one class with close to 1 probability, and the rest close to 0, even when there is a significant chance of error. (That error might be because the model is not accurate enough, or because of noise in the training data.) **Label smoothing**, first introduced in [14], solves this by forcing the model to produce less confident predictions. It replaces the one-hot label with a new soft label, where the ground truth class has probability $1-\varepsilon$, and all other classes have probability $\frac{\varepsilon}{n-1}$, where n is the number of classes, and $\varepsilon < 0.5$ is the coefficient of label smoothing.

CutMix and mixup

In computer vision, various augmentation techniques are widely used to increase the generalization power of models, and to prevent overfitting. For classification network, most of these are transformations under which the label should remain invariant. However, CutMix [31] and mixup [20] go further than that, by taking two independent sample images, combining them, and changing the ground truth label to a linear combination of the two labels as well.

CutMix works by cutting out a randomly sampled rectangular patch from the first image and replacing it with the corresponding pixels of the second image. The weight of the two classes in the label will then be the relative area of each source image. **mixup**, on the other hand, takes a convex combination of the two images, and the label is also a convex combination of the two labels (represented as one-hot vectors), with the same coefficients. Figure 2.1 illustrates how the two techniques work.

Figure 2.1. Examples of CutMix (top) and mixup (bottom) mixing of two images, with the associated soft labels displayed at the top

RandAugment

Algorithm 2.1 RandAugment

Parameters: transforms, N (number of operations), M (magnitude)
Input: image

repeat N **times**
  $k \sim \mathcal{U}\big([0, \text{transforms.size} - 1]\big)$ ▷ choose integer uniformly
  transform = transforms[k]
  transform(image, M) ▷ apply transform to image with magnitude M
end

return image

RandAugment [35] performs a series of randomly chosen augmentations, described in Algorithm 2.1. It applies each transformation with equal probability. Figure 2.2 shows different instances of RandAugment applied to a sample image.

Figure 2.2. A sample image from ImageNet (leftmost), and four different transformations of it with RandAugment (with $N = 2$ and $M = 9$)

The magnitude M is an integer between 0 and 30 that describes how drastically to apply the transformation. Each transformation comes with its own magnitude range; if that transformation is applied, M is linearly scaled to that range. Table 2.2 describes

each transformation used in RandAugment, with their respective magnitude ranges. If the range is centered around 0, M is scaled to be between 0 and its maximum, and then flipped with probably $\frac{1}{2}$.

operation	**description**	**magnitude range**
Identity	no operation is performed	
AutoContrast	adjust contrast to maximal level	
Equalize	equalize image histogram	
Solarize	invert all pixels with intensity above m	$[0, 256]$
Posterize	discretize each pixel intensity to m bits	$[4, 8]$
Rotate	rotate the image by m degrees	$[-30, 30]$
Color	adjust the saturation of the image by a factor of $1 + m$	$[-0.9, 0.9]$
Contrast	adjust the contrast of the image by a factor of $1 + m$	$[-0.9, 0.9]$
Brightness	adjust the brightness of the image by a factor of $1 + m$	$[-0.9, 0.9]$
Sharpness	adjust the sharpness of the image by a factor of $1 + m$	$[-0.9, 0.9]$
ShearX	shear the image along the horizontal axis with rate m	$[-0.3, 0.3]$
ShearY	shear the image along the vertical axis with rate m	$[-0.3, 0.3]$
TranslateX	shift image along the horizontal axis by m pixels	$[-101, 101]$
TranslateY	shift image along the vertical axis by m pixels	$[-101, 101]$

Table 2.2. Description of transformations for RandAugment, with their magnitude ranges [27]. The parameter m in the description is the scaled version of the magnitude parameter M. Where a magnitude range is not present, the operation is magnitude-independent

2.2.2 Implementation details

We pretrained each encoder for 300 epochs, using Adam, and cosine annealing learning rate scheduling with linear warmup. We used decoupled weight decay and label smoothing for regularization, as well as stochastic depth for ResNet50 with a rate of 0.1, and the Swin Transformer with a rate of 0.2. We trained each model twice, using two different training setups, that we dubbed "simple" and "advanced". Here we provide an overview of each method. Table 2.3 provides an overview of the differences between the two schemes.

	simple scheme	**advanced scheme**
batch size	4096	4096 for convnets 1024 for Swin Transformer
base learning rate	0.004	$0.001 \cdot \frac{\text{batch size}}{1024}$
warmup length	20 for U-Net 5 for other models	20
augmentation	random flip	RandAugment, mixup, CutMix, random erasing

Table 2.3. Hyperparameters used in the two pretraining schemes

Algorithm 2.2 Augmenting images for ImageNet pretrainig

Input: image, label

$A \sim \mathcal{U}([0.8, 1])$ ▷ sample uniform random number
crop rectangle from image with relative area A ▷ random crop
resize rectangle to 224×224

$\omega \sim \mathcal{U}(\{0, 1\})$ ▷ sample uniform random bit
if ω **then**
  apply horizontal flip
end if

if advanced pretraining **then**
  RandAugment(image, $N = 2$, $M = 9$)

  $\xi \sim \mathcal{U}([0, 1])$
  if $\xi \leq 0.9$ **then**
    sample an (image_2, label_2) pair uniformly from the whole dataset
    apply random crop and RandAugment to image_2
    if $\xi \leq 0.5$ **then**
      CutMix(image, image_2, label, label_2)
    else
      mixup(image, image_2, label, label_2)
    end if
  end if

  $\psi \sim \mathcal{U}([0, 1])$
  if $\psi \leq 0.25$ **then** ▷ random erasing
    $S \sim \mathcal{U}([0.02, 0.33])$
    $r \sim \mathcal{U}([0.3, 3.3])$
    $h = \sqrt{Sr}$, $w = \sqrt{\frac{S}{r}}$
    randomly chosen area with relative size (h, w) is blackened out
  end if

end if

return image, label

During ImageNet pretraining, the encoder architectures were extended with a classification head. The output of the encoder was pooled along the spatial axes, and then a one-layer dense classification head with a softmax activation function was added to predict class probabilities. As for the loss function, the standard categorical cross entropy was used, with a label smoothing coefficient of 0.1. The Adam optimizer was applied, with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and decoupled weight decay of 0.05.

For our **simple pretraining**, we used a five-epoch learning rate warmup for the U-Net encoder, and twenty epochs for the other two models. We used a base learning rate of 0.004 and a batch size of 4096 for all models. We only used random resizing, crops, and

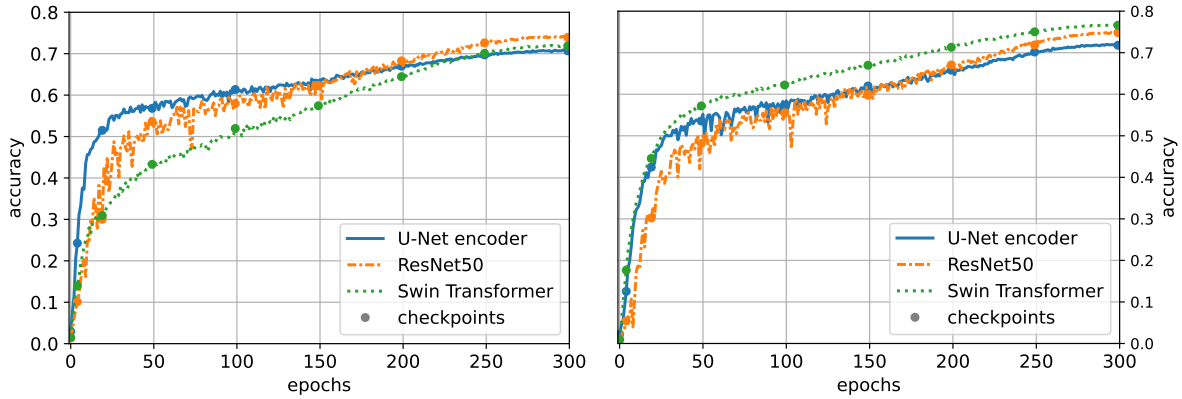
flips as augmentation during training.

Our **advanced pretraining** scheme is based on [42], following [45]. We followed the 1024 batch size for Swin Transformer, but we used a 4096 batch size for the convolutional networks, following the training setup used for ConvNeXt [47]. We used a fixed batch size to learning rate ratio, so we trained the Swin Transformer with a learning rate of 0.001, and the convolutional models with 0.004. We used a twenty-epoch warmup for all models. RandAugment, mixup, CutMix, and random erasing [34] were used for augmentation. The pseudocode for the augmentation is described in Algorithm 2.2. Label smoothing is applied after mixup or CutMix. For validation, images were resized so their shorter side would be 224, and then center crop was applied.

	model size		**accuracy**	
	GFLOPs	parameters	simple scheme	advanced scheme
U-Net encoder	14.9	19.9M	0.707	0.718
ResNet50	3.8	22.8M	0.739	0.748
Swin Transformer	3.1	21.2M	0.719	0.766

Table 2.4. Comparison of the model size and ImageNet classification performance of the three architectures used

Table 2.4 shows the size and final validation performance of each model. Figure 2.3 shows the learning curves. The weights of each model were savd after the 1st, 5th, 20th, 50th, 100th, 150th, 200th, 250th, and 300th epoch. In downstream segmentation experiments, detailed in Section 2.4, encoder weights were initialized from these checkpoints.



(a) Simple pretraining

(b) Advanced pretraining

Figure 2.3. Progression of validation accuracies on ImageNet during training

2.3 Medical image datasets

Three downstream segmentation datasets were used for training and evaluating segmentation models. Each was chosen from a different domain – one containing cardiac MRI, one chest X-ray, and one retina scan images –, the domains chosen to represent a large portion of medical imaging tasks. Two of the datasets were used for binary, and one for multiclass segmentation.

	ACDC	COVID-QU	IDRiD
domain	cardiac MRI	chest X-ray	retina scan
task type	multiclass	binary	binary
color channels	grayscale	grayscale	RGB
train set size	1902	2330	6426
test set size	1076	583	1350
size of preprocessed images	256×256	256×256	448×448

Table 2.5. Short description of each dataset

Table 2.5 provides basic information about each of the three datasets. Figure 2.4 shows a sample image, and its corresponding segmentation mask, from each dataset. In the rest of this section, a brief description of each dataset is provided.

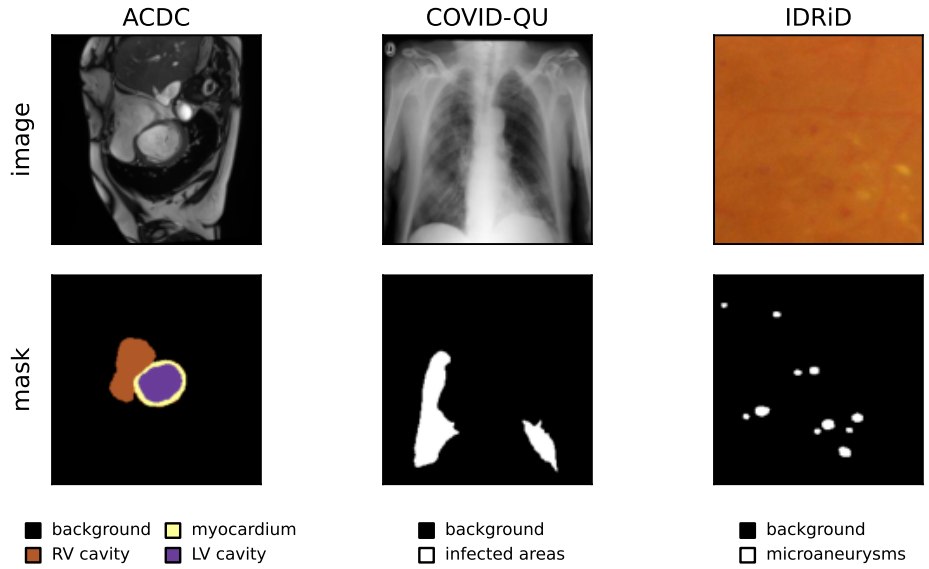


Figure 2.4. An example of a preprocessed image and its segmentation mask from each dataset

To the best of our knowledge, data in each dataset was collected and used responsibly.

ACDC

The ACDC (Automated Cardiac Diagnosis Challenge) dataset [24] contains cardiac MRI images of a total of 150 patients. The processed dataset provides a 4D representation of the interval of the cardiac cycle. From this, we only used the end-diastolic (ED) and the end-systolic (ES) frames per case, since segmentation masks are only provided for these 3D frames. On these masks, the right and left ventricular cavities, and the myocardium are annotated, resulting in a total of three classes (not counting the background). The same train–test split was used as in the original challenge, with 100 train and 50 test cases. The 3D frames were sliced along the z axis, providing on average twenty 2D slices, resulting in 1902 train and 1076 test images. All images were grayscale, 256×256 pixels.

COVID-QU

The COVID-QU-Ex dataset [46] consists of chest X-ray images from both healthy patients and people infected with COVID-19, or viral or bacterial pneumonia. It contains a total of 33 920 images, but only 2913 of those – all from people with COVID-19 – have masks with pixel-level markings of the infected areas. Out of these, the official 583-image test was used, which left us with 2 330 training images. All images were grayscale, 256×256 pixels. Records from healthy patients were used neither for training nor for inference.

IDRiD

The Indian Diabetic Retinopathy Image Dataset [32], or IDRiD, is a dataset consisting of retina scans, with labels related to diabetic retinopathy. It has around five hundred images with classification labels, and 81 images with segmentation masks for four different types of lesions – microaneurysms, hemorrhages, soft exudates, and hard exudates. For our experiments, we chose the microaneurysm masks to define a binary segmentation task. The images were divided into 54 train and 27 test sources, following the split of the original challenge. Each image is 2848×4288 pixels, which was first resized to 1120×2240, then the training and test slices were sampled from these. For testing, we simply partition the original image into 50 equal-sized 224×224 slices, giving us 1350 test images. For training, 119 slices were taken from the resized images by sliding a 448×448 window over the image, with a stride of 112. This resulted in 6426 RGB images with 448×448 pixels.

2.4 Downstream experiments

Implementation details

Images from ACDC and COVID-QU were resized to 224×224 using bilinear interpolation. For IDRiD, a randomly rotated 224×224 square was selected from the original 448×448 image. Grayscale images were converted to three channels by repeating their singular color channel three times. Images were augmented using random rotations (with angle sampled uniformly from $[0, \pi]$) and flips (with probability $\frac{1}{2}$). Images from IDRiD were also augmented by randomly jittering the brightness, saturation, contrast, and hue values of colored images, as described in Algorithm 2.3.

We trained for 150 epochs, using SGD with momentum 0.9 and weight decay 10^{-4}. We used polynomial learning rate decay, following Equation 2.3, with $\gamma = 0.9$. The learning rate was updated after each optimizer step. The loss was calculated as

$$\mathcal{L}(y, \hat{y}) = \frac{1}{C_{\text{out}} + 1} \sum_{c=0}^{C_{\text{out}}} \mathcal{D}(y_{\cdot,c}, \hat{y}_{\cdot,c}), \tag{2.10}$$

where $\mathcal{D}(y_{\cdot,c}, \hat{y}_{\cdot,c})$ is the binary Dice loss – as defined in Equation 1.10 – corresponding to class c. The Dice loss is calculated along the batch axis as well.

For multiclass segmentation, $\hat{y}$ is obtained by applying a pixelwise softmax to the segmentation mask. For binary segmentation, a sigmoid activation function determines $\hat{y}_{\cdot,1}$, and intensities for the background are calculated as $\hat{y}_{\cdot,0} = \mathbb{1} - \hat{y}_{\cdot,1}$.

Each architecture was trained on each dataset, with the encoder initialized randomly, and from each of the eighteen checkpoints explained in Section 1.3. Furthermore, for each initialization, two different training setups were investigated: training the full model, and training only the decoder. This resulted in 342 configurations.

Algorithm 2.3 ColorJitter

Input: $x \in [0,1]^{H \times W \times 3}$ image

$\lambda_b \sim \mathcal{U}\big([0.9, 1.1]\big)$ ▷ brightness jitter
$x = \lambda_b x$

$w = (0.2989, 0.587, 0.114)$
$\bar{x} = \sum_{c=1}^{3} w_c x_{\cdot,c}$ ▷ convert x to grayscale; weighting channels by w

$\lambda_c \sim \mathcal{U}\big([0.95, 1.05]\big)$ ▷ contrast jitter
$\mu = \frac{1}{HW} \sum_{h,w} \bar{x}_{h,w}$
$x = \lambda_c x + (1 - \lambda_c)\mu$

$\lambda_s \sim \mathcal{U}\big([0.9, 1.1]\big)$ ▷ saturation jitter
$x = \lambda_s x + (1 - \lambda_s)\bar{x}$

$x = \mathrm{HSV}(x)$ ▷ transform from RGB to HSV (hue–saturation–value space)

$\lambda_h \sim \mathcal{U}\big([-0.05, 0.05]\big)$ ▷ hue jitter
$x_{\cdot,1} = x_{\cdot,1} + \lambda_h \mod 1$ ▷ first channel of HSV is hue

$x = \mathrm{RGB}(x)$ ▷ transform from HSV to RGB

return x

Results

Each of the 342 configurations was trained five times. In all the following tables, performance was calculated by taking the average of these five runs. Tables 2.6–2.8 summarize* the validation performance of each model on each dataset, comparing training from scratch to initializing with a fully pretrained encoder. (Pretrained either with the simple or advanced pretraining scheme.)

However, full training or no pretraining did not always provide the best performance. Table 2.9 displays the Dice indices of the best performing of the 19 encoder initializations. Compared with the previous tables, it shows that, for example, the ResNet50 Attention U-Net and Swin U-Net models, with their encoders trained from scratch or pretrained fully on ImageNet, performed comparably worse on IDRiD than if the encoder was initialized from some intermediate weights.

This motivates the main goal of this thesis, which is to find a way to appropriately select a set of weights, during pretraining, which would then lead to optimal downstream performance. Table 2.10 is included here to help put these efforts into context later. It displays the ratio of the performance of the worst and best training setups. Entries where it is close to 1 correspond to training setups where the choice of initialization does not seem to matter. Conversely, entries where it is low suggest that choosing a good checkpoint to initialize the encoder from can lead to significant gain in downstream performance.

*Our GitHub repository contains a detailed table, with more metrics displayed, and with the individual results of each of the five runs. The table is available at `https://github.com/aielte-research/MedSegPretrainImageNet/blob/main/results/downstream_scores.csv`.

model	pretraining	DSC	IoU	mAP	−HD95	AUC	acc.
basic U-Net	none	0.908	0.833	0.966	−3.199	0.998	0.995
	simple	0.907	0.831	0.965	−3.190	0.998	0.995
	advanced	0.904	0.826	0.963	−3.354	0.998	0.995
R50 Atn. U-Net	none	0.899	0.819	0.961	−3.673	0.998	0.994
	simple	0.907	0.831	0.964	−3.161	0.997	0.995
	advanced	0.907	0.832	0.965	−3.072	0.998	0.995
Swin U-Net	none	0.807	0.682	0.880	−6.099	0.997	0.989
	simple	0.892	0.808	0.955	−3.492	0.999	0.994
	advanced	0.893	0.809	0.955	−3.241	0.999	0.994

(a) Full training

model	pretraining	DSC	IoU	mAP	−HD95	AUC	acc.
basic U-Net	none	0.872	0.776	0.941	−5.219	0.998	0.993
	simple	0.881	0.789	0.946	−4.236	0.997	0.993
	advanced	0.887	0.799	0.952	−4.065	0.998	0.994
R50 Atn. U-Net	none	0.849	0.740	0.920	−5.728	0.997	0.991
	simple	0.902	0.823	0.962	−3.581	0.998	0.995
	advanced	0.902	0.823	0.963	−3.713	0.998	0.995
Swin U-Net	none	0.683	0.524	0.736	−12.45	0.991	0.980
	simple	0.832	0.716	0.907	−5.668	0.998	0.990
	advanced	0.831	0.714	0.906	−5.684	0.998	0.990

(b) Decoder only

Table 2.6. Validation performances on ACDC

model	pretraining	DSC	IoU	mAP	−HD95	AUC	acc.
basic U-Net	none	0.845	0.732	0.927	−22.84	0.987	0.959
	simple	0.844	0.730	0.930	−23.14	0.987	0.959
	advanced	0.846	0.732	0.930	−22.93	0.986	0.960
R50 Atn. U-Net	none	0.836	0.718	0.916	−24.68	0.985	0.956
	simple	0.854	0.744	0.934	−22.20	0.988	0.961
	advanced	0.853	0.744	0.934	−22.38	0.987	0.961
Swin U-Net	none	0.767	0.622	0.847	−31.01	0.971	0.936
	simple	0.832	0.713	0.916	−23.80	0.985	0.955
	advanced	0.819	0.694	0.902	−25.25	0.983	0.951

(a) Full training

model	pretraining	DSC	IoU	mAP	−HD95	AUC	acc.
basic U-Net	none	0.798	0.664	0.879	−30.08	0.977	0.944
	simple	0.809	0.680	0.902	−26.59	0.981	0.951
	advanced	0.807	0.677	0.899	−26.06	0.981	0.950
R50 Atn. U-Net	none	0.778	0.637	0.858	−33.91	0.974	0.937
	simple	0.843	0.728	0.928	−23.46	0.987	0.960
	advanced	0.843	0.729	0.929	−24.23	0.987	0.959
Swin U-Net	none	0.706	0.546	0.767	−34.39	0.951	0.919
	simple	0.822	0.698	0.907	−24.81	0.983	0.952
	advanced	0.826	0.703	0.916	−23.39	0.985	0.955

(b) Decoder only

Table 2.7. Validation performances on COVID-QU

model	pretraining	DSC	IoU	mAP	−HD95	AUC	acc.
basic U-Net	none	0.473	0.310	0.442	−60.00	0.982	0.999
	simple	0.481	0.316	0.448	−59.22	0.983	0.999
	advanced	0.475	0.312	0.447	−60.37	0.985	0.999
R50 Atn. U-Net	none	0.478	0.314	0.444	−60.99	0.980	0.999
	simple	0.468	0.306	0.441	−62.75	0.979	0.999
	advanced	0.477	0.313	0.451	−59.80	0.981	0.999
Swin U-Net	none	0.398	0.249	0.334	−76.32	0.975	0.999
	simple	0.479	0.315	0.443	−62.82	0.992	0.999
	advanced	0.482	0.317	0.444	−61.56	0.989	0.999

(a) Full training

model	pretraining	DSC	IoU	mAP	−HD95	AUC	acc.
basic U-Net	none	0.431	0.275	0.385	−70.99	0.982	0.999
	simple	0.475	0.311	0.448	−64.18	0.987	0.999
	advanced	0.471	0.308	0.446	−61.20	0.986	0.999
R50 Atn. U-Net	none	0.441	0.283	0.400	−68.49	0.982	0.999
	simple	0.478	0.314	0.449	−59.47	0.983	0.999
	advanced	0.476	0.313	0.445	−60.35	0.983	0.999
Swin U-Net	none	0.340	0.205	0.266	−84.00	0.952	0.999
	simple	0.454	0.293	0.407	−67.91	0.989	0.999
	advanced	0.446	0.287	0.398	−68.10	0.989	0.999

(b) Decoder only

Table 2.8. Validation performances on IDRiD

	basic U-Net		**R50 Atn. U-Net**		**Swin U-Net**	
	full training	decoder only	full training	decoder only	full training	decoder only
ACDC	0.908	0.895	0.907	0.904	0.895	0.853
COVID-QU	0.846	0.829	0.853	0.844	0.835	0.827
IDRiD	0.486	0.483	0.494	0.488	0.494	0.464

Table 2.9. Validation performance (Dice index) of the best performing encoder initialization for each model and each dataset

	basic U-Net		**R50 Atn. U-Net**		**Swin U-Net**	
	full training	decoder only	full training	decoder only	full training	decoder only
ACDC	0.992	0.975	0.991	0.935	0.902	0.801
COVID-QU	0.987	0.962	0.979	0.922	0.915	0.854
IDRiD	0.932	0.906	0.924	0.924	0.824	0.740

Table 2.10. Ratio of the downstream scores (Dice indices) of the worst and best performing encoder initialization for each model and each dataset

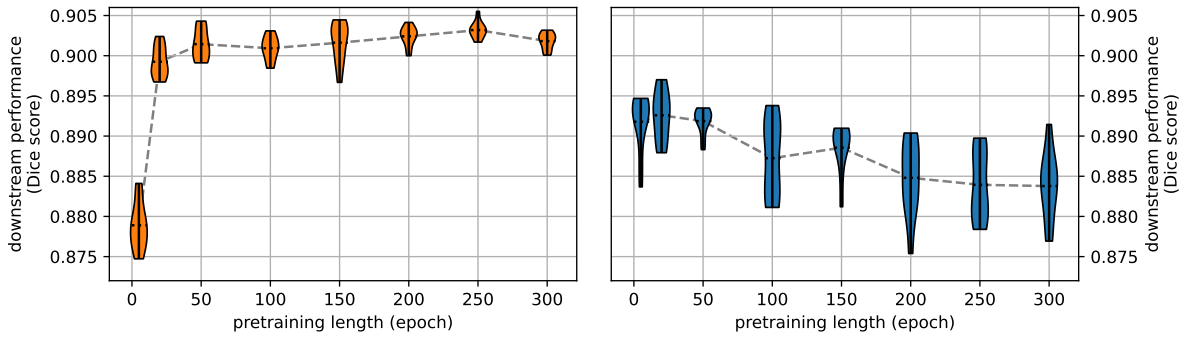
Chapter 3

Transferability metrics

Transfer learning concerns the practice of training a neural network on a large dataset, in order to utilize the learned representation to enhance performance on a smaller downstream dataset. This is done by taking the learned weights of the pretrained network and using them to initialize a model for the target task. The idea behind this process is that the model can learn basic low-level features from the large pretraining dataset, that are relevant for the downstream task as well. Transfer learning regularly leads to faster convergence speed and improved performance on the target task.

Traditionally, pretraining is conducted until convergence in some metric regarding only the pretraining task. However, overlong pretraining can lead to learning dataset- and task specific features, which can compromise the generalization capabilities of the model. This creates the need for a method to find the optimal amount of pretraining that does not impair downstream performance.

In the field of medical image segmentation, the scarcity of high quality labeled training data is a well-known issue. Thus pretraining neural networks on large datasets like ImageNet to then use them as a building block in the segmentation model is a common practice. Pretraining is often helpful in this domain, but overtraining on the source data is also a present danger. An example of two different configurations can be seen in Figure 3.1 for an example where full pretraining, and another one where less ImageNet training provides optimal weights.



(a) ResNet50 Attention U-Net

(b) Basic U-Net

Figure 3.1. Two examples of ImageNet pretraining length influencing downstream performance. Models were trained on the ACDC dataset, with the encoder weights frozen after pretraining on ImageNet. At each encoder checkpoint, violin plots calculated from five downstream trainings are shown

An alternative approach would be to be able to tell at what point a set of encoder weights is optimal as weight initialization for a segmentation network, that would then be trained on a given downstream dataset. This leads to the more general field of **transferability**: given a set of pretrained models, which one would perform the best if further trained on a downstream dataset. If those models are different weights of the same model, obtained from different checkpoints during pretraining on ImageNet, a good transferability metric is suitable for providing an optimal pretraining length a given model needs for a specific downstream dataset.

Transferability and transferability metrics are a relatively new area of research. As a consequence of this, no clear language has been developed for the subject. Section 3.1 provides the notation that will be used throughout the rest of this thesis. Section 3.2 provides a brief overview of existing metrics, using the language developed in Section 3.1. As it will be seen, none of these methods are useful for the goal of measuring transferability of ImageNet-pretrained models for a downstream task of semantic segmentation. This will require a novel approach, which will be introduced in Chapter 4.

3.1 Notion of transferability

Here follows the notation regarding transferability used throughout this thesis. A **task** is defined as a triplet $\mathcal{X} = (X, L, f^*)$, where X is a dataset, L is the possible set of labels for all datapoints, and $f^* : X \to L$ is the ground truth labeling function. A **model** for $\mathcal{X}$ is a function $f : X \to L$ approximating f^*. A **metric** $m_\mathcal{X}$ on $\mathcal{X}$ is a function that maps models to real numbers (scores). Metrics need to be bounded from above or below, with $m_\mathcal{X}(f^*)$ being optimal. For notational simplicity, it will be assumed that all metrics are bounded from above, and assume their maximum at f^*.

Consider a source task $\mathcal{S} = (S, L_\mathcal{S}, f^*_\mathcal{S})$, and a target (downstream) task $\mathcal{T} = (T, L_\mathcal{T}, f^*_\mathcal{T})$ with evaluation metric $m_\mathcal{T}$. We assume that obtaining a model with good performance on $\mathcal{S}$ is straightforward, while doing so on $\mathcal{T}$ is harder. Furthermore, we also assume that for each model f for $\mathcal{S}$, we can derive a model f' for $\mathcal{T}$. Therefore, we want to find a model with good $m_\mathcal{T}$-performance by first obtaining a model f on $\mathcal{S}$ and then deriving f' for $\mathcal{T}$. In the case of this work, this will consist of pretraining a segmentation model encoder on the source task to get f, using these weights to initialize the model, then fine tuning on the segmentation task to get f'.

When discussing transferability, we consider a family of models $\{f_\vartheta\}_{\vartheta\in\Theta}$ for $\mathcal{S}$, indexed by elements of a finite set Θ. The goal is to find $\vartheta \in \Theta$ such that the downstream performance $m_\mathcal{T}(f'_\vartheta)$ is optimal. For this we are looking for a **transferability indicator** $\varrho_{\mathcal{S},\mathcal{T}}(f_\vartheta)$, and define an associated **transferability indicator score** (TIS) of $\varrho_{\mathcal{S},\mathcal{T}}$ on Θ as

$$\mathrm{TIS}(\varrho_{\mathcal{S},\mathcal{T}}, \Theta) = \frac{m_\mathcal{T}(f'_{\vartheta^*})}{\max\limits_{\vartheta\in\Theta}\big(m_\mathcal{T}(f'_\vartheta)\big)}, \tag{3.1}$$

where

$$\vartheta^* = \operatorname*{argmax}_{\vartheta\in\Theta}\big(\varrho_{\mathcal{S},\mathcal{T}}(f_\vartheta)\big).$$

This score measures the relative downstream performance of the model chosen based on $\varrho_{\mathcal{S},\mathcal{T}}$ to the best performing downstream model. It reaches 1 when $\varrho_{\mathcal{S},\mathcal{T}}$ predicts the highest score for the model with the best $m_\mathcal{T}$-performance. (In the case of searching for the optimal length of ImageNet-pretraining, Θ consists of model weights saved at different times during pretraining.)

3.2 Overview of transferability metrics in literature

Transferability metrics for task and domain shift in deep learning has been a popular topic in recent years. This section provides an overview of existing transferability metrics and measures. As we will show, none of the below methods – and, to the best of our knowledge, no other existing methods – are fit for estimating transferability from the ImageNet classification task to a medical image segmentation task.

Task transfer

In general literature, a common usage of transferability metrics is task transfer, where the source task $\mathcal{S} = (S, L_{\mathcal{S}}, f^*_{\mathcal{S}})$ and target task $\mathcal{T} = (T, L_{\mathcal{T}}, f^*_{\mathcal{T}})$ share the same data (or at least they share the same modality, i.e., S and T have similar distributions), but have differing labels. Examples include methods based on taskonomy [23] (tax taskonomy) and DEPARAs [36] (deep attribution graphs), which predict transferability between tasks, thus cutting down on total training time in the case there is a large set of tasks and models are needed for all of them. However, these approaches do not work when the domains are different, like in the case of natural versus medical images. Furthermore, both examples require already trained models for each task to map it to a feature space, therefore they are computationally costly.

If the sets of possible ground truth labels $L_{\mathcal{S}}$ and $L_{\mathcal{T}}$ are both finite (i.e. if both $\mathcal{S}$ and $\mathcal{T}$ are classification tasks), then one can estimate the negative conditional entropy [30] (NCE) of the target labels given the source labels

$$-H(L_{\mathcal{T}}|L_{\mathcal{S}}) = \sum_{t\in L_{\mathcal{T}}}\sum_{s\in L_{\mathcal{S}}} \mathbb{P}\big(f^*_{\mathcal{S}}(x)=s, f^*_{\mathcal{T}}(x)=t\big)\log \mathbb{P}\big(f^*_{\mathcal{T}}(x)=t\big|f^*_{\mathcal{S}}(x)=s\big). \tag{3.2}$$

This value can then be used as an approximation of transferability from $\mathcal{S}$ to $\mathcal{T}$, in the transfer learning scenario where a model f is trained on $\mathcal{S}$, then its backbone is frozen, replaced by another classification head, and trained on $\mathcal{T}$.

NCE is a model agnostic approach. This solves the issue with taskonomy and DEPARAs where models need to be pretrained on all tasks, but it also means that it cannot select the best $\mathcal{S}$-pretraind model for $\mathcal{T}$.

LEEP

If the sets of possible ground truth labels $L_{\mathcal{S}}$ and $L_{\mathcal{T}}$ are both finite (i.e. if both $\mathcal{S}$ and $\mathcal{T}$ are classification tasks), then one can estimate transferability based on the $L_{\mathcal{S}}$-predictions of an $\mathcal{S}$-pretrained model on $\mathcal{T}$, and the true labels. For this, given an $\mathcal{S}$-pretrained model f that predicts a distribution over $L_{\mathcal{S}}$, the joint distribution $\mathbb{P}\big(f(x)=s, f^*_{\mathcal{T}}(x)=t\big)$ $(x\in T, s\in L_{\mathcal{S}}, t\in L_{\mathcal{T}})$ can be estimated as

$$\hat{p}(s,t) = \frac{1}{|T|}\sum_{f^*_{\mathcal{T}}(x)=t} f(x)_s, \tag{3.3}$$

where $f(x)_s$ is the probability model f assigns to class s. (In standard classification tasks, $f(x)$ will be an $|L_{\mathcal{S}}|$ nonnegative long vector whose entries sum to 1, and $f(x)_s$ will be its entry at coordinate s.)

Similarly, the marginal distribution $\mathbb{P}\big(f(x)=s\big)$ can be estimated as

$$\hat{p}(s) = \frac{1}{|T|}\sum_{x\in T} f(x)_s, \tag{3.4}$$

and from here the conditional distribudion $\mathbb{P}\big(f^*_{\mathcal{T}}(x) = t \big| f(x) = s\big)$ can be estimated as

$$\hat{p}(t|s) = \frac{\hat{p}(s,t)}{\hat{p}(s)}. \tag{3.5}$$

From here we can define the **expected empirical predictor** (EEP) classifier, that, for a given $x \in T$, first draws a "dummy" label from $L_{\mathcal{S}}$ according to the distribution given by $f(x)$, then draws $t \in L_{\mathcal{T}}$ from the distribution $\hat{p}(t|s)$. From this predictor, the average log likelihood of the EEP (LEEP [37]) can be calculated as a transferability indicator, formally described as

$$\varrho_{\mathcal{S},\mathcal{T}}(f) = \frac{1}{|T|} \sum_{x \in T} \log \left(\sum_{s \in L_{\mathcal{S}}} \hat{p}\big(f^*_{\mathcal{T}}(x) \big| s\big) f(x)_s \right). \tag{3.6}$$

While LEEP is easy to compute and is proven to be effective both in theory and practice, the fact that it only works on classification tasks is a serious drawback. $\mathcal{N}$LEEP [41] aims to solve this by replacing $L_{\mathcal{S}}$ in Equation 3.6 by an abstract set of clusters V, and then fitting a Gaußian mixture model $f_{\mathcal{N}}$ with $|V|$ components on features extracted from $\mathcal{T}$ by f. Then the probabilities $f(x)_s$ ($s \in L_{\mathcal{S}}$) in Equations 3.3–3.6 are replaced with the probabilites $f_{\mathcal{N}}(x)_v$ ($v \in V$).

$\mathcal{N}$LEEP thus provides a solution for transferability estimation where $\mathcal{S}$ is not a classification task – e.g., if the models are pretrained in an unsupervised or self-supervised fashion. However, it still requires a finite set of labels in $L_{\mathcal{T}}$. While in the case of semantic segmentation this is theoretically true – a segmentation mask of C classes with size $H \times W$ has "only" C^{HW} possible values –, the possible set of labels is large enough to practically be regarded as infinite. Furthermore, the marginal distribution $\mathbb{P}\big(f^*_{\mathcal{T}}(x) = t\big)$, and therefore its derived joined and conditional distributions, can not be meaningfully estimated, since most possible mask values t appear either one or zero times in the dataset T.

($\mathcal{N}$LEEP offers a solution for this, which is to convert a segmentation – or object detection – task to a classification task for the sake of Equations 3.3–3.6 by assigning to an image each class that it contains. This is however still not a feasible method for binary segmentation tasks, or datasets where most images contain most classes – like ACDC.)

LogME

Regression can be viewed as a generalization of classification, since classification tasks can also be stated as regression tasks where the objectives are the one-hot ground truth labels. Thus, a transferability indicator that works for regression tasks is in theory preferable to one that only works on classification tasks.

LogME [43] (logarithmic maximum evidence) considers an $\mathcal{S}$-pretrained model f, with derived feature extractor $\bar{f} : T \to \mathbb{R}^d$. Then, given parameters $\alpha, \beta \in \mathbb{R}$, from $y = f^*_{\mathcal{T}}(T)$ and $\hat{y} = \bar{f}(T)$, we get

$$\ell(\alpha, \beta) = \frac{1}{2} \log \frac{\beta}{2\pi} + \frac{d}{2|T|} \log \frac{\alpha}{2\pi} + \frac{1}{|T|} \log \int e^{-\frac{\alpha}{2} w^\top w - \frac{\beta}{2} \|\hat{y} w - y\|_2^2} \, \mathrm{d}w. \tag{3.7}$$

$\ell(\alpha, \beta)$ can be calculated numerically, and from that an optimal (α^*, β^*) can be estimated through numerical methods. The approximation of $\ell(\alpha^*, \beta^*)$ is used as the transferability indicator dubbed LogME. In the case of multivariate regression problems (including

multiclass classification, as discussed above), each variable is treated separately, with a separate LogME score calculated, then the average is taken.

Note that while LogME offers transferability estimation from an arbitrary source task to target tasks more general than classification, it does not solve the problem that LEEP had when the target task is semantic segmentation.

Chapter 4

Robustness

This chapter defines the main contribution of this work, which is a novel transferability indicator based on the robustness of the target data representation of a pretrained model. Section 4.1 provides a robustness definition, based on contrastive learning. Based on this definition, we evaluated the derived transferability indicator on the medical segmentation experiments described in Section 2.4. Section 4.2 shows these results.

4.1 Defining robustness

Contrastive learning

Contrastive methods – first appearing in [1] – aim to train a model to represent the structure inherent to a dataset, without comparing its output to a set of labels in a supervised manner. Instead, the task is to distinguish between related and unrelated pairs of datapoints. What constitutes as two related datapoints varies based on the data, task, and information available. In the case of image processing, it can be differently augmented versions of the same image, while in NLP they can be nearby text chunks in next sentence prediction [28]. If classification data is available, two datapoints might be considered related if they belong to the same class [38].

Due to its usefulness in self-supervised learning, this approach has grown popular in recent years. A common contrastive learning method is to use a triplet loss function [13], which takes triplets of queries, positive keys, and negative keys, denoted by q, k^+, and k^- respectively. Similar representations of q and k^+ are rewarded, as well as distinct representations of q and k^-. The triplet loss is defined as

$$\mathcal{L}_{\text{triplet}}\big(f(q), f(k^+), f(k^-)\big) = \max\big(0, d\big(f(q), f(k^+)\big) - d\big(f(q), f(k^-)\big) + \varepsilon\big), \tag{4.1}$$

where d is a distance measure, and $\varepsilon > 0$ is a margin parameter.

Robustness as a transferability metric

Further on, we restrict ourselves to the case where the source task $\mathcal{S}$ is classification on ImageNet, and the target task $\mathcal{T}$ is a medical image segmentation problem, with the Dice index as the evaluation metric $m_{\mathcal{T}}$. Our direct goal is to provide a transferability indicator for the encoder weights at different steps during pretraining, thus avoiding overtraining, and potentially decreasing pretraining time.

Therefore we define Θ as a set of checkpoints, where f_ϑ is a model pretrained on ImageNet for ϑ epochs. From a pretrained encoder f_ϑ, we obtain f'_ϑ by initializing the

encoder weights of the segmentation model with the pretrained encoder backbone and then fine-tuning it on $\mathcal{T}$.

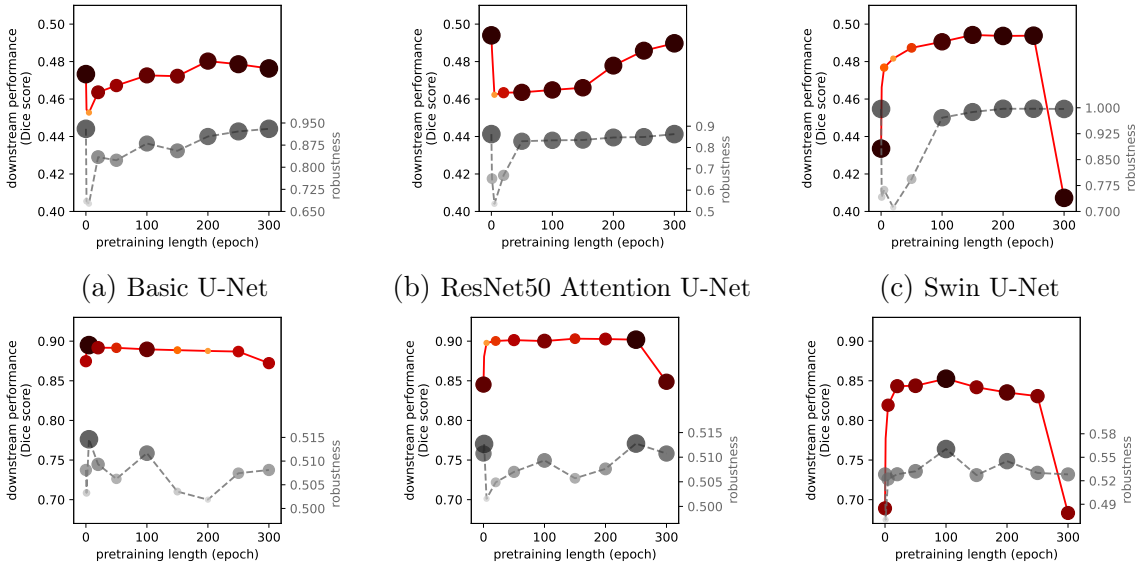
The choice of ϑ can have a significant effect on downstream preformance $m_{\mathcal{T}}$, as seen in Subfigure 3.1a. However, using the accuracy on ImageNet as a transferability indicator is also ill-advised, as evidenced by Subfigure 3.1b.

Instead, we propose to use a contrastive learning inspired indicator $\varrho_{\mathcal{S},\mathcal{T}}$, based on Equation 4.1. For each q_i target datapoint we obtain k_i^+ by applying a set of random augmentations for q_i, and let k_i^- be a randomly chosen and augmented other target datapoint. We define the **robustness** of the representation of model f_ϑ as

$$\varrho_{\mathcal{S},\mathcal{T}}(f_\vartheta) = 1 - \frac{1}{|T|} \sum_{x_i \in T} \max\Big(0, d\big(\hat{f}_\vartheta(\tilde{q}_i), \hat{f}_\vartheta(k_i^+)\big) - d\big(\hat{f}_\vartheta(\tilde{q}_i), \hat{f}_\vartheta(k_i^-)\big) + \varepsilon\Big), \tag{4.2}$$

where T is the target dataset, d is a distance measure, $\tilde{q}$ is an augmented version of q, and $\hat{f}_\vartheta$ obtains a representation by extracting it from an inner layer of f_ϑ. The precise choice of these parameters is discussed in Section 4.2.

We present two applications of the robustness defined above as a transferability indicator. Our first approach is to measure robustness at regular intervals during pretraining, and in the end use the weights corresponding to the highest score to transfer to the target task. This is an **offline** method, since best weights can still only be obtained after a full pretraining. We also suggest an **online** method, where the robustness score acts as an early stopping condition for pretraining. This approach potentially only finds local optima, thus providing less improvement in performance than the offline version, but prevents unnecessary pretraining steps.



(a) Basic U-Net (b) ResNet50 Attention U-Net (c) Swin U-Net

(d) Basic U-Net (e) ResNet50 Attention U-Net (f) Swin U-Net

Figure 4.1. Robustness compared to performance on a downstream segmentation dataset. A larger and darker dot indicates higher robustness. Under each graph, the exact robustness scores are indicated by dashed gray lines. All encoders were pretrained using the "advanced" scheme. Subfigures (a)–(c) are fully trained on IDRiD. Subfigures (d)–(f) are trained on ACDC, with the encoder weights frozen during training.

4.2 Results

During our research, we experimented with different choices of d, ε, and $\hat{f}_\vartheta$ in Equation 4.2. The highest scores resulted from using the cosine distance

$$d(q,k) = 1 - \frac{qk}{\|q\|\|k\|} \tag{4.3}$$

for the distance metric when calculating the robustness, with a margin of $\varepsilon = 0.5$. We obtained $\tilde{q}$, k^+, and k^- by applying random jittering to the hue, saturation, brightness, and contrast values, as described in Algorithm 2.3. The representation $\hat{f}_\vartheta(\cdot)$ was taken from the second-to-last level of the encoder. When we were only planning to train the decoder, a pooled representation was taken; for full training, we did not pool.

We report the transferability indicator (or relative accuracy) scores – see Equation 3.1 – using the Dice index as the downstream evaluation metric. (As can be seen in Figure 1.6, choosing another reasonable metric would not have influenced our results significantly.) We compare the TIS of our robustness indicator to the two common-practice baselines: no pretraining, and performing a full pretraining on ImageNet. (The latter can also be understood as using the ImageNet accuracy as a transferability indicator.) As discussed in Section 3.2, apart from these naïve methods, to the best of our knowledge, no other method to choose the best-transferring model, can be utilized for transferring from ImageNet to a medical image segmentation task.

	ACDC		**COVID-QU**		**IDRiD**	
	full training	decoder only	full training	decoder only	full training	decoder only
robustness (offline)	**0.989**	**0.999**	**0.996**	**0.986**	**0.991**	**0.984**
robustness (online)	0.984	0.955	0.982	0.956	0.959	0.948
ImageNet accuracy	0.982	0.950	0.982	0.953	0.963	0.930
no pretraining	0.973	0.909	0.967	0.921	0.945	0.866

(a) Average transferability indicator scores on each dataset

	basic U-Net		**R50 Atn. U-Net**		**Swin U-Net**	
	full training	decoder only	full training	decoder only	full training	decoder only
robustness (offline)	0.996	**0.986**	**0.990**	**0.996**	**0.990**	**0.987**
robustness (online)	0.978	0.985	0.989	0.953	0.958	0.921
ImageNet accuracy	**0.998**	0.969	0.989	0.963	0.940	0.900
no pretraining	0.994	0.958	0.990	0.929	0.901	0.809

(b) Average transferability indicator scores of each model

Table 4.1. Comparison of the TIS of our robustness scores, ImageNet accuracy, and training from scratch, averaged across pretraining schemes, and three models (a) or three datasets (b)

Across all our setups, our robustness score produced a TIS of over 98% on average calculated offline, and over 97% calculated online. We report a worst-case performance of over 95% offline, and 91% online. As a comparison, both no pretraining and ImageNet

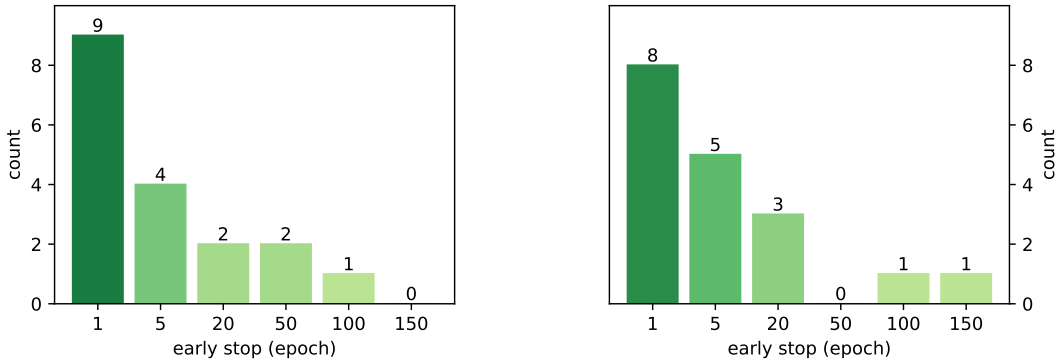
pretraining can provide under 75% TIS. Even at its worst, our offline method shows a 0.006 absolute decrease in Dice index, or a 1.3% relative decrease, compared to full ImageNet pretraining.

Figure 4.1 shows examples of training setups with downstream performances and robustness indicated for each checkpoint. Note that offline robustness provides good predictions in various circumstances, including ones where downstream performance monotonically increases with pretraining time up to epoch 250 (Subfigure 4.1c), and ones where the clear optimum is at a much shorter, 100 epoch long pretraining (Subfigure 4.1f).

Table 4.1 shows our results, compared to full pretraining and no pretraining. For each model, we had three datasets and two pretraining schemes; we averaged these scores out over the model and pretraining scheme axis in Subtable 4.1a, and the dataset and pretraining schemes in Subtable 4.1b.

We note that randomly initialized encoders often provide really high robustness scores, even when that does not correlate with a higher downstream performance. For this reason, we calculated online scores by disregarding the randomly initialized model, and only starting to calculate robustness after the first epoch. We find that using robustness as an online predictor provides similar results to always using a fully pretrained network – with only a fraction of the pretraining time –, and is almost always better than no pretraining.

Figure 4.2 shows how many times each epoch index was predicted as best in an online manner. Out of the 18 configurations (six ImageNet pretraining, each evaluated for three datasets), no online method would have trained for longer than 150 epochs (as opposed to the full ImageNet pretraining, which is 300 epochs), and most would have stopped after only one epoch – while still providing better initial weights than random initialization.



(a) Pooled representation

(b) Unpooled representation

Figure 4.2. Counts of predicted best epoch indices, when using an online method – i.e., the first local maxima of the robustness score, disregarding that of the randomly initialized model

Certain dataset–model configurations tend to provide the same performance, regardless of encoder weights. With these setups, no major improvement is needed over training from scratch. In contrast, experiments run on IDRiD (which is the hardest dataset – see Table 2.9), or using Swin U-Net (which is the model where pretraining tends to have the largest impact – see Table 2.10) show that both a from-scratch initialization and a full pretraining can result in a significantly worse downstream performance than what could be achieved with the encoder weights initialized from the right checkpoint.

Table 4.4 shows only results on IDRiD, with the encoders pretrained using the advanced scheme. In that example, using full or no pretraining for Swin U-Net provides

under 90% relative accuracy for the full training, and around 75% relative accuracy when only the encoder is trained; compared to that, robustness leads to an over 99% relative accuracy in both cases.

Investigating other parameters

Besides the cosine distance, we investigated other distance metrics for d – namely the (inverse of the) L^2 distance and the Pearson correlation coefficient (PCC). We found that the cosine distance works consistently best across all examined datasets. The other two metrics also provided better transferability indicator scores than either the ImageNet accuracy or training from scratch.

We find that the margin parameter ε does not influence performance for the most part. Switching from 0.5 to 0.25 has practically no effect. Going above 0.5 leads to a slight average decrease, and rarely to a more significant decrease.

When using an encoder module as a feature extractor, it is natural to make use of the whole model – that is, use the output of the last layer as the input's embedding. However, we found that using the last level instead of the second to last one leads to worse performance on average. We theorize this is because the last level learns dataset-specific features, which do not transform well to the downstream task.

Another question is whether or not to apply pooling over the spatial dimensions of the embedded image. We found that when the full model was trained on the downstream task, no pooling was slightly better, but when only the encoder was trained, it did not help the average performance. In this case, we reported our results with the use of pooling, as that would be preferable in practice, due to a lowered computational cost.

Tables 4.2–4.4 list the detailed results for each configuration.* When a parameter (encoder level, pooling, distance metric, or margin) is not indicated, it was set as described in Section 4.2. The figures of the final method – that of cosine distance with a margin of 0.5 – are outlined with a black frame.

Limitations

Naturally, our work is not without its limitations. Mainly, the robustness of a randomly initialized encoder is often bigger than that of a model pretrained for a few epochs, which presents a challenge when using robustness as an early stopping condition. In this work, we elected to disregard randomly initialized weights, which leads to good average performance, but it also means that whenever no pretraining would lead to optimal downstream performance – which can happen –, our online method fails to recognize that. We note that this phenomenon can be counteracted by also training a segmentation network from scratch, but this would require additional training time.

The online method is also generally worse than the offline method, indicating that it is prone to find local maxima. More generally, the Spearman correlation of robustness and downstream performance is relatively low, even if their optima coincide.

We also note that generally robustness is a better indicator when the encoder weights are not modified during training for the downstream task, but this setup also results in worse performance on average, and is therefore less widely used in practice.

*Individual robustness scores for all combinations of investigated values of d, ε, and $\hat{f}_\vartheta$ can be found at `https://github.com/aielte-research/MedSegPretrainImageNet/blob/main/results/robustness_scores.csv`.

		basic U-Net		**R50 Atn. U-Net**		**Swin U-Net**	
		full training	decoder only	full training	decoder only	full training	decoder only
ImageNet accuracy		0.999	0.987	1.000	0.998	1.000	0.997
no pretraining		1.000	0.978	0.992	0.939	0.905	0.818
level	**pooled**						
last	yes	0.994	1.000	0.999	1.000	0.905	0.818
	no	0.997	0.995	0.999	1.000	0.998	1.000
second to last	yes	0.992	0.998	0.999	1.000	0.998	0.997
	no	0.996	0.990	0.999	1.000	0.998	1.000
metric	**margin**						
PCC	0.50	0.996	0.998	0.992	0.936	0.998	0.818
L^2 distance		0.992	0.981	0.995	0.971	0.956	0.948
cosine distance	0.25	0.996	0.998	0.999	1.000	0.998	0.967
	0.50	0.996	0.998	0.999	1.000	0.998	0.997
	0.75	0.996	0.998	0.999	1.000	0.998	0.997
	1.00	0.996	0.998	0.999	1.000	0.998	0.997

(a) Simple pretraining

		basic U-Net		**R50 Atn. U-Net**		**Swin U-Net**	
		full training	decoder only	full training	decoder only	full training	decoder only
ImageNet accuracy		1.000	0.975	0.992	0.940	0.902	0.801
no pretraining		0.996	0.977	0.992	0.936	0.952	0.808
level	**pooled**						
last	yes	0.996	0.977	1.000	0.999	0.998	0.987
	no	0.996	0.977	1.000	0.999	1.000	0.980
second to last	yes	0.996	1.000	1.000	0.999	0.999	1.000
	no	0.996	0.994	0.992	0.975	0.952	0.808
metric	**margin**						
PCC	0.50	0.996	1.000	0.992	0.999	0.952	0.980
L^2 distance		0.997	0.998	0.996	0.997	0.952	0.989
cosine distance	0.25	0.996	1.000	0.992	0.999	1.000	1.000
	0.50	0.996	1.000	0.992	0.999	0.952	1.000
	0.75	0.996	1.000	0.992	0.999	0.952	1.000
	1.00	0.996	1.000	0.992	0.999	0.952	1.000

(b) Advanced pretraining

Table 4.2. Detailed results on ACDC

		basic U-Net		**R50 Atn. U-Net**		**Swin U-Net**	
		full training	decoder only	full training	decoder only	full training	decoder only
ImageNet accuracy		0.998	0.976	1.000	0.999	0.998	1.000
no pretraining		1.000	0.962	0.980	0.923	0.920	0.859
level	**pooled**						
last	yes	0.990	0.973	0.993	1.000	0.975	0.943
	no	0.990	0.973	0.993	1.000	0.975	0.943
second to last	yes	0.997	0.979	1.000	0.999	0.988	0.970
	no	0.998	0.976	0.993	0.995	1.000	0.998
metric	**margin**						
PCC	0.50	1.000	0.985	0.993	0.928	1.000	0.943
L^2 distance		1.000	0.990	0.989	0.928	0.975	0.925
cosine distance	0.25	1.000	0.985	0.993	0.997	1.000	0.988
	0.50	1.000	0.985	0.993	0.995	1.000	0.988
	0.75	1.000	0.985	0.993	0.995	1.000	0.988
	1.00	1.000	0.985	0.993	0.995	1.000	0.988

(a) Simple pretraining

		basic U-Net		**R50 Atn. U-Net**		**Swin U-Net**	
		full training	decoder only	full training	decoder only	full training	decoder only
ImageNet accuracy		0.999	0.968	0.980	0.922	0.919	0.854
no pretraining		1.000	0.984	0.980	0.927	0.919	0.872
level	**pooled**						
last	yes	0.997	0.985	0.995	0.996	0.990	1.000
	no	1.000	0.980	0.996	0.995	1.000	0.985
second to last	yes	1.000	0.980	0.995	0.996	0.989	0.993
	no	0.999	0.968	0.996	0.995	0.990	1.000
metric	**margin**						
PCC	0.50	0.999	0.985	0.980	0.955	0.990	0.993
L^2 distance		0.993	0.999	0.992	0.955	0.993	0.980
cosine distance	0.25	0.999	0.985	0.980	0.996	0.990	0.993
	0.50	0.999	0.980	0.996	0.996	0.990	0.993
	0.75	0.999	0.997	0.980	0.996	0.919	0.993
	1.00	0.999	0.997	0.980	0.996	0.919	0.993

(b) Advanced pretraining

Table 4.3. Detailed results on COVID-QU

		basic U-Net		R50 Atn. U-Net		Swin U-Net	
		full training	decoder only	full training	decoder only	full training	decoder only
ImageNet accuracy		1.000	0.997	0.973	0.999	1.000	1.000
no pretraining		0.981	0.906	0.994	0.925	0.830	0.740
level	**pooled**						
last	yes	0.941	0.969	0.949	0.988	0.830	0.740
	no	0.999	1.000	0.994	0.925	0.830	0.740
second to last	yes	0.941	0.969	0.935	1.000	0.994	0.970
	no	0.999	1.000	0.960	0.972	1.000	0.998
metric	**margin**						
PCC	0.50	0.981	0.947	0.960	0.925	1.000	0.938
L^2 distance		0.963	0.967	0.978	0.932	0.830	0.909
cosine distance	0.25	0.981	0.995	0.994	0.925	0.830	0.938
	0.50	0.999	0.969	0.960	1.000	1.000	0.970
	0.75	0.981	0.995	0.927	0.925	1.000	0.745
	1.00	0.981	0.995	0.927	0.925	1.000	0.745

(a) Simple pretraining

		basic U-Net		R50 Atn. U-Net		Swin U-Net	
		full training	decoder only	full training	decoder only	full training	decoder only
ImageNet accuracy		0.992	0.910	0.991	0.924	0.824	0.748
no pretraining		0.986	0.941	1.000	0.926	0.877	0.758
level	**pooled**						
last	yes	0.984	0.991	1.000	0.926	0.993	0.993
	no	0.986	0.941	1.000	0.926	0.999	0.990
second to last	yes	0.984	0.991	0.938	0.983	0.993	0.993
	no	0.986	0.941	1.000	0.926	0.999	0.990
metric	**margin**						
PCC	0.50	0.986	0.969	0.941	0.983	0.999	0.929
L^2 distance		0.943	0.977	0.938	0.953	0.965	0.975
cosine distance	0.25	0.986	1.000	1.000	0.983	0.965	0.993
	0.50	0.986	0.991	1.000	0.983	0.999	0.993
	0.75	0.986	1.000	0.938	0.983	0.999	0.993
	1.00	0.986	1.000	0.938	0.983	0.999	0.993

(b) Advanced pretraining

Table 4.4. Detailed results on IDRiD

Bibliography

[1] Chopra, S., Hadsell, R. & LeCun, Y. Learning a similarity metric discriminatively, with application to face verification. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) **1** (San Diego, California, United States of America, June 2005), pp. 539–546. DOI:10.1109/CVPR.2005.202.

[2] Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems **1** (Lake Tahoe, Nevada, United States of America, December 2012), pp. 1097–1105.

[3] Cireşan, D. C., Giusti, A., Gambardella, L. M. & Schmidhuber, J. Deep neural networks segment neuronal membranes in electron microscopy images. In: Proceedings of the 25th International Conference on Neural Information Processing Systems **2** (Lake Tahoe, Nevada, United States of America, December 2012), pp. 2843–2851.

[4] Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations (San Diego, California, United States of America, May 2015).

[5] Simonyan, K. & Zisserman, A. Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations (San Diego, California, United States of America, May 2015).

[6] Shelhamer, E., Long, J. & Darrell, T. Fully convolutional networks for semantic segmentation. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Boston, Massachusetts, United States of America, June 2015), pp. 640–651. DOI:10.1109/CVPR.2015.7298965.

[7] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. & Rabinovich, A. Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Boston, Massachusetts, United States of America, June 2015), pp. 1–9. DOI:10.1109/CVPR.2015.7298594.

[8] Ronneberger, O., Fischer, P. & Brox, T. U-Net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015 **3** (Munich, Germany, November 2015), pp. 234–241. DOI:10.1007/978-3-319-24574-4_28.

[9] He, K., Zhang, X., Ren, S. & Sun, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In: 2015 IEEE International Conference on Computer Vision (ICCV) (Santiago, Chile, December 2015), pp. 1026–1034. DOI:10.1109/ICCV.2015.123.

[10] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. & Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* **115,** pp. 211–252. DOI:10.1007/s11263-015-0816-y (December 2015).

[11] Ba, J. L., Kiros, J. R. & Hinton, G. E. Layer normalization. June 2016. arXiv: 1607.06450v1 [stat.ML].

[12] He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Las Vegas, Nevada, United States of America, June 2016), pp. 770–778. DOI:10.1109/CVPR.2016.90.

[13] Oh Song, H., Xiang, Y., Jegelka, S. & Savarese, S. Deep metric learning via lifted structured feature embedding. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Las Vegas, Nevada, United States of America, June 2016), pp. 4004–4012. DOI:10.1109/CVPR.2016.434.

[14] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. Rethinking the Inception architecture for computer vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Las Vegas, Nevada, United States of America, June 2016), pp. 2818–2826. DOI:10.1109/CVPR.2016.308.

[15] Huang, G., Sun, Y., Liu, Z., Sedra, D. & Weinberger, K. Q. Deep networks with stochastic depth. In: Computer Vision – ECCV 2016 **4** (Amsterdam, The Netherlands, October 2016), pp. 646–661. DOI:10.1007/978-3-319-46493-0_39.

[16] Loshchilov, I. & Hutter, F. SGDR: Stochastic gradient descent with warm restarts. In: 5th International Conference on Learning Representations **1** (Toulon, France, April 2017), pp. 1769–1784.

[17] Sudre, C. H., Li, W., Vercauteren, T., Ourselin, S. & Jorge Cardoso, M. Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations. In: Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support (Québec City, Quebec, Canada, September 2017), pp. 240–248. DOI:10.1007/978-3-319-67558-9_28.

[18] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems (Long Beach, California, United States of America, December 2017), pp. 6000–6010.

[19] Dumoulin, V. & Visin, F. A guide to convolution arithmetic for deep learning. January 2018. arXiv: 1603.07285v2 [stat.ML].

[20] Zhang, H., Cisse, M., Dauphin, Y. N. & Lopez-Paz, D. mixup: Beyond empirical risk minimization. In: 6th International Conference on Learning Representations (Vancouver, British Columbia, Canada, May 2018).

[21] Zhang, Z., Liu, Q. & Wang, Y. Road extraction by deep residual U-Net. *IEEE Geoscience and Remote Sensing Letters* **15,** pp. 749–753. DOI:10.1109/LGRS.2018.2802944 (May 2018).

[22] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y. & He, K. Accurate, large minibatch SGD: Training ImageNet in 1 hour. June 2018. arXiv: 1706.02677v2 [cs.CV].

[23] Zamir, A. R., Sax, A., Shen, W., Guibas, L. J., Malik, J. & Savarese, S. Taskonomy: Disentangling task transfer learning. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (Salt Lake City, Utah, United States of America, June 2018), pp. 3712–3722. DOI:10.1109/CVPR.2018.00391.

[24] Bernard, O., Lalande, A., Zotti, C., Cervenansky, F., Yang, X., Heng, P.-A., Cetin, I., Lekadir, K., Camara, O., Ballester, M. A. G., *et al.* Deep learning techniques for automatic MRI cardiac multi-structures segmentation and diagnosis: is the problem solved? *IEEE Transactions on Medical Imaging* **37,** pp. 2514–2525. DOI:10.1109/TMI.2018.2837502 (November 2018).

[25] Schlemper, J., Oktay, O., Schaap, M., Heinrich, M., Kainz, B., Glocker, B. & Rueckert, D. Attention gated networks: Learning to leverage salient regions in medical images. *Medical Image Analysis* **53,** pp. 197–207. DOI:10.1016/j.media.2019.01.012 (April 2019).

[26] Loshchilov, I. & Hutter, F. Decoupled weight decay regularization. In: 7th International Conference on Learning Representations (New Orleans, Louisiana, United States of America, May 2019).

[27] Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V. & Le, Q. V. AutoAugment: Learning augmentation strategies from data. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (Long Beach, California, United States of America, June 2019), pp. 113–123. DOI:10.1109/CVPR.2019.00020.

[28] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies **1** (Minneapolis, Minnesota, United States of America, June 2019), pp. 4171–4186. DOI:10.18653/v1/N19-1423.

[29] He, K., Girshick, R. & Dollár, P. Rethinking ImageNet pre-training. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (Seoul, South Korea, October 2019), pp. 4918–4927. DOI:10.1109/ICCV.2019.00502.

[30] Tran, A. T., Nguyen, C. V. & Hassner, T. Transferability and hardness of supervised classification tasks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (Seoul, South Korea, October 2019), pp. 1395–1405. DOI:10.1109/ICCV.2019.00148.

[31] Yun, S., Han, D., Chun, S., Oh, S. J., Yoo, Y. & Choe, J. CutMix: Regularization strategy to train strong classifiers with localizable features. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV) (Seoul, South Korea, October 2019), pp. 6022–6031. DOI:10.1109/ICCV.2019.00612.

[32] Porwal, P., Pachade, S., Kokare, M., Deshmukh, G., Son, J., Bae, W., Liu, L., Wang, J., Liu, X., Gao, L., Wu, T., Xiao, J., Wang, F., Yin, B., Wang, Y., Danala, G., He, L., Choi, Y. H., Lee, Y. C., Jung, S.-H., Li, Z., Sui, X., Wu, J., Li, X., Zhou, T., Toth, J., Baran, A., Kori, A., Chennamsetty, S. S., Safwan, M., Alex, V., Lyu, X., Cheng, L., Chu, Q., Li, P., Ji, X., Zhang, S., Shen, Y., Dai, L., Saha, O., Sathish, R., Melo, T., Araújo, T., Harangi, B., Sheng, B., Fang, R., Sheet, D., Hajdu, A., Zheng, Y., Mendonça, A. M., Zhang, S., Campilho, A., Zheng, B., Shen, D., Giancardo, L., Quellec, G. & Mériaudeau, F. IDRiD: Diabetic retinopathy – Segmentation and grading challenge. *Medical Image Analysis* **59,** pp. 101561. DOI:10.1016/j.media.2019.101561 (January 2020).

[33] Punn, N. S. & Agarwal, S. Inception U-Net architecture for semantic segmentation to identify nuclei in microscopy cell images. *ACM Transactions on Multimedia Computing, Communications, and Applications* **16,** pp. 1–15. DOI:10.1145/3376922 (February 2020).

[34] Zhong, Z., Zheng, L., Kang, G., Li, S. & Yang, Y. Random erasing data augmentation. In: Proceedings of the AAAI Conference on Artificial Intelligence **34** (New York, New York, United States of America, February 2020), pp. 13001–13008. DOI:10.1609/aaai.v34i07.7000.

[35] Cubuk, E. D., Zoph, B., Shlens, J. & Le, Q. V. RandAugment: Practical automated data augmentation with a reduced search space. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (Seattle, Washington, United States of America, June 2020), pp. 3008–3017. DOI:10.1109/CVPRW50498.2020.00359.

[36] Song, J., Chen, Y., Ye, J., Wang, X., Shen, C., Mao, F. & Song, M. DEPARA: Deep attribution graph for deep knowledge transferability. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (Seattle, Washington, United States of America, June 2020), pp. 3922–3930. DOI:10.1109/CVPR42600.2020.00398.

[37] Nguyen, C., Hassner, T., Seeger, M. & Archambeau, C. LEEP: A new measure to evaluate transferability of learned representations. In: Proceedings of the 37th International Conference on Machine Learning (July 2020), pp. 7294–7305.

[38] Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C. & Krishnan, D. Supervised contrastive learning. In: Proceedings of the 34th International Conference on Neural Information Processing Systems (December 2020), pp. 18661–18673.

[39] Chen, J., Lu, Y., Yu, Q., Luo, X., Adeli, E., Wang, Y., Lu, L., Yuille, A. L. & Zhou, Y. TransUNet: Transformers make strong encoders for medical image segmentation. February 2021. arXiv: 2102.04306v1 [cs.CV].

[40] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. & Houlsby, N. An image is worth 16×16 words: Transformers for image recognition at scale. In: 9th International Conference on Learning Representations (Vienna, Austria, May 2021).

[41] Li, Y., Jia, X., Sang, R., Zhu, Y., Green, B., Wang, L. & Gong, B. Ranking neural checkpoints. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (Nashville, Tennessee, United States of America, June 2021), pp. 2662–2672. DOI:10.1109/CVPR46437.2021.00269.

[42] Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A. & Jegou, H. Training data-efficient image transformers & distillation through attention. In: Proceedings of the 38th International Conference on Machine Learning (July 2021), pp. 10347–10357.

[43] You, K., Liu, Y., Wang, J. & Long, M. LogME: Practical assessment of pre-trained models for transfer learning. In: Proceedings of the 38th International Conference on Machine Learning (July 2021), pp. 12133–12143.

[44] Wen, Y., Chen, L., Deng, Y. & Zhou, C. Rethinking pre-training on medical imaging. *Journal of Visual Communication and Image Representation* **78,** pp. 103145. DOI:10.1016/j.jvcir.2021.103145 (July 2021).

[45] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S. & Guo, B. Swin Transformer: Hierarchical vision transformer using shifted windows. In: 2021 IEEE/CVF International Conference on Computer Vision (ICCV) (Montreal, Quebec, Canada, October 2021), pp. 9992–10002. DOI:10.1109/ICCV48922.2021.00986.

[46] Tahir, A. M., Chowdhury, M. E. H., Khandakar, A., Rahman, T., Qiblawey, Y., Khurshid, U., Kiranyaz, S., Ibtehaz, N., Rahman, M. S., Al-Maadeed, S., Mahmud, S., Ezeddin, M., Hameed, K. & Hamid, T. COVID-19 infection localization and severity grading from chest X-ray images. *Computers in Biology and Medicine* **139,** pp. 105002. DOI:10.1016/j.compbiomed.2021.105002 (December 2021).

[47] Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T. & Xie, S. A ConvNet for the 2020s. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (New Orleans, Louisiana, United States of America, June 2022), pp. 11976–11986. DOI:10.1109/CVPR52688.2022.01167.

[48] Cao, H., Wang, Y., Chen, J., Jiang, D., Zhang, X., Tian, Q. & Wang, M. Swin-Unet: Unet-like pure transformer for medical image segmentation. In: Computer Vision – ECCV 2022 Workshops **3** (Tel Aviv, Israel, October 2023), pp. 205–218. DOI:10.1007/978-3-031-25066-8_9.

[49] neptune.ai: experiment tracker. Available at https://docs.neptune.ai. 2024.

[50] Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., Hirsh, B., Huang, S., Kalambarkar, K., Kirsch, L., Lazos, M., Lezcano, M., Liang, Y., Liang, J., Lu, Y., Luk, C., Maher, B., Pan, Y., Puhrsch, C., Reso, M., Saroufim, M., Siraichi, M. Y., Suk, H., Suo, M., Tillet, P., Wang, E., Wang, X., Wen, W., Zhang, S., Zhao, X., Zhou, K., Zou, R., Mathews, A., Chanan, G., Wu, P. & Chintala, S. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems **2** (La Jolla, California, United States of America, April 2024), pp. 929–947. DOI:10.1145/3620665.3640366.