Master's degree in Computational Engineering
and Intelligent Systems

Department of Computer Science and Artificial Intelligence

Master's Thesis

Transformer-based architecture for 2D semantic segmentation of mitochondria

Aitor González Marfil

Advisors

Ignacio Arganda Carreras
Ikerbasque
University of the Basque Country (UPV/EHU)

Fadi Dornaika
Ikerbasque
University of the Basque Country (UPV/EHU)

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Konputazio Ingeniaritza eta Sistema Adimentsuak Unibertsitate Masterra
Máster Universitario en Ingeniería Computacional y Sistemas Inteligentes

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Master's Thesis

Master's Degree in Computational Engineering and Intelligent Systems

Transformer-based architecture for 2D semantic segmentation of mitochondria

Aitor González Marfil

Advisors
Ignacio Arganda Carreras
Fadi Dornaika

September 2022

Abstract

In this project, we have studied the state-of-the-art of semantic segmentation of biomedical images and compared the performance of a new Transformer-based architecture with the most used convolutional architecture for semantic segmentation of mitochondria in Electron Microscopy (EM) images. This is particularly interesting because both architectures are quite similar, with the main difference being the use of a Transformer as an encoder in one of them. For this comparison, we have adapted an existing Transformer-based architecture used in 3D medical images to perform 2D semantic segmentation, explored multiple variations in both the convolutional and Transformer parts, and finally performed a comparison between the two architectures under the same conditions. Furthermore, we also analyzed the impact of applying different self-supervised learning tasks as a pre-training strategy for the network.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In recent years, the number of digital images available in different domains has increased significantly due to advances in multiple image acquisition techniques. The huge amount of images and the facilities to access them have significantly driven the development of the field of computer vision in the last decade. One of those imaging methods is electron microscopy (EM), which in recent years has enabled scientists to study different organelles, such as nuclei or mitochondria. The study of various organelles with nano-scale precision plays a crucial role in the detection of serious diseases such as cancer [1], Parkinson [2] or Alzheimer [1].

Since labeling images is neither easy nor cheap, and certainly not at the rate at which they are obtained, the need arises to automate the process. In recent years, techniques based on deep neural networks have dominated a wide range of disciplines, including several computer vision tasks such as image classification [3]. However, in this project, we focus on the semantic segmentation task, which aims to classify different elements of an image. An example can be seen in Fig. 1.1. In recent years, this task [4, 5] as well as other computer vision tasks such as super-resolution [6], has been dominated by convolutional neural networks (CNN).

Moreover, in recent years, the Transformer architecture [7] has achieved state-of-the-art results in many Natural Language Processing (NLP) tasks [8, 9, 10] with very competent performance in multiple other problems, even when the network has not been trained to solve those specific tasks [11]. All these results have led to further research in various areas, such as audio [12], image [13, 14], or even more, multimodal applications [15, 16], where it has proven to be a very versatile architecture [17]. However, the main potential of this architecture is scalability, where the bigger the dataset, the better. In that sense, the number of annotated images is still an open problem in biomedical computer vision [18]. Various approaches have been proposed to solve this problem, including data augmentation [19, 20, 21, 22], synthetic data generation [22, 23] and transfer learning [24, 25]. But they still require a large number of task-related annotated images to perform well. Moreover, it has been shown that models trained over one dataset often struggle to generalize over other datasets with different distributions. In this sense, several approaches have also been proposed, such as style transfer or multitask neural networks [26].

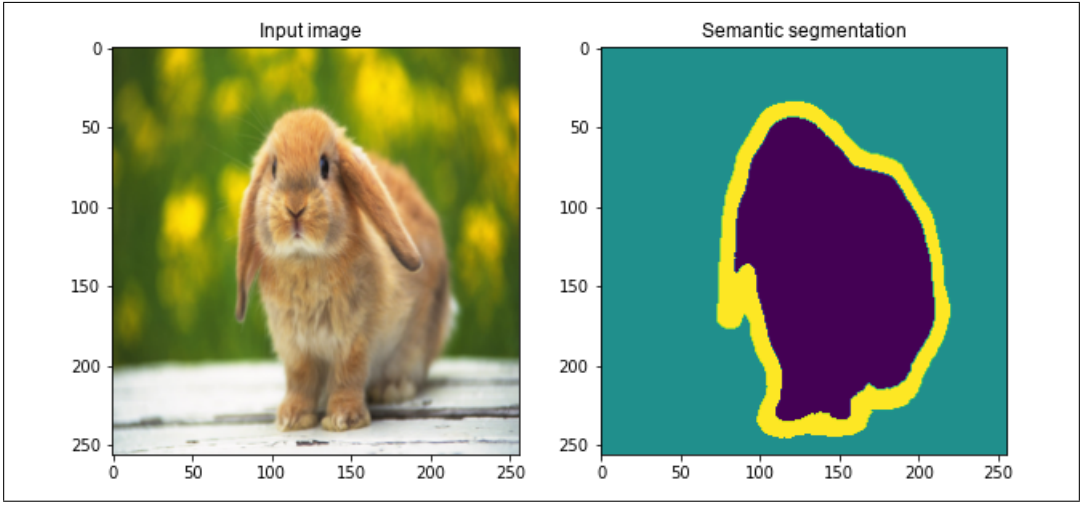


Figure 1.1: Example of semantic segmentation. From left to right: the input image and its semantic segmentation. Each class is represented by a color, being navy blue the animal-class, yellow the border of the animal, and turquoise the rest.

One way to address the data sparsity problem is by self-supervised learning (SSL), which consists of establishing a pre-training task using unlabeled related images that do not require expert annotations to train the model, and then using that model as the starting training point for the downstream (segmentation) task. Although the idea is not new, SSL has recently resurfaced thanks to its great success in NLP [27, 28, 29, 30] and in other computer vision applications [31, 32].

In this particular project, we will analyze a very recent Transformer-based segmentation architecture (UNETR [33]), and compare it with the state-of-the-art in EM image mitochondrial segmentation: a U-Net [34] type of architecture. The main difference between the architectures is that, even if both maintain the same idea, one develops it by using a fully convolutional neural network, while the other one combines the previous architecture with a Transformer encoder.

In brief, our main contributions are as follows:

1. We have studied UNETR variants in both the convolutional and Transformer components of the architecture.
2. We have compared both the state-of-the-art U-Net-based model and our Transformer-based model under the same conditions to see the impact of the Transformer in the biomedical segmentation task across a variety of configurations.
3. We have designed a hybrid model and compared it with the previous approaches.
4. We have analyzed the effect of applying different SSL techniques.
5. Derived from this work, we have designed, implemented and published [26] a new convolutional model for semantic segmentation using domain adaptation (see Appendix A).

CHAPTER 2

Related Work

In this chapter, we will explain the two main architectures we are going to work with. For a better understanding, some context of related architectures will be also given.

2.1 U-Net and CNN architectures

Following the flow of historical events, the first architecture to be presented is the U-Net [4]. This architecture starts with a very well known encoder, used previously by other famous networks as AlexNet [35]. The encoder consists of multiple 2D convolutions combined with pooling layers, where the deeper, the smaller the spatial resolution but bigger the channel dimension. More precisely, each pooling layer divides by 2 the spatial resolution. Unlike AlexNet, those fully-connected layers that make up the classification head are not used. Following a similar idea, but with 2D transposed convolutions instead of pooling layers, the decoder is formed. This time, the deeper, the bigger the spatial resolution but smaller the channel dimension. Inversely proportional to the encoder, the spatial resolution is multiplied by 2 for each transposed convolution. Putting both parts together, we obtain as a result a fully convolutional autoencoder, where the input and the output keeps the same exact dimensions. What U-Net does is to incorporate multiple skip-connections throughout the network. The skip-connections are built layer-wise, passing the signal from the encoder to the decoder directly, so they always work with the same spatial dimensions at each symmetric level of the architecture. These skip-connections let the network restore multiple details that could be lost when reducing the image size in the pooling layers. When we represent graphically the network (see Fig. 2.1), the U shape appears, this is why the network is called U-Net.

Even if the U-Net was first presented for biomedical image segmentation, it has been used for a wide range of tasks. As long as the output keeps the same image size as the input, this network could be used. For example, in the denoising [36] or inpainting [37] tasks. Moreover, with few modifications as incorporating an extra decoding layer, other tasks as super-resolution can be performed [38].

Since the U-Net was introduced, multiple variants have appeared [39, 40, 34], achieving very competent results. As an example, one of those U-Net variants replaced the simple

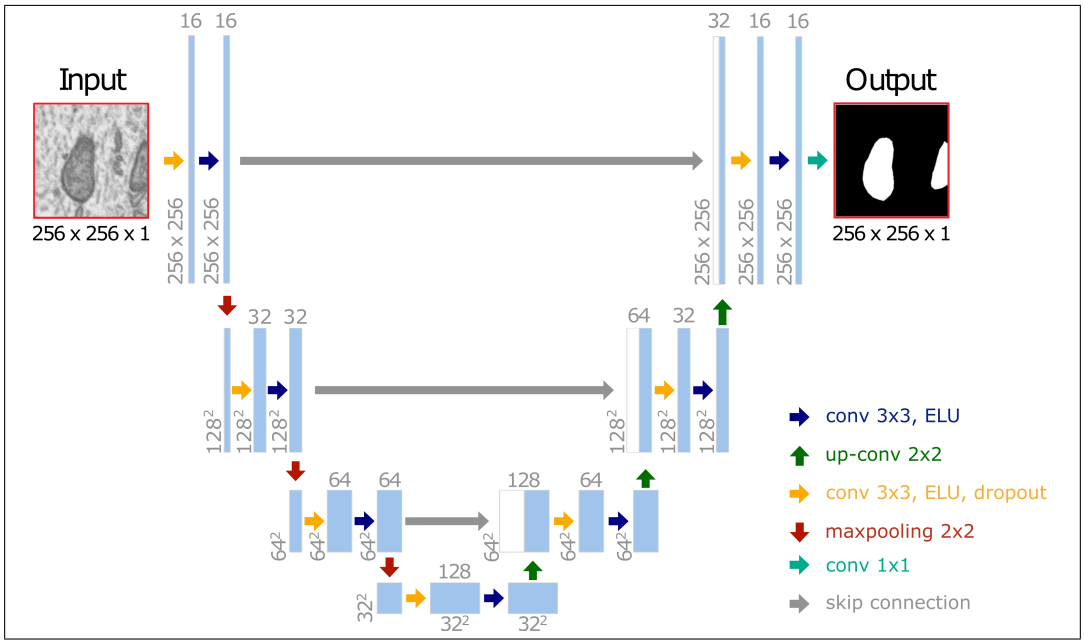


Figure 2.1: U-Net architecture [34] example with 3 layers. Each one of the blue boxes corresponds to a multichannel feature map. The number of channels is indicated on top of each box. The spatial size is provided at the lower left edge of each box. The white boxes represent copied feature maps. The arrows represent different operations.

concatenation applied over every skip connection and introduced an attention gate [41]. By using attention gates, the features are scaled with the calculated attention coefficients in the attention gate. The resultant Attention U-Net and the functioning of attention gates are shown in Fig. 2.2.

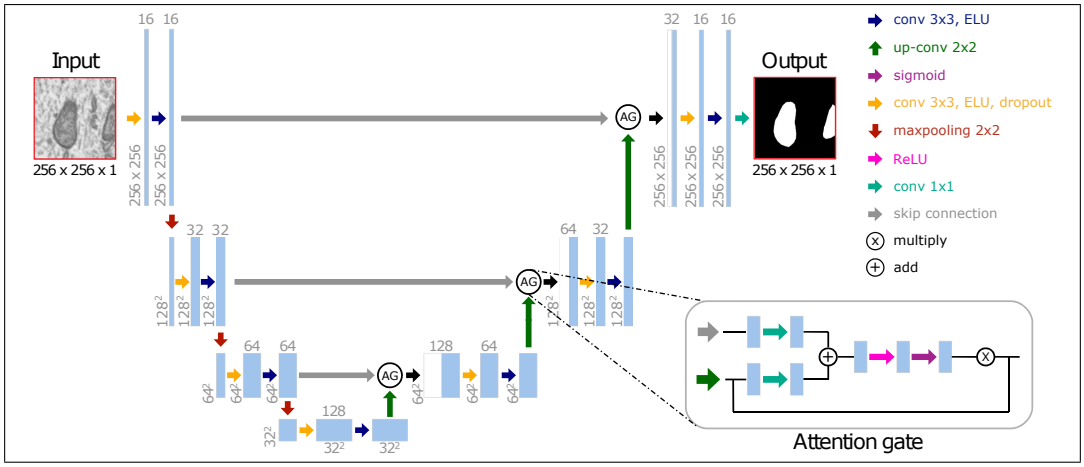


Figure 2.2: 2D Attention U-Net architecture [34] example with 3 layers. Each one of the blue boxes corresponds to a multichannel feature map. The number of channels is indicated on top of each box. The spatial size is provided at the lower left edge of each box. The white boxes represent copied feature maps. The arrows represent different operations. In the lower right side of the figure, a detailed description of the attention gates is given. Image obtained from [34]

Also with a shape related naming, the Y-Net is another of the U-Net variants. In this case, the Y shape comes from placing a second encoder [42] or decoder [43, 26] in the network.

While the second decoder is placed following an autoencoder approach, the second encoder could be a pretrained encoder, with useful features already learned. However, despite so many variants, the U-Net remains a fully competent alternative, with state-of-the-art results in some tasks such as the one we will be working on in this project: segmentation of mitochondria in EM images [34].

2.2 Transformer architecture

In the years prior to the appearance of the Transformer [7], multiple attention strategies had been already proposed in the area of NLP [44, 45]. The problem at the time was the high computational cost of networks, due to such attentions in recurrent neural networks (RNNs) involve $O(n)$ sequential operations. Nevertheless, the Transformer architecture proposes the usage of the self-attention (with $O(1)$ sequential operations) to compute representations of the input and output, without any kind of recursivity or convolutions. Basically, their proposal is based on the idea that "Attention is all you need", as the title of the paper explains. Here we briefly present the architecture proposed in that paper.

The Transformer architecture follows the encoder-decoder scheme (see Fig. 2.3). Originally, the architecture was proposed for transduction, where the encoder works with the input signal and the decoder works with the expected signal. As an example, for translation, the encoder will receive as input the language L_i while the decoder will work with the language into which it is to be translated L_o. With the passage of time, different variants have been proposed for multiple other NLP tasks, some based only on multiple encoder-layers as BERT [9], and others based only on multiple decoder-layers as GPT [46].

Before starting with the encoder and decoder, a positional embedding is added to the input signal, formed by a sequence of words in the case of NLP tasks. This is a very important point due to the fact that the Transformer architecture is fully feed forward, so it has no information about the order, and thus it is necessary to do it explicitly. Once we have the signal ready, we can start explaining the rest of the architecture.

2.2.1 Encoder-layer

The encoder-layer contain two residual modules: the first one contain the multi-head attention (MHA) block followed by a skip connection and a normalization layer, the second one contains a feed-forward block followed by another skip connection and a normalization layer.

2.2.1.1 Multi-head attention block

The MHA block requires of query, key, and value signals, which are packed together into Q, K, and V matrices, respectively. In the case of the encoder-layer, these signals come from exactly the same input signal, which is trifurcated. Each one of the signals is passed through a linear transformation before self-attention is computed. Using these signals, the output of a single head can be expressed as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V, \tag{2.1}$$

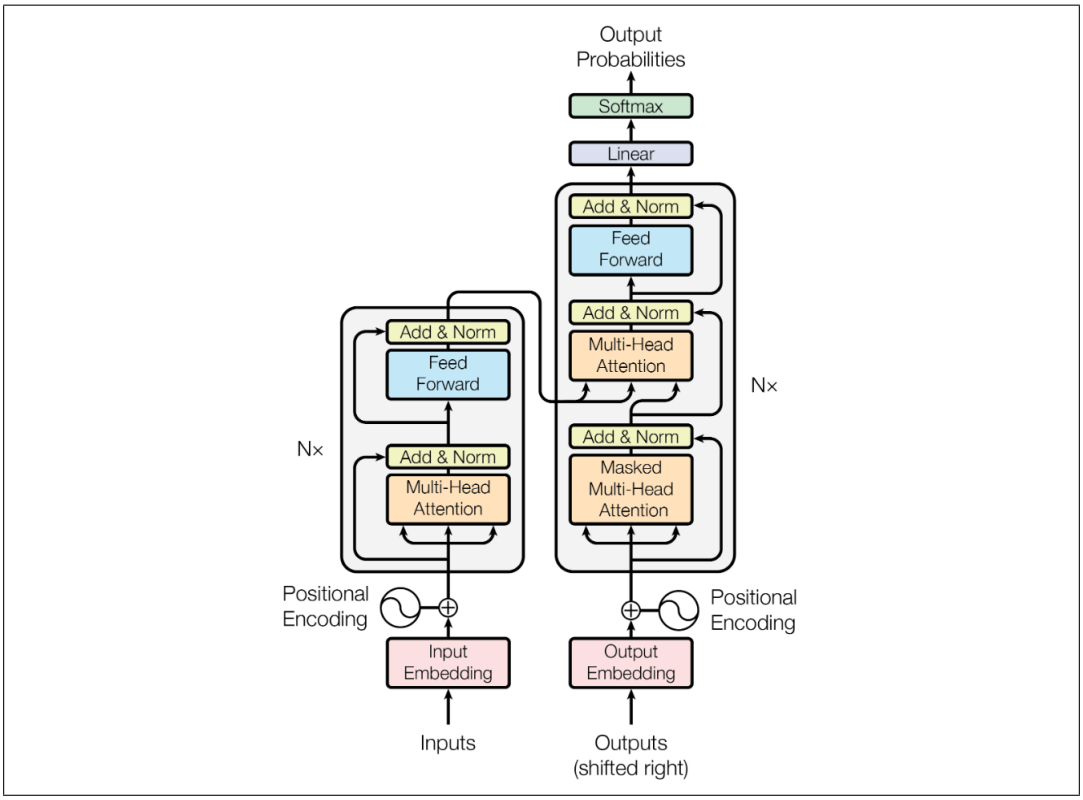


Figure 2.3: Transformer architecture [7]. The left gray box shows how a single encoder layer is constructed, being the full encoder a stack of N encoder layers. This encoder layer is built by two residual modules, the first one with a multi-head attention block, and the second one with a feed-forward block. Similarly, the right gray box shows how a single decoder layer is constructed, being the full decoder a stack of N decoder layers. This decoder layer is built by three residual modules: the first one with a masked multi-head attention block, the second one with a multi-head attention block, and the last one with a feed-forward block. Each of the residual connections are followed by a layer normalization. In both parts of the network, a positional encoding is added to the input signals. The output of the encoder is given as *Query* and *Key* signals to the multi-head attention block of the second residual module of the decoder. Finally, at the top of the network, a classification head has been added. At each time step, a single prediction is given as an output, to build a full sequence, multiple iterations are needed, where at each time step, previously predicted values are given as an input to the decoder. Image obtained from [7]

where d_k refers to the dimensionality of the key and query signals. The attention scores are first computed by multiplying Q and K^T. Then, in order to avoid large values which could create extremely small gradients after applying the softmax function, the scores are scaled by $\frac{1}{\sqrt{d_k}}$. The outcome of applying a softmax function is an attention matrix with values between 0 and 1. After calculating the attention for each word, this is applied to the value signal by multiplying it. This process is repeated for each of the heads (h). A visual representation of the block is shown in Fig. 2.4.

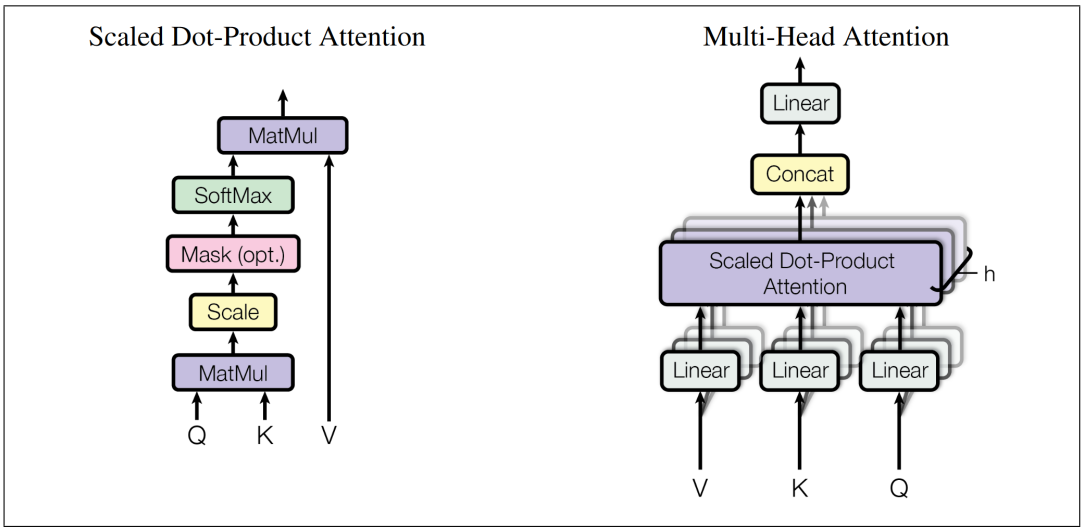


Figure 2.4: Transformer attention. Left: Scaled Dot-Product Attention [7]. The block starts by multiplying Q and K signals. The output is then scaled by $\frac{1}{\sqrt{d_k}}$. In the case of the masked multi-head attention placed on the decoder, now the masking would be performed by replacing the values to be masked with $-\infty$. In the case of normal multi-head attention block, this masking step is directly skipped. By applying softmax to the resulting signal, an attention matrix is obtained. Finally, this matrix is multiplied to the V signal. Right: Multi-Head Attention block [7], with h heads. For each of the signals, first a linear transformation is applied, and then is passed through the scaled dot-product attention block which computes the self-attention. This is repeated for each of the h heads. Finally, the outputs of each of the heads are concatenated and a final linear transformation is applied. Images obtained from [7].

Each of the computed heads could focus on different aspects of the input signal, such as paying attention to the adjacent words or tracking certain syntactic relations [47]. Even though, the heads are learned during training, so we do not know what each head will learn. Post-training analysis is necessary for this, and even then, the interpretation plays a very important role, where usually it is not easy to understand what exactly they are attending at. Although we can not explain what each head does, we can assess their importance and prune those heads that are less important. Doing it, we can achieve lighter model without compromising seriously the model's performance [47]. The MHA design is specially interesting by the parallelization capability, as each of the heads can be processed completely independently of the others. Finishing with the MHA block, after each head is computed, all of them are concatenated and passed through a linear transformation.

2.2.1.2 Feed-forward block

Unlike the MHA block, the feed-forward block is a simple block composed by two linear transformations with a ReLU activation function between them. The block can also be

defined as follows:

$$FF(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.2}$$

where W_1 and W_2 are the weights of the first and second linear transformations, respectively. And, b_1 and b_2 are their corresponding biases.

2.2.2 Decoder-layer

The decoder-layer is composed by three residual modules with their respective normalization layers, as the encoder does. The first module is formed by a masked multi-head attention block, followed by a MHA block, and a feed forward block.

Taking into account that the network only predicts one word at a time, it is necessary to incorporate the previous predictions into the network. This is exactly what the decoder receives as an input in the masked multi-head attention block, with their respective positional embeddings. But since the input size is fixed, and we only have to attend to the values we already have at any given time, we have to mask out those future values that the network has yet to predict. Precisely for that purpose, a masking layer is placed in the MHA block, right after scaling the values. The values belonging to future predictions are replaced by $-\infty$, which become zeros in the attention matrix after softmax is performed.

The outcome of the first module is delivered as V to the MHA block of the next module, and as a residual connection as well, while Q and K come from the output of the encoder. The result is finally given to the last feed-forward module, as we have seen in the encoder.

2.2.3 Stacked layers

So far, we have explained what each of the encoder and decoder layers look like. The complete encoder is formed by N stacked encoder-layers, and similarly, the complete decoder is formed by N stacked decoder-layers. Finally, to perform word classification, on the top of the network, at the end of the decoder, a classification head is placed, formed by a linear transformation followed by a softmax function, to obtain the output probabilities.

2.2.4 Problems

Over time, Transformer-based models have shown amazing scalability, where the larger the network size and the dataset, the better the performance [8, 13]. At the moment, one of the biggest models with state-of-the-art results in many NLP-tasks is GPT-3 with $175 \cdot 10^9$ parameters [8]. But it is not all good, the resources needed to train these models are not accessible to everyone. As an example, the previous version GPT-2 with 1542M parameters, requires one week of training on 32 TPUv3 chips, with an estimated cloud computing cost of between $12,902 and $43,008 [48], which also leads to high CO_2 consumption.

One of the problems of this architecture is the per-layer complexity, where self-attention layer complexity is $O(n^2 \cdot d)$. Where d is the representation dimension also known as hidden dimension. But the biggest problem is n, which refers to the length of the input sequence. To solve this, multiple alternatives has appeared over time, with different attention strategies. But recently it was shown that even if they require less computation, they also achieve

lower accuracy values than the regular self-attention in most of the cases [49]. Even so, they are still an alternative to be considered.

2.2.5 Vision Transformer (ViT) and UNETR

The success of Transformers in many NLP tasks, motivated further research in other areas like computer vision. However, the quadratic cost with respect to the length of the input sequence is a serious problem which limited severely the research in this area in its early stages [50]. A few years later, researchers discovered that one possible approach was to divide the image into patches, flatten each patch into a vector, apply a linear transformation, and work with them as if each vector were an embedded word [13]. Hence, the title of the original paper: "An image is worth 16x16 words: Transformers for image recognition at scale". The architecture they used, known as Vision Transformer (ViT) is a stack of multiple Transformer encoder-layers (see Fig. 2.5). Once more, exploring the idea of increasing the Transformer size, they propose three different models with different configurations, presented in Table 2.1.

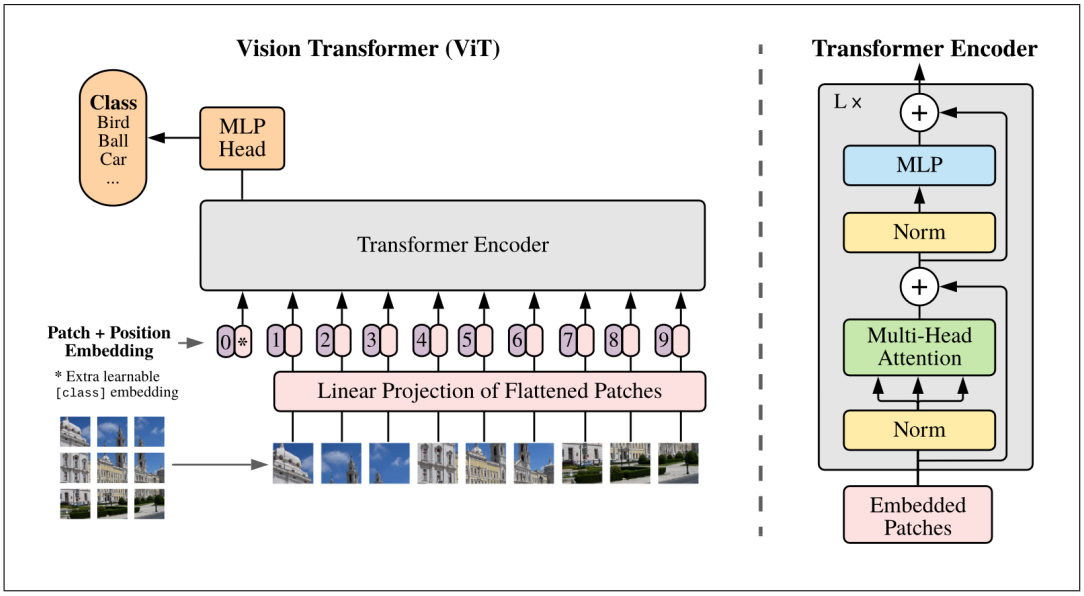


Figure 2.5: ViT architecture [13]. The input image is first split in fixed-size patches, which are linearly embedded, then position embeddings are added and fed to a Transformer encoder. In order to perform classification, an extra learnable "classification token" (also known as [CLS] or [CLASS]) is added to the sequence. Image obtained from [13].

Model	Layers	Hidden size	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 2.1: Details of Vision Transformer model variants [13].

Later on, by analyzing the ViT architecture, researchers discovered that spatial information is maintained throughout the network [51]. This is specially interesting for tasks such as object detection or semantic segmentation, which is why further research on the

subject has been initiated. One of those proposals is the UNETR [33] we are going to work with in this project. The name comes from "UNEt TRansformers".

Originally, UNETR was presented for 3D biomedical image segmentation. The architecture takes a U-Net as the base, where instead of 2D operations, 3D ones are performed. From this 3D U-Net, they replace the original convolutional encoder and place instead 12 Transformer encoder-layers, in a ViT style. But in this case, the Transformer input is a sequence of non-overlapping 3D patches instead of 2D ones. At this point, the encoder is like the ViT with the only change of the extra input token [CLS], designed for the classification task. Since this network is designed for semantic segmentation, this token is not added.

The skip connection signal of each layer l comes from the output of the $3 \cdot l$ -th encoder layer's output. Where the first layer $l = 0$, creates a direct skip connection from the input, with two convolutional blocks. Since the output of each encoder layer has the same dimensions as the input, the first step is to reshape the signal and restore the three dimensions. To adjust the resolution of the signal in each layer, several blocks based on convolution and transpose-convolution are integrated in each skip connection. The network is visually represented in Fig. 2.6.

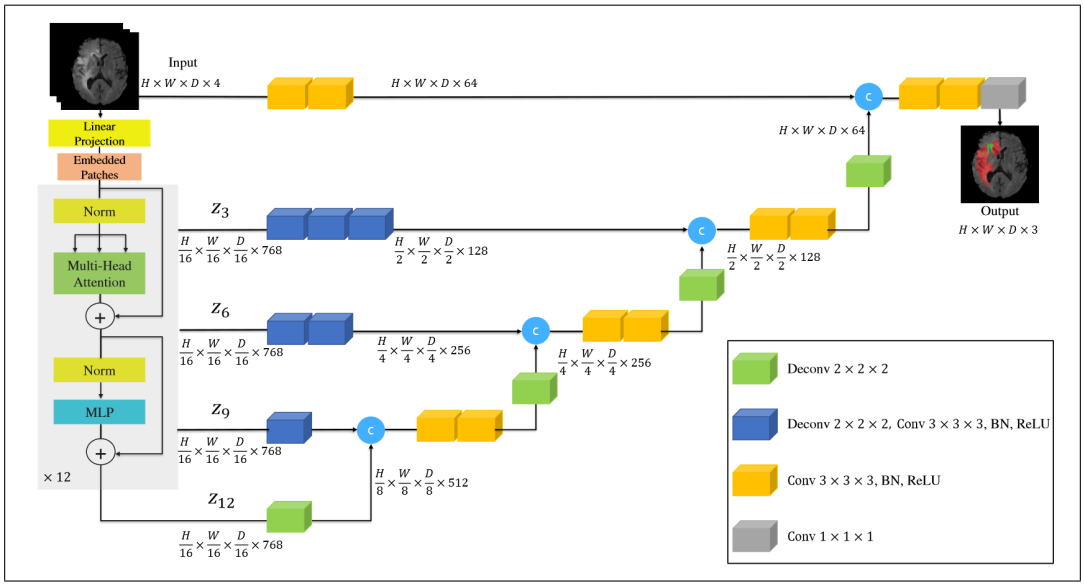


Figure 2.6: UNETR architecture [33]. The 3D input volume contains four channels. The input volume is divided in patches of $16 \times 16 \times 16$ to introduce them later in the Transformer. The hidden size of the Transformer is 768. The output contains three different classes in the original segmentation task. Image obtained from [33].

CHAPTER 3

Methodology

In this chapter, the changes applied to the architectures are presented. In addition, the way we perform SSL is also explained.

3.1 UNETR-2D

In order to perform a fair comparison between both U-Net and UNETR architectures, the first step is to adapt the UNETR architecture to perform 2D semantic segmentation. To do this, first of all we feed the encoder with a sequence of 2D patches, as ViT does. Moreover, we have adapted the rest of the network to perform 2D operations instead of 3D ones. We will refer to this architecture as UNETR-2D.

This way, we are contrasting the same architecture, with just two main differences: (1) while the U-Net uses a convolutional encoder, the UNTER-2D uses a Transformer-based encoder; and (2) the usage of convolutional blocks and transposed convolutions by the UNETR-2D on those skip connections coming from the ViT. Notice that the first skip connection ($l = 0$) that comes from the input image directly by using two convolutional blocks becomes similar to the first skip connection of the U-Net, which also comes from two convolutional blocks. The difference is that U-Net keeps using the signal after applying a pooling layer, and UNETR instead only uses this signal for the skip connection.

Regarding the hyperparameters, we adapted the hidden dimensionality to our particular case. Originally, the base model proposed for ViT [13] had 768 hidden dimensions, which keeps the same number of dimensions for each patch (note that every $16 \times 16 \times 3$ patch is flattened). In our case, with just one channel, we set the hidden dimensions of the base model to 256. For the rest of the Transformer configurations, we scaled all dimensions proportionally as they do.

3.2 YNETR-2D

Notice the fact that ViT starts to attend more those patches that are close between them when the model is trained over huge datasets [51]. On the contrary, CNNs have this local attention implicitly, by the way it works and its perceptive fields. Similarly, given its

architecture, the ViT can easily pay attention to every patch (the whole image), while CNNs usually cannot [51].

Moreover, we have tried a hybrid network. This network, is built with both convolutional and Transformer-based encoders, and only a single decoder, following the Y-Net approach. We can see this network as a 2D U-Net where a second encoder has been placed, but this time a Transformer encoder. Making use of the same reshape and convolutional blocks used for skip connection signals in UNETR-2D, the skip connection signals of both encoders, are concatenated and passed directly to the decoder, respecting the spatial dimensionality of each layer. After these concatenations, 2D spatial dropout layers are used to force the network to use both encoders. Experimentally, better results have been obtained with such dropout. The idea behind the network is to take advantage of the qualities of each architecture and combine them. From now on, we will refer to this network as YNETR-2D. The YNETR-2D is visually represented in Fig. 3.1.

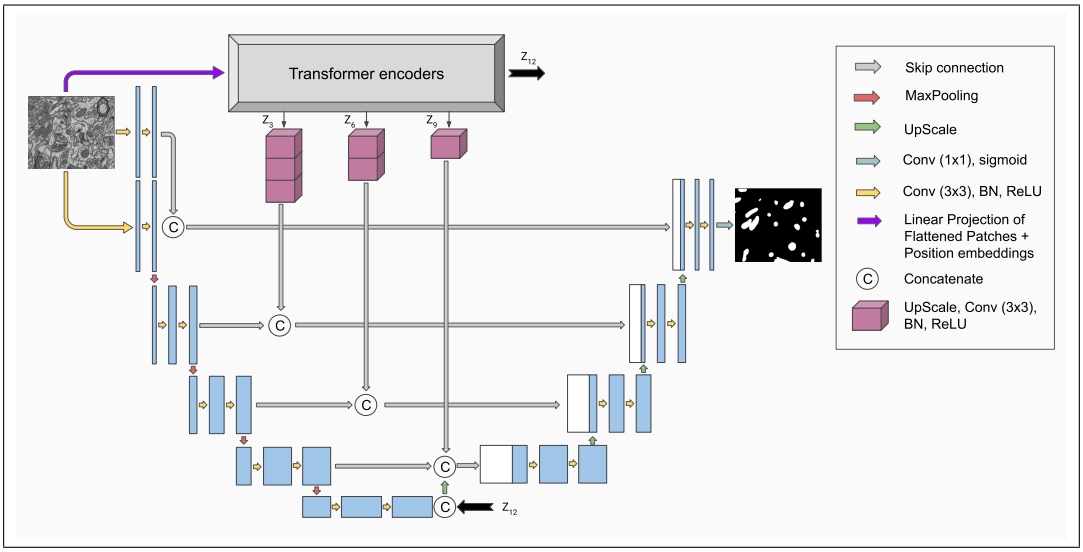


Figure 3.1: YNETR-2D architecture overview. Each one of the blue boxes corresponds to a multi-channel feature map. The white boxes represent copied feature maps. The arrows represent different operations. This architecture corresponds to the YNETR-2D Base version, with 12 Transformer layers, which could be changed. Any of the Transformer output signals is first reshaped, like in UNETR-2D, to recover the shape $\frac{H}{16} \times \frac{W}{16} \times 256$, in the case of using a patch size of 16×16 and 256 hidden dimensions. We reduce the spatial resolution by two using max-pooling, while we use up-scaling or transposed convolutions to increase the spatial resolution also by 2. BN refers to Batch Normalization.

3.3 Self-supervised learning

This section is specially interesting due to the fact that we can improve the model's performance by using data without any label. The way this is performed is by creating an altered version of each image, and letting the network learn how to recover the image, using as ground truth the unaltered image. Doing it, we let the network learn features from the given data that will be useful for future tasks. Once the model is pretrained, we use that model as starting point for the training in the desired downstream task, in our case, semantic segmentation.

There are many ways to alter images, but this project will explore the followings:

- **Cutout [52]:** Using a constant value 0, fill per image between one and five areas, each one having 20% of the corresponding size of the height and width.
- **Additive Gaussian Noise:** Add Gaussian noise to each image, sampled once per pixel from a normal distribution $\mathcal{N}(0, 0.15 \cdot 255)$.
- **Coarse salt and pepper:** Mark 20% of all pixels in a mask to be replaced by salt and pepper noise. The mask has a size between 4×4 and 16×16 pixels. The mask is then upscaled to the input image size, which leads to large rectangular areas being marked as to be replaced. These areas are then replaced in the input image by salt and pepper noise.
- **Low resolution emulation:** To emulate low resolution images, we used the same strategy used in [26], which in short consists of using Gaussian noise with $\mu = 0$ and $\sigma = 0.1$. Next, the images are downsampled and upsampled again by the same factor, in both cases using bilinear interpolation. Instead of using a factor of 2, we have used a factor of 4, which experimentally has given us better results.
- **Gaussian filter:** Apply a Gaussian filter to each image. The standard deviation used in the Gaussian kernel is 3. After the application, we kept the range of $[0, 255]$ values.
- **Defocus blur:** Emulate the aberration formed in the image when this is out of focus. This effect can be emulated by the usage of certain kernels (in our case, we use the kernel obtained with $severity = 2$).
- **Motion blur:** Emulate the apparent streaking of moving objects in a photograph. This effect can be emulated by the usage of certain kernels (in our case, we use a kernel of size 15×15 with an angle randomly picked per image).
- **Pixel dropout:** For each image, sets a random percentage of pixels between 0% and 20% to zero.

For convenience, we work with the *imgaug* [53] library for the modifications, except for the low resolution emulation. A visual example of each of the previously mentioned alterations can be seen in Fig. 3.2.

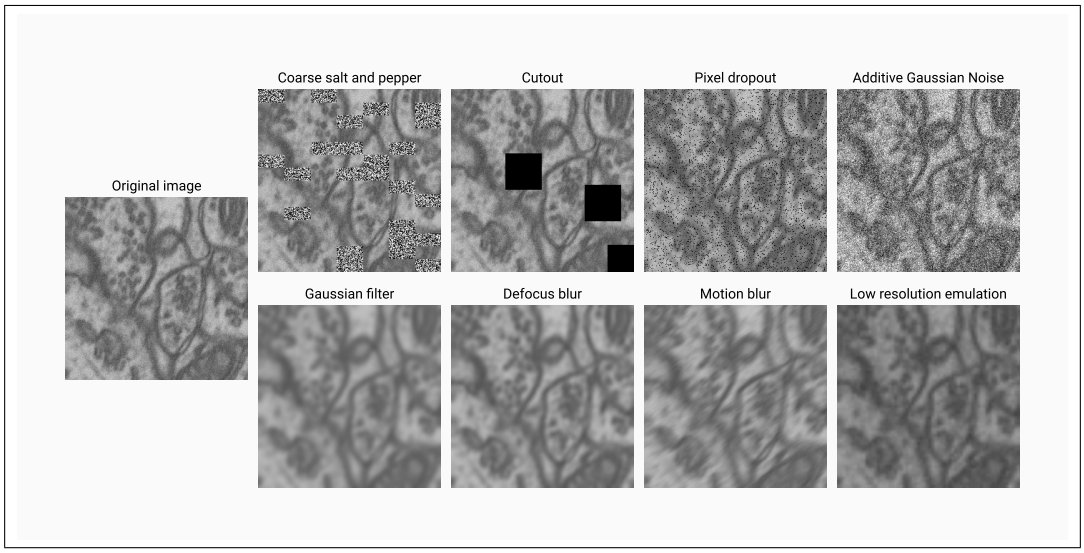


Figure 3.2: Examples of each of the image alteration methods used for SSL.

CHAPTER 4

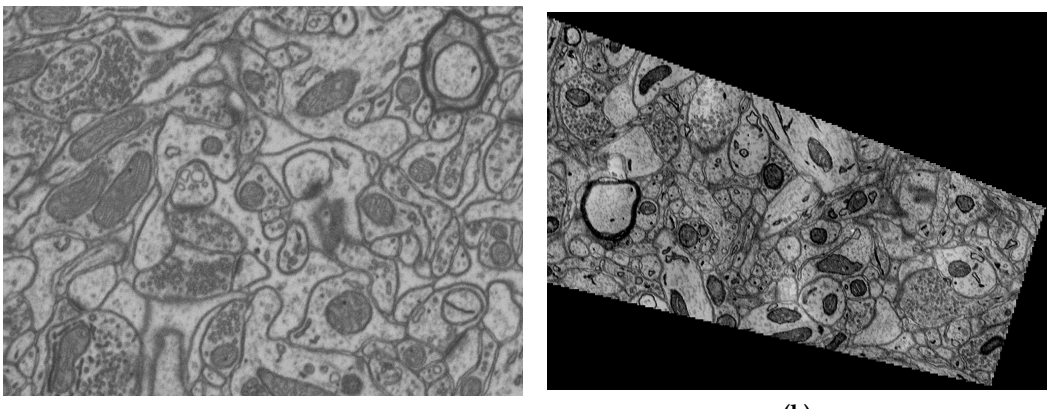
Experiments

4.1 Datasets

Every single experiment performed in this project is based on any of the following publicly available datasets.

4.1.1 Lucchi

The EPFL Hippocampus or Lucchi dataset [54] represents a $5 \times 5 \times 5 \mu m$ section taken from the CA1 hippocampus region of a mouse brain, with an isotropic resolution of $5 \times 5 \times 5 nm$ per voxel. The volume contains in total $2048 \times 1536 \times 1065$ voxels, acquired by the usage of focused ion beam scanning electron microscopy (FIB-SEM). An example image can be seen in Fig. 4.1.



(a) (b)

Figure 4.1: (a) Lucchi image example. (b) Kasthuri image example.

Regarding the labels, the mitochondria of two neighboring subvolumes formed by 165 slices of 1024×768 pixels were manually labeled by experts. This two subvolumes are usually used as training and test sets. In this project, we will use mainly the labels of Lucchi++ [55], which has been polished recently by two neuroscientists and a senior

biologist. Taking into account that we are going to work with 2D images, we will work with 165 slices of 1024×768 for both train and test sets.

4.1.2 Kasthuri++

In a similar way to Lucchi++, Kasthuri++ [55] is the re-labeling of the dataset by Kasthuri et al. [56]. In this case, the volume is a part of the somatosensory cortex of an adult mouse and was acquired using serial section electron microscopy (ssEM). The train set volume dimensions are $85 \times 1463 \times 1613$ voxels, while the test set ones are $75 \times 1334 \times 1553$ voxels, with an anisotropic resolution of $3 \times 3 \times 30nm$ per voxel in both cases. An example image can be seen in Fig. 4.1.

To work with 2D images, we will use 85 slices of 1463×1613 for the training set and 75 slices of 1334×1553 for the test set.

4.2 Evaluation

Recalling that Transformers use a fixed image size, we need to apply any methodology to measure the segmentation quality under the same exact images as the others.

The first evaluation tested was cropping each image in 256×256 sequential patches with no overlapping, pass them through the network and then reconstruct the full size segmentation. But doing it, several imperfections appears in the reconstructed segmentation. Those regions with most errors were clearly on the joints between patches. So we discarded this method.

The second evaluation tested is the one we have chosen and used. The idea is to reconstruct the full-size segmentation but just using the central 128×128 part (for the sake of simplicity, let's call it a relevant patch) of each 256×256 patch. In this way, for each relevant patch, the outline is available to the network as a context, reducing the errors we found in the joints with the previous method. Let's divide it in two cases:

- If the height and width of the given image are divisible by 128, the method starts by adding 64 pixels of padding to each side of the image. Next we divide the image in 256×256 patches with 128 pixels of overlapping between them. Finally, after processing with the network, relevant patches are extracted and by concatenating it full size segmentation is achieved. This method is visually represented in Fig. 4.2.

- If the height and width of the given image are not divisible by 128, instead of adding 64 pixels of padding, the minimum number of padding pixels are added until the height and width are divisible. As far as possible, the padding is equal on every side of the image (left-right and top-bottom). Next, we follow the same steps as in the previous case, with only one change in the last part. This time, border relevant patches are no longer 128×128, for each relevant patch, padding side pixels are incremented until reaching the original border of the image, i.e., no padding pixel is used for any relevant-patch. This method is visually represented in Fig. 4.3.

Initially we used zero-padding, but we realized that padding damaged seriously the prediction. Due to the self-attention, the error was propagated through the whole 256×256

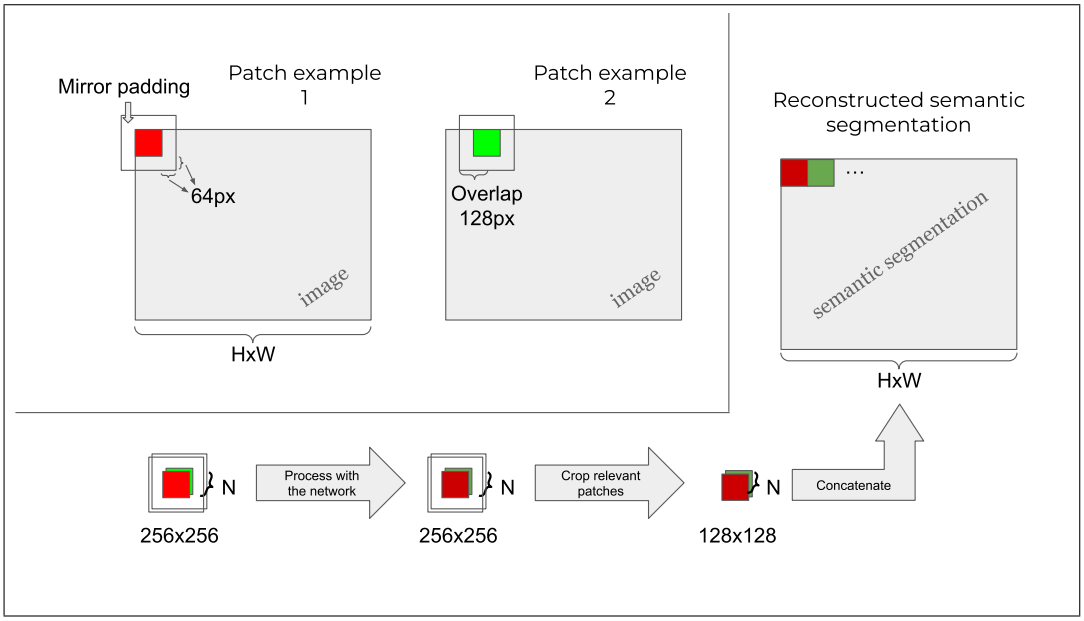


Figure 4.2: Illustrated evaluation method, first case: The image height (H) and width (W) are divisible by 128. With 64px of mirror padding, the image is first divided into N patches of 256×256, with 128px of overlapping between them. After processing them with the network, the colored 128×128 relevant patches are cropped from each patch. Finally, by concatenating them while keeping the place of origin, the complete semantic segmentation is reconstructed.

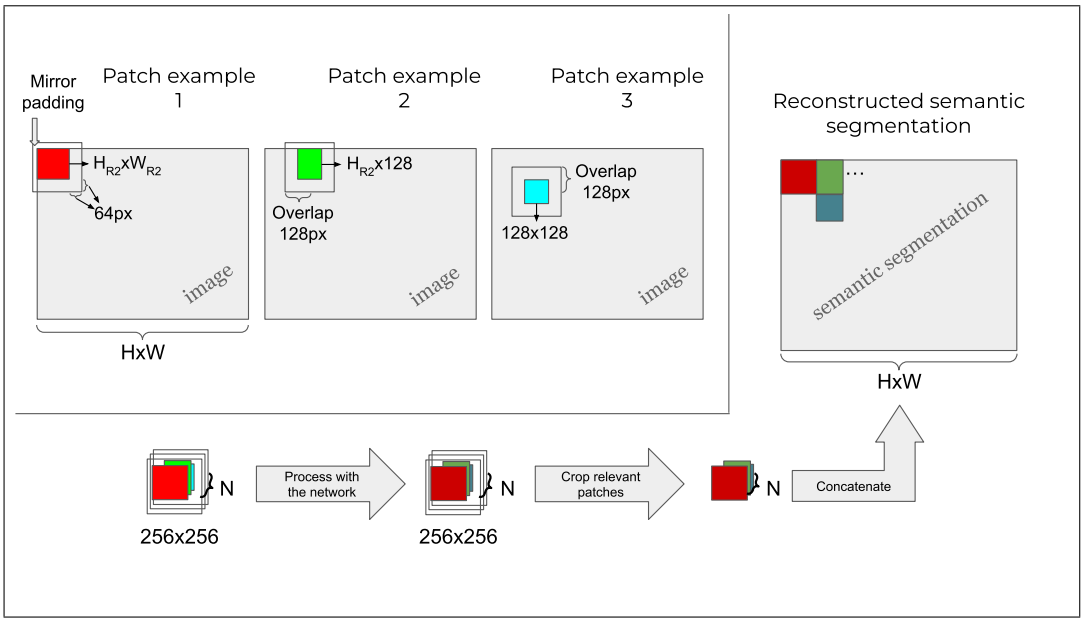


Figure 4.3: Illustrated evaluation method, second case: The image height (H) and width (W) are not divisible by 128. With as little mirror padding as possible to become the image H and W divisible by 128, the image is first divided into N patches of 256×256, with 128px of overlapping between them. After processing them with the network, the colored relevant patches are cropped from each patch. Those relevant patches, which belong to the edge of the image, will have the height (H_{R2}) and width (W_{R2}) necessary to cover the whole image. Finally, by concatenating them while keeping the place of origin, the complete semantic segmentation is reconstructed.

patch. It is important to note that these experiments have been carried out with a model trained with Lucchi++, which does not contain large black elements. Finally, we solved that by applying mirror-style padding. An example of each method's result is shown in Fig. 4.4. Note that, although in the above explanations we have shown the values we have used in our particular case, this method could perfectly well be used with other parameters.

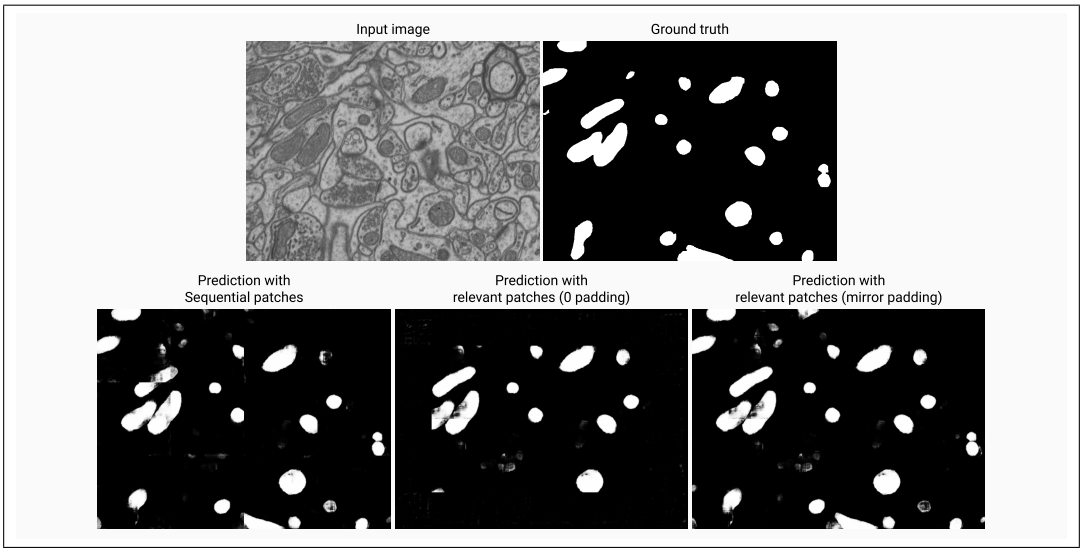


Figure 4.4: Example of results using each reconstruction method. The predicted images show the reconstruction of the multiple probability maps, given directly by the trained UNETR-2D network, with the probability of each pixel belonging to a mitochondrion, where the whiter the pixel, the higher the probability.

In the case of convolutional neural networks, such as U-Net or Attention U-Net [4, 41, 34], the input size is not limited, and we can make use of a different image size than the one used during training. Therefore, the evaluation has been carried out by giving as input the full-size images. Consequently, a full-size segmentation is obtained. Unlike Transformer-based models, no special processing or assembly technique is required. The best results have been obtained by this way of working [34].

4.2.1 Metrics

The semantic segmentation task consists on performing pixel-wise classification for a given image. As an example, if the classes we are working with are 0 for animals and 1 for the rest of elements, the expected segmentation for a given image, is a mask of the same spatial resolution, filled with 0 and 1 values, depending on if the given image contains any animal or not. Where those pixels that are part of any animal in the image should contain 1 value, and the rest of pixels should contain 0 value.

To measure the quality of the predicted segmentation, several metrics can be used, although not all of them fit well for this problem with highly unbalanced classes. Some of the most famous metrics used for this task are the Dice coefficient, also known as F1-score and the intersection over union (IoU), also known as Jaccard index.

Dice measure two times the intersection between the predicted mask and the ground

truth, and divide it by the sum of both sets. This metric could be also defined as follows:

$$Dice = \frac{2TP}{2TP + FP + FN} \tag{4.1}$$

where TP are the true positives, FP the false positives and FN the false negatives, being the foreground the positive class and the background the negative one. This dice coefficient give us a similarity coefficient in the range [0,1] where 1 means a perfect coincidence or segmentation, and 0 no coincidence.

On the other hand, IoU measures the overlapping between the predicted mask and the ground truth, and divide it by the union of both. And the given value is also between [0,1], where 1 means a perfect coincidence or segmentation, and 0 no coincidence. This metric is illustrated in Fig. 4.5. Usually this metric is performed for each class individually, and the final value is just the mean over all classes. In our particular case, the background is much more likely than the foreground class, where the foreground is the mitochondria and the background is the rest of the elements. Therefore, we will work with the IoU_F value, which measure the IoU of the foreground class exclusively. Doing it, we avoid any IoU inflation given by the background class. IoU_F can be also defined as follows:

$$IoU_F = \frac{TP}{TP + FP + FN} \tag{4.2}$$

To compare ourselves with others, we will make use of the IoU_F metric. To predict foreground pixels from the probability map provided by the neural network, a threshold value of 0.5 is applied.

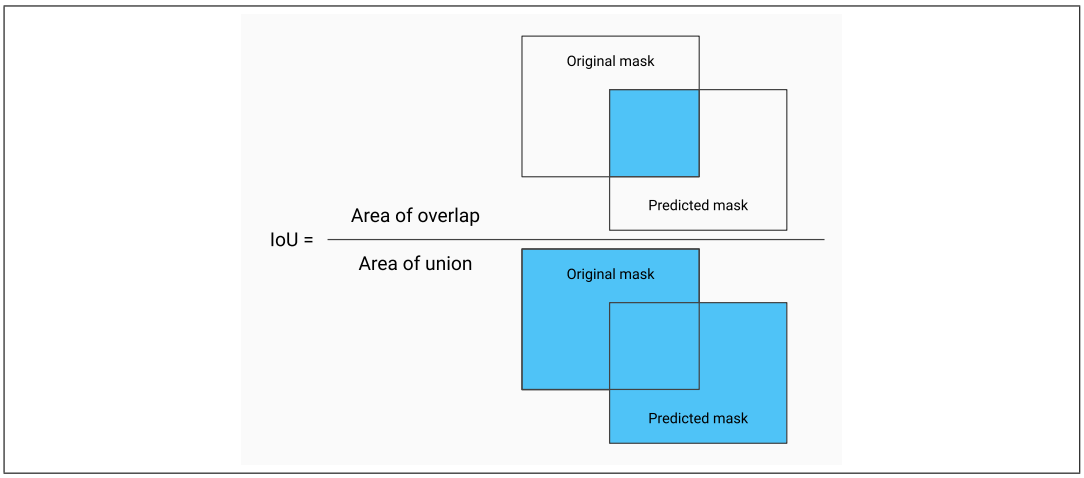


Figure 4.5: Illustration of the intersection over union metric for a single class.

4.3 Implementation details

4.3.1 Data processing

Since UNETR-2D works with Transformers, a fixed input size is needed. To reproduce the training conditions used in the U-Net [34], which uses patches of 256×256, we have set the input size of UNETR-2D to $256 \times 256 \times 1$ ($H \times W \times C$), which refers to height, width, and channel respectively. To obtain the patches of this resolution, extra needed padding

is added to each image until getting a H and W multiple of 256, with as little padding as possible. The padding is applied in a mirror way. If the image H and W was already multiple of 256, no padding is added. Next, 256×256 sequential patches are extracted, with no overlapping. The data is normalized in the $[0, 1]$ range.

4.3.2 Training setup

We explored a wide range of hyperparameters, with different schedulers, batch sizes, etc. Table 2 in Appendix B lists all the hyperparameters explored. To be more precise, we minimize the binary cross-entropy (BCE) loss using the AdamW optimizer [57] with $1e-5$ of weight decay, a cosine decay learning rate schedule, a learning rate of $1e-4$, and batch size of 6. We train for 360 epochs, with a patience of 60 epochs, monitoring the evolution of the validation loss. The validation set is formed with the 10% of the training data, where images are selected at random. During the training process, we also apply the on-the-fly data augmentation (DA) proposed [34] for this task, namely random rotations, vertical flips and horizontal flips. The code has been implemented using *Python v3.8.10* [58] as the programming language, and *Tensorflow v2.9.1* [59] and *Keras v2.9.0* [60] as the main libraries. To speed up the training process, we used a GPU card, in particular the *NVIDIA GeForce RTX 3090*.

CHAPTER 5

Results

Every result presented in this chapter is the average of 10 repetitions with the same setup and have been obtained without any kind of test-time data augmentation or post-processing.

5.1 Ablation study

In order to investigate the relevance of each component of our architectures, we conducted an extensive ablation study in which we examined several variations of both the convolutional and Transformer parts of the networks.

5.1.1 Variations of convolutional parts

Keeping the complete Transformer part intact, we compare the same components that have been explored with the U-Net [34], also in an incremental way. For this first comparison, 16 initial filters are used in both architectures and the UNETR-2D Base model is selected. The versions compared are the following:

1. Baseline: four-level 2D U-Net and UNETR-2D with ReLU activations, Glorot uniform kernel initialization, 16 feature maps in the first level, which is doubled on each level, and no regularization or DA.
2. Baseline with DA.
3. Adding batch normalization.
4. Adding dropout as normalization in an incremental way. To be specific [0.1, 0.1, 0.2, 0.2, 0.3], which means that in the first level 0.1 dropout probability is applied, in the second level also 0.1, then 0.2 and so on.
5. Replacing ReLU activation with ELU activation.
6. Replacing Glorot uniform kernel initialization with He normal kernel initialization [61].

7. Adding attention gates [41].

The results on the Lucchi dataset for each case are shown in Table 5.1. For both U-Net and UNETR-2D architectures, the use of data augmentation improves the results significantly, with a notable difference in the case of the UNETR-2D. Then, while the batch normalization is significantly helpful in UNETR-2D, it is the only discarded element in the U-Net. In the case of the U-Net, the rest of the elements achieve similar or better results. In contrast, in the case of the UNETR-2D, only the He normal kernel initialization improved slightly the previous results. Overall, the U-Net achieves better and more consistent results. We can see that the effect of the different components is lower in the U-Net than in the UNETR-2D, even if the U-Net is a fully convolutional architecture, while in the UNETR-2D we are just modifying the convolutional part.

	Foreground IoU	
Method	U-Net [34]	UNETR-2D Base
Baseline	0.739 ± 0.002	0.462 ± 0.024
+ DA	0.871 ± 0.004	0.774 ± 0.012
+ Batch norm. ★	0.869 ± 0.002	0.824 ± 0.011
+ Dropout ⧫	0.881 ± 0.002	0.819 ± 0.008
+ ELU ⧫	0.881 ± 0.002	0.813 ± 0.010
+ He normal	0.881 ± 0.003	**0.829 ± 0.017**
+ Attention Gates ⧫	**0.884 ± 0.002**	0.788 ± 0.014

★ Discarded on U-Net
⧫ Discarded on UNETR-2D

Table 5.1: Ablation study in a contrastive way, comparing U-Net [34] and UNETR-2D Base on Lucchi test set. From top to the bottom, on each row, components are incrementally applied. Those components that impair performance have been discarded. The best result for each model is shown in bold.

Since we know which components are the most suitable, we will continue to use them from now on. This time we will explore a different number of initial filters. As always, each layer doubles the previous number of filters. In the case of the U-Net, we already know that the best assignment is 16 [34]. Therefore, we will only study the case of the UNETR-2D Base, this time with the Lucchi++ dataset. The results can be found in Table 5.2, where we found that the best assignment for the UNETR-2D is 32 initial filters. From now on, we will continue to use the best configuration for each architecture.

N initial filters	IoU_F	N Params	Inference time (ms)
8	0.822 ± 0.022	44.7M	6.5 ± 7.5
16	0.846 ± 0.008	45.6M	6.8 ± 7.7
32	**0.853 ± 0.010**	48.7M	7.4 ± 7.7
64	0.850 ± 0.014	60.2M	10 ± 7.4

Table 5.2: Effect of the number of initial filters on UNETR-2D Base's segmentation performance on Lucchi++ test set, number of parameters and inference time on 256×256 patches. The best result is shown in bold.

5.1.2 Transformer variants

Using the best convolutional part found in the previous section, we compare different Transformer versions. Since the Huge and Large Transformer versions do not fit in memory, we have not been able to train them. On the other hand, we tried with smaller versions (renamed as Mini and Small). Note that the skip-connections of the UNETR-2D, with 12 encoder layers, come from the output of the $3 \cdot l$ -th encoder layer (where l is the layer, and the first layer is $l = 0$), being the Transformer output $l = 4$. In each version, we change the number of layers and consequently the encoder layers from which the skip-connection signals are taken. For the sake of clarity, from now on we will work with a more general term S_m, and say that for each skip-connection the output of the encoder layer $S_m \cdot l$ is used. The specifications of each of the tested variants are given in the Table 5.3.

Model	Layers	Hidden size	MLP size	Heads	S_m
UNETR-2D Mini	4	64	256	4	1
UNETR-2D Small	8	128	512	8	2
UNETR-2D Base	12	256	1024	12	3

Table 5.3: Specifications of our UNETR-2D Transformer variants: Mini, Small and Base.

5.2 Network comparison

To test how well each network performs, we compare them using both datasets: Lucchi++ and Kasthuri++, keeping the best configuration and hyperparameters found in Lucchi++ for Kasthuri++ as well. For YNETR-2D, we also used the best configuration and hyperparameters found for UNETR-2D, except for the dropout, which uses not only dropout layers in the convolutional block, but also 2D spatial dropout layers after each of the concatenations of the signals from both encoders to force the network to use both encoders. The probabilities used for these 2D spatial dropout layers are the incremental probabilities mentioned above $[0.1, 0.1, 0.2, 0.2, 0.3]$. This configuration has proven useful experimentally and leads to better results. As for the generalization or cross-dataset performance, this is analyzed in more detail in the Appendix A.

The results of each network can be seen in Table 5.4. Focusing first on the Transformer variants, we realized that we can obtain similar or even better results with smaller Transformers, and therefore with fewer parameters than with Base Transformers. This might be related to the fact that we do not use a massive dataset with several million images like JFT-300M or ImageNet-21K [62], which are commonly used in ViT papers [51, 13]. Therefore, it is not necessary to use a large number of parameters to work with this kind of small datasets. Another interesting interpretation of these results is that reducing the dimensionality by using a smaller hidden size could help to remove the noise from the image and thus achieve better segmentation with less effort and parameters.

Focusing this time on a more general view of each architecture, we have seen that, even if UNETR-2D achieves good results, better results can be obtained with the U-Net or Attention U-Net networks, which are also the smallest and fastest. In the case of YNETR-2D, the gap with U-Net becomes smaller in both datasets, being even better than U-Net in the case of Kasthuri++. However, it should be remembered that YNETR-2D is the largest architecture in terms of parameters.

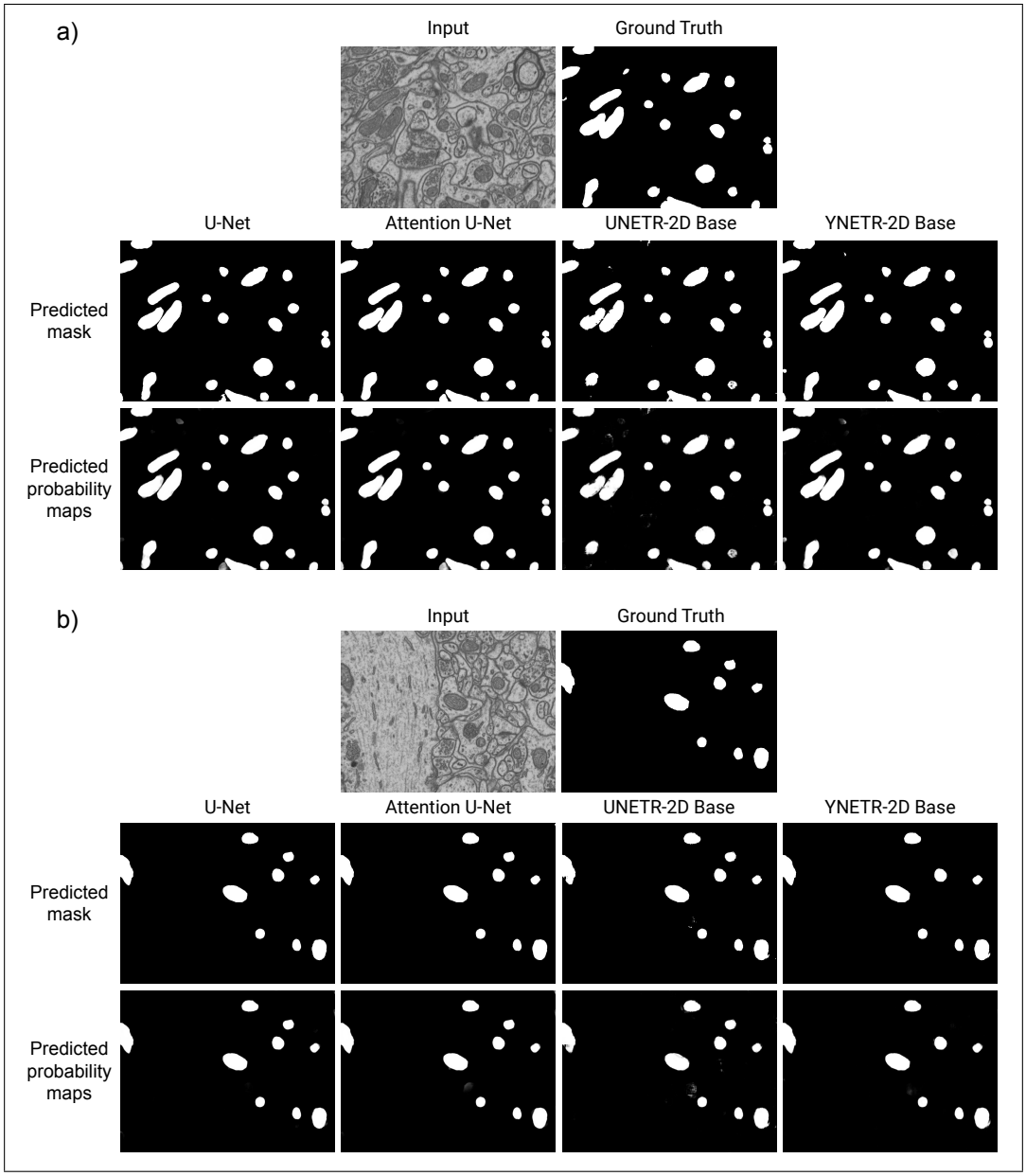


Figure 5.1: Qualitative network comparison. Two different cases are shown: a) and b), with exactly the same distribution. From left to right, the first row contain the input image and the corresponding ground truth mask. Followed in the next row by the predicted binary mask of U-Net, Attention U-Net, UNETR-2D Base and YNETR-2D Base respectively. And in the last row, their corresponding predicted probability maps. Note that multiple gray values can be found in the predicted probability maps. This would enable further analysis to be carried out.

Dataset	Model	Params	IoU_F	Inference time (ms)
Lucchi++	2D U-Net [34]	1.95M	0.903 ± 0.007	2.9 ± 1.1
	2D Attention U-Net [34]	1.99M	**0.906 ± 0.003**	2.9 ± 1.1
	UNETR-2D Mini (ours)	4.24M	0.861 ± 0.008	4.7 ± 3.8
	UNETR-2D Small (ours)	9.34M	0.857 ± 0.010	5.4 ± 4.8
	UNETR-2D Base (ours)	48.7M	0.853 ± 0.010	7.4 ± 7.7
	YNETR-2D Mini (ours)	10.3M	0.880 ± 0.024	5.3 ± 4.9
	YNETR-2D Small (ours)	15.4M	0.894 ± 0.009	6.1 ± 5.8
	YNETR-2D Base (ours)	54.7M	0.892 ± 0.013	8.4 ± 7.2
Kasthuri++	2D U-Net [34]	1.95M	0.910 ± 0.002	2.5 ± 0.9
	2D Attention U-Net [34]	1.99M	0.910 ± 0.001	2.5 ± 0.9
	UNETR-2D Mini (ours)	4.24M	0.886 ± 0.004	3.9 ± 3.3
	UNETR-2D Small (ours)	9.34M	0.888 ± 0.003	4.6 ± 3.8
	UNETR-2D Base (ours)	48.7M	0.887 ± 0.004	6.7 ± 5.4
	YNETR-2D Mini (ours)	10.3M	**0.917 ± 0.003**	4.5 ± 4.5
	YNETR-2D Small (ours)	15.4M	0.916 ± 0.001	5.2 ± 4.7
	YNETR-2D Base (ours)	54.7M	0.915 ± 0.002	7.5 ± 6.1

Table 5.4: Quantitative comparisons of segmentation performance. The best result for each dataset is shown in bold. The processing time average and standard deviation of a single 256×256 patch is shown in the inference time column.

For a qualitative comparison, see Fig. 5.1. As can be seen, UNETR-2D makes even more mistakes than U-Net architectures, especially under-segmenting mitochondria. Overall, U-Net architectures make smoother segmentations of the mitochondria, while UNETR-2D introduces a bit more noise both inside and outside the mitochondria, on elements that share some similarity. Although most of this noise disappears in the binary mask, giving mostly sharpen edges in those conflicting mitochondria. On the other hand, the aliasing found in UNETR-2D disappears completely in the case of YNETR-2D. But unlike the other architectures, it makes some new errors. For example, in the upper-right part of the second case shown, there is a complete mitochondrion that have not been correctly classified.

As mitochondria are mainly small objects, Transformer-based architectures may not be the best alternative. This is because attention to close details is required, where Transformer-based architectures may have more difficulties. We have also seen this in the YNETR-2D where the presence of the convolutional encoder has been helpful. This could also be the reason why U-Net performs better than UNETR-2D. Perhaps this type of Transformer-based architectures is better suited to other datasets with long-range dependencies. The use of YNETR-2D could also be interesting in such datasets, where we can exploit a bit more the potential of both encoders, both with local attention and with self-attention, which achieves a more global perspective.

5.3 Self-supervised learning

In this section, we compare the impact of different pretext tasks on semantic segmentation performance. This way, we can see how much improvement we can achieve without new data or labels, and also which pretext tasks provide a better result. This could be especially

Pretext task	Modification function		IoU_F
None - Baseline			0.852 ± 0.015
Denoise	Pixel dropout	(+0.009)	0.861 ± 0.011
	Cutout	(+0.015)	0.867 ± 0.010
	Defocus blur	(+0.015)	0.867 ± 0.008
	Additive Gaussian Noise	(+0.019)	0.871 ± 0.012
	Coarse salt and pepper	(+0.020)	0.872 ± 0.008
	Gaussian filter	(+0.021)	0.873 ± 0.009
	Motion blur	(+0.022)	0.874 ± 0.005
	Low resolution emulation	**(+0.024)**	$\mathbf{0.876 \pm 0.007}$

Table 5.5: Quantitative comparison of segmentation performance in Lucchi++ test set among the usage of different pretext tasks, following the SSL strategy with UNETR-2D Base model. Top to bottom: techniques are ordered by the IoU_F, where highest is at the bottom side and lowest values in the top side. To the left of each value of IoU_F is shown the difference between the means with respect to the baseline. The best result is shown in bold.

interesting in Transformer-based architectures for using larger unlabeled datasets, or to squeeze some performance out of the data we already have. As we have already seen, the use of DA has a critical impact on performance. For every pretext task, we used the same hyperparameters as for semantic segmentation, with the only exception being the loss function, where we used the mean squared error (MSE) loss function instead of BCE.

The results are shown in Table 5.5. We can see that, although the impact of SSL is not as great as that of DA, SSL performs better than the baseline in every case and achieves more stable results. Then, analyzing these techniques, we find that the functions that modify the image by using the previous values (such as Gaussian filter or motion blur) generally perform better than those that directly replace some values with other, constant, random or normally distributed values (such as pixel dropout or cutout). In any case, the IoU_F difference among most of the techniques is quite small. The best pretext task we have found is denoising with low resolution emulation.

CHAPTER 6

Conclusions and Future Work

In this project, we have analyzed and compared state-of-the-art U-Net architectures with novel Transformer-based UNETR-2D architectures for the task of semantic segmentation of EM images. After exploring several hyperparameters and Transformer variants and making use of the best configuration found, we have seen that both models achieve close IoU_F values, although U-Net remains above. Next, we combined both architectures in YNETR-2D and compared it with the previous models. This time, the results are even closer than before with respect to U-Net, improving the U-Net results for the Kasthuri++ dataset by a small amount. Finally, we explored the influence of using SSL techniques with UNETR-2D and found that we could improve the IoU_F value by 0.024 using a simple pre-training for denoising with the low resolution emulation technique. Although we have not tried this for every model due to time constraints, we can expect similar improvements for all of them.

Considering that the EM image datasets we have used contained a very limited amount of images, as future work it would be interesting to pre-train the model with a larger dataset. For instance, we could use denoising as we did for SSL, and then fine-tune for semantic segmentation with the smaller dataset. This way, we could exploit the potential of the Transformer-based models a bit more, and see if they are able to outperform U-Net-like architectures in this task. Another interesting aspect to do would be to analyze the generalization achieved with Transformer-based architectures, since this is an open problem in the area [26]. A good dataset to analyze all the above-mentioned could be the recently published CEM500K [63], with almost $500,000$ unlabeled EM images. Finally, another interesting way to continue this work would be to repeat the comparison with other segmentation datasets beyond mitochondria in EM images.

Appendix A: Unsupervised Domain Adaptation

In this Appendix, we describe a collateral result derived from this project. In particular, we have designed, implemented and published [26] a new convolutional model for semantic segmentation using domain adaptation.

Related work

Here we will explore a little bit about another related task, namely unsupervised domain adaptation. By domain and style, we refer to the intrinsic feature space and characteristics of a particular dataset and the distribution from where it is drawn. Domain adaptation can be seen as a particular type of transfer learning where instead of trying to transfer the knowledge from task A in domain A to task B in domain B, the tasks are kept the same while the domains are different. On the other hand, style transfer is mainly focused on adapting the domain from one dataset to another.

Existing domain adaptation methods can be divided depending on the label availability during the training process. Thus, they can be supervised, if both source and target domain labels are available; semi-supervised, if source labels and some target labels are available; and unsupervised, if only source labels are available while target data is entirely unlabeled [64]. Moreover, methods can also be categorized based on the learning model used, i.e., either shallow (usually relying on predefined image features and traditional machine learning models) or deep (if they use deep learning architectures). We will focus on the strategy known as deep unsupervised domain adaptation.

The approach is based on multi-task deep neural network architectures that receive both source and target samples as input. In this case, apart from solving the downstream task for the source (labeled) data, the model aims to exploit the features of the target domain to learn the feature shift between domains. Among these types of unsupervised and semi-supervised domain adaptation methods, we find the Y-Net [43], used for the segmentation of EM images. Its architecture consists of an encoder-decoder such as a U-Net [4], coupled with a second decoder in an autoencoder strategy. While one decoder is trained for segmentation, using the images with available labels, the second decoder is trained to reconstruct all available images, including the unlabeled ones, in an unsupervised manner. Since both decoders share the same encoder, the features learned by the autoencoder are used for segmentation too. Consequently, the model works with unlabeled (target domain) data features. Following this idea, in combination with adversarial losses, similar models such as Domain Adaptive Multi-Task Learning network (DAMT-Net) [65] have been proposed. This

network builds on top of the Y-Net architecture and adds two discriminators during training, following a Generative Adversarial Network (GAN) approach. The first discriminator uses the predicted segmentation, while the second discriminator uses the final feature maps of the network.

To address the problem of domain adaptation between different EM datasets, we present an approach that reduce the domain shift. Firstly, a cross-domain 2D Attention U-Net and UNETR-2D Base are introduced, trained only on source domains. Next, a simple histogram matching between domains is added as pre-processing prior to the use of the 2D Attention U-Net model. Finally, more sophisticated domain adaptation approaches are presented based on state-of-the-art domain adaptation multi-task deep neural networks.

Methodology

Histogram matching

A straightforward approach to make the images of one domain look closer to the images of another domain is histogram matching. Most commonly, this technique is applied to one source image so that its histogram matches the histogram of a target image [66]. Here instead, we use as target histogram the mean histogram of the target domain images, so the histogram of all source images are transformed to match it.

Some images of our datasets present zero-padding surrounding the tissue, which provokes an artificial high pick at the zero value in their histograms. Since we are only interested in matching the histogram of the tissue part of the images, we modified the actual number of zeros with linear regression using the first bins of the original histogram. We set the value to zero in the absence of initial values or when predicting a negative number. This process is done for both target and source histograms.

Multi-task neural networks

Following the idea behind Y-Net [43], we have built a similar architecture taking as a base model the previously mentioned Attention U-Net [34]. We refer to this network as Attention Y-Net. In short, the architecture consists of the classical encoder-decoder setup, where a new second decoder is placed. We can see the architecture as the combination of the Attention U-Net and an autoencoder, where both parts share the same encoder. The architecture is illustrated in Fig. 1.

The network is trained using a loss function ($\mathcal{L}$) made of two terms: a segmentation term based on the binary cross-entropy between the predicted and ground truth masks ($\mathcal{L}_{BCE}$), and a reconstruction term based on the mean squared error between the predicted and the original grayscale images ($\mathcal{L}_{MSE}$), as given by

$$\mathcal{L} = \alpha \mathcal{L}_{MSE} + (1 - \alpha)\mathcal{L}_{BCE}, \tag{1}$$

where the weight α is a numeric value between 0 and 1. For those images without available labels (binary masks), the $\mathcal{L}_{BCE}$ value will be 0.

In its original work, the training of the Y-Net [43] was proposed in two sequential steps. First, the network is trained unsupervised to perform only reconstruction ($\alpha = 1$).

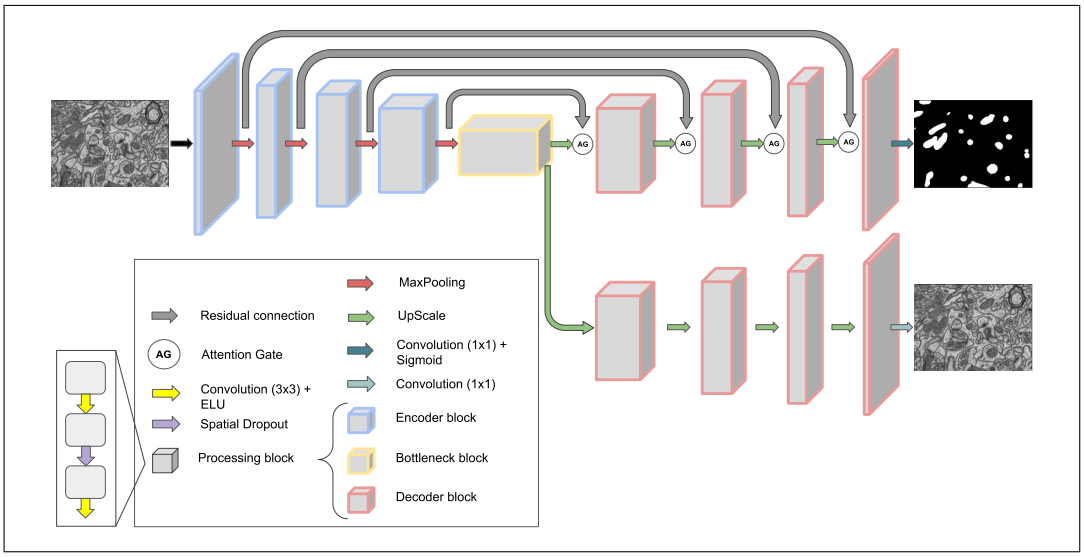


Figure 1: Architecture of the proposed Attention Y-Net used for domain adaptation. The architecture is formed by one encoder and two decoders: one for image reconstruction (without skip connections) and one for segmentation (with skip connections and attention gates).

Then, the model is fine-tuned to perform segmentation with the available labels ($\alpha = 0$). However, we have experienced instability in this step. Namely, quite often, the predicted reconstruction of the network was a flat grey-value image. Therefore, we propose a new additional step before the unsupervised pre-training, which combines both tasks using all the available data. We set $\alpha = 0.98$, which was experimentally found to help balancing both loss terms.

With our additional pre-training step, the network consistently outputs improved results, out of the local minimum achieved with the flat grey-value image. Next, we freeze the network encoder (blue blocks in Fig. 1). Otherwise, the network forgets the target domain features in the next step. Experimentally, we observed that the network performs better if we let the bottleneck and the two decoders unfrozen. Remarkably, as observed with the self-supervised approach, the performance of the whole process was greatly enhanced thanks to the use of histogram matching after the first step.

The first step was carried out for 50 epochs. We used an initial learning rate of $1e-3$ that got reduced when reaching plateaus, stochastic gradient descent (SGD) as optimizer and a patience of 7 epochs over the monitored validation loss. In the second training step, we train for 40 epochs (with a patience of 6). We use a learning rate of $2e-4$, and a "reduce on plateau" scheduler once again, but this time with Adam optimizer. Finally, in the last training step, we train for 100 epochs with a patience of 15 while monitoring source validation loss. We follow a one-cycle learning rate policy [67] with a maximum learning rate of $2e-4$, and use Adam as optimizer. For all training steps, the optimal batch size was found to be 1. The input to the model consists of 1000 random cropped patches of 256×256 pixels, from which 10% is used for validation. This training configuration was empirically found.

Results

All the methods proposed here were applied to all the possible source-target combinations of the two EM datasets introduced in section 4.1. Moreover, for a more detailed evaluation and comparison with the state-of-the-art, we executed as well the same experiments using the publicly available implementation of DAMT-Net [65]. As it is an extended practice on EM image processing, we also tested Attention U-Net method on the same image data after preprocessing them using contrast limited adaptive histogram equalization (CLAHE) [68]. Notice CLAHE is a contrast equalization method, thus not intended to match two intensity distributions. However, its effect on the image contrast may bring the histogram of our datasets closer to each other.

As we can see in Table 1, when we try to use the Attention U-Net or UNETR-2D in a different domain, the performance is drastically lost, from almost 0.9 of IoU_F in both datasets, with supervised training on target directly, to almost 0 of IoU_F in both datasets, when is not trained for that specific domain. Although UNETR-2D shows a little more generalization capacity, it is not enough in our case. Nevertheless, it is an interesting feature to keep in mind for future work. Due to time constraints, no further experiments with Transformer-based models (UNETR-2D or YNETR-2D) have been carried out.

Just by applying CLAHE or histogram matching, we are able to significantly improve performance. Meanwhile, with Attention Y-Net and histogram matching, we achieve not only good results, but also more stable under multiple repetitions and under different datasets, improving significantly results achieved with DAMT-Net. Nevertheless, as happens with most of the methods in Table 1, the performance could be better or worse depending on the datasets used as source and target. If we make use of more datasets, even with Attention Y-Net we can see variations in the performance. This can be clearly seen in [26], as well as other proposals for this task and further analysis. Although these proposals achieve good results, this is only one step in this area, and much work remains to be done to achieve results similar, or at least close, to the supervised strategies.

Method	**Source:**	Lucchi++	Kasthuri++
	Target:	Kasthuri++	Lucchi++
UNETR-2D Base (ours)		0.023±0.025	0.060±0.027
2D Attention U-Net [34]		0.017±0.008	0.000±0.000
2D Attention U-Net [34] + CLAHE		0.620±0.051	0.433±0.085
2D Attention U-Net [34] + HM (ours)		**0.679±0.043**	0.268±0.048
2D Attention Y-Net + HM (ours)		0.668±0.020	**0.704±0.045**
DAMT-Net [65]		0.279±0.078	0.569±0.088

Table 1: Cross-dataset domain adaptation methods evaluation. Results are shown based on the mean IoU_F value (± standard deviation) obtained in the test partition of the target datasets. The best results of each column are shown in bold. HM refers to Histogram Matching

Appendix B: Hyperparameter Search

Resources

Code availability

The code is publicly available at https://github.com/AAitorG/UNETR_2D.

Data availability

The Lucchi++ and Kasthuri++ datasets can be downloaded from https://sites.google.com/view/connectomics/.

Hyperparameter search space

This section describes in detail the search we performed for the optimal training configuration and set of hyperparameters in our proposed UNETR-2D Base. The corresponding search space and best values are summarized in the table below using the following notation:

- (a, b, c): All values set, e.g., dropout(0.1, 0.2, 0.3) in a 3-depth level network indicate that 0.1 dropout value has been set in the first level, 0.2 dropout in the second level and 0.3 in the third level.
- choice[a, b, ...]: One value between a, b and so on. E.g. [10, 15, 20, 30, 60] possible values are: 10 or 15 or 20 or 30 or 60 (but only one).
- a, b, c, ...: All tested values, e.g., flips, rotations.
- BCE: Binary cross entropy.
- SGD: Stochastic gradient descent.
- AdamW: Adam algorithm with weight decay [57].
- Reduce on Plateau: A learning rate policy to reduce the value of the learning rate when the monitored metric has stopped improving (https://keras.io/api/callbacks/reduce_lr_on_plateau/).
- OneCycle: One-cycle learning rate policy for super-convergence [67].

- Cosine Decay: Learning rate decay with cosine annealing [69].
- ReLU: Rectified Linear Unit activation function.
- ELU: Exponential Linear Unit activation function.
- GELU: Gaussian Error Linear Unit activation function [70].
- He normal: He normal [61] as kernel initialization.
- Glorot uniform: Glorot uniform [71] as kernel initialization.

Hyperparameter	Search space	Best assignment
Data		
Validation	True	True
Random validation	True	True
% of train as validation	10%	10%
Patch size	256×256	256×256
Shuffle train on each epoch	True	True
Data augmentation	flips, rotation_range(180)	flips, rotation_range(180)
Training		
Number of epochs	*choice*[50, 100, 150, 200, 360]	360
Batch size	*choice*[1, 2, 6, 12]	6
Loss type	*choice*[BCE + Dice, BCE]	BCE
Optimizer	*choice*[SGD, Adam, AdamW]	AdamW
Weight decay (in AdamW)	*choice*[1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 5e-2, 0.3]	1e-5
Learning rate	1e-4	1e-4
Scheduler	*choice*[OneCycle, Reduce on Plateau, Cosine Decay, None]	Cosine Decay
Patience	*choice*[5, 30, 40, 50, 60, 100, 200, 360]	60
Architecture		
Initial feature maps	*choice*[8, 16, 32, 64]	32
Dropout type	*choice*[None, Spatial Dropout(0.1, 0.1, 0.2, 0.2, 0.3), Dropout(0.1, 0.1, 0.2, 0.2, 0.3)]	None
Kernel initializer	*choice*[He normal, Glorot uniform]	He normal
Convolutional part's activation	*choice*[ReLU, ELU, GELU]	ReLU

Table 2: Hyperparameter search space for semantic segmentation task using the UNETR-2D Base.

Bibliography

[1] Michelle Barbi De Moura, Lucas Santana Dos Santos, and Bennett Van Houten. Mitochondrial dysfunction in neurodegenerative diseases and cancer. *Environmental and molecular mutagenesis*, 51(5):391–405, 2010. See page 1.

[2] Angela C Poole, Ruth E Thomas, Laurie A Andrews, Heidi M McBride, Alexander J Whitworth, and Leo J Pallanck. The pink1/parkin pathway regulates mitochondrial morphology. *Proceedings of the National Academy of Sciences*, 105(5):1638–1643, 2008. See page 1.

[3] Shutao Li, Weiwei Song, Leyuan Fang, Yushi Chen, Pedram Ghamisi, and Jón Atli Benediktsson. Deep learning for hyperspectral image classification: An overview. *IEEE Transactions on Geoscience and Remote Sensing*, 57(9):6690–6709, 2019. See page 1.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. See pages 1, 3, 18, and 29.

[5] Mohammad Hesam Hesamian, Wenjing Jia, Xiangjian He, and Paul Kennedy. Deep learning techniques for medical image segmentation: achievements and challenges. *Journal of digital imaging*, 32(4):582–596, 2019. See page 1.

[6] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing-Hao Xue, and Qingmin Liao. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia*, 21(12):3106–3121, 2019. See page 1.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. See pages 1, 5, 6, and 7.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. See pages 1, 8.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. See pages 1, 5.

[10] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. See page 1.

[11] Pushpankar Kumar Pushp and Muktabh Mayank Srivastava. Train once, test anywhere: Zero-shot learning for text classification. *arXiv preprint arXiv:1712.05972*, 2017. See page 1.

[12] Yuan Gong, Yu-An Chung, and James Glass. Ast: Audio spectrogram transformer. *arXiv preprint arXiv:2104.01778*, 2021. See page 1.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. See pages 1, 8, 9, 11, and 23.

[14] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip H. S. Torr, and Li Zhang. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers, 2020. See page 1.

[15] Hassan Akbari, Liangzhe Yuan, Rui Qian, Wei-Hong Chuang, Shih-Fu Chang, Yin Cui, and Boqing Gong. Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text. *Advances in Neural Information Processing Systems*, 34:24206–24221, 2021. See page 1.

[16] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022. See page 1.

[17] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *arXiv preprint arXiv:2106.04554*, 2021. See page 1.

[18] Muhammad Imran Razzak, Saeeda Naz, and Ahmad Zaib. Deep learning for medical image processing: Overview, challenges and the future. *Classification in BioApps*, pages 323–350, 2018. See page 1.

[19] Konstantinos Kamnitsas, Wenjia Bai, Enzo Ferrante, Steven McDonagh, Matthew Sinclair, Nick Pawlowski, Martin Rajchl, Matthew Lee, Bernhard Kainz, Daniel Rueckert, et al. Ensembles of multiple models and architectures for robust brain tumour segmentation. In *International MICCAI brainlesion workshop*, pages 450–462. Springer, 2017. See page 1.

[20] Zeshan Hussain, Francisco Gimenez, Darvin Yi, and Daniel Rubin. Differential data augmentation techniques for medical imaging classification tasks. In *AMIA annual symposium proceedings*, volume 2017, page 979. American Medical Informatics Association, 2017. See page 1.

[21] Amy Zhao, Guha Balakrishnan, Fredo Durand, John V Guttag, and Adrian V Dalca. Data augmentation using learned transformations for one-shot medical image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8543–8553, 2019. See page 1.

[22] Jakub Nalepa, Michal Marcinkiewicz, and Michal Kawulok. Data augmentation for brain-tumor segmentation: a review. *Frontiers in computational neuroscience*, 13:83, 2019. See page 1.

[23] Nripendra Kumar Singh and Khalid Raza. Medical image generation using generative adversarial networks. *arXiv preprint arXiv:2005.10687*, 2020. See page 1.

[24] Laith Alzubaidi, Mohammed A Fadhel, Omran Al-Shamma, Jinglan Zhang, J Santamaría, Ye Duan, and Sameer R. Oleiwi. Towards a better understanding of transfer learning for medical imaging: a case study. *Applied Sciences*, 10(13):4523, 2020. See page 1.

[25] Mohammad Amin Morid, Alireza Borjali, and Guilherme Del Fiol. A scoping review of transfer learning research on medical image analysis using imagenet. *Computers in biology and medicine*, 128:104115, 2021. See page 1.

[26] Daniel Franco-Barranco, Julio Pastor-Tronch, Aitor Gonzalez-Marfil, Arrate Muñoz-Barrutia, and Ignacio Arganda-Carreras. Deep learning based domain adaptation for mitochondria segmentation on em volumes deep learning based domain adaptation on em volumes. *Computer Methods and Programs in Biomedicine*, page 106949, 2022. See pages 1, 2, 4, 13, 27, 29, and 32.

[27] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*, 2021. See page 2.

[28] Yao-Hung Hubert Tsai, Yue Wu, Ruslan Salakhutdinov, and Louis-Philippe Morency. Self-supervised learning from a multi-view perspective. *arXiv preprint arXiv:2006.05576*, 2020. See page 2.

[29] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2020. See page 2.

[30] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019. See page 2.

[31] Abhinav Dhere and Jayanthi Sivaswamy. Self-supervised learning for segmentation. *arXiv preprint arXiv:2101.05456*, 2021. See page 2.

[32] Veronika Cheplygina, Marleen de Bruijne, and Josien PW Pluim. Not-so-supervised: a survey of semi-supervised, multi-instance, and transfer learning in medical image analysis. *Medical image analysis*, 54:280–296, 2019. See page 2.

[33] Ali Hatamizadeh, Yucheng Tang, Vishwesh Nath, Dong Yang, Andriy Myronenko, Bennett Landman, Holger Roth, and Daguang Xu. Unetr: Transformers for 3d medical image segmentation, 2021. See pages 2, 10.

[34] Daniel Franco-Barranco, Arrate Muñoz-Barrutia, and Ignacio Arganda-Carreras. Stable deep neural network architectures for mitochondria segmentation on electron microscopy volumes. *Neuroinformatics*, pages 1–14, 2021. See pages 2, 3, 4, 5, 18, 19, 20, 21, 22, 25, 30, and 32.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. See page 3.

[36] Joshua Batson and Loic Royer. Noise2self: Blind denoising by self-supervision. In *International Conference on Machine Learning*, pages 524–533. PMLR, 2019. See page 3.

[37] Amreen Kaur, Ankit Raj, N Jayanthi, and S Indu. Inpainting of irregular holes in a manuscript using unet and partial convolution. In *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 778–784. IEEE, 2020. See page 3.

[38] Federico Vaccaro, Marco Bertini, Tiberio Uricchio, and Alberto Del Bimbo. Fast video visual quality and resolution improvement using sr-unet. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 1221–1229, 2021. See page 3.

[39] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. *CoRR*, abs/1807.10165, 2018. See page 3.

[40] Foivos I Diakogiannis, François Waldner, Peter Caccetta, and Chen Wu. Resunet-a: A deep learning framework for semantic segmentation of remotely sensed data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 162:94–114, 2020. See page 3.

[41] Jo Schlemper, Ozan Oktay, Michiel Schaap, Mattias Heinrich, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention gated networks: Learning to leverage salient regions in medical images. *Medical image analysis*, 53:197–207, 2019. See pages 4, 18, and 22.

[42] Ahmed Mohammed, Sule Yildirim, Ivar Farup, Marius Pedersen, and Øistein Hovde. Y-net: A deep convolutional neural network for polyp detection. *arXiv preprint arXiv:1806.01907*, 2018. See page 4.

[43] Joris Roels, Julian Hennies, Yvan Saeys, Wilfried Philips, and Anna Kreshuk. Domain adaptive segmentation in volume electron microscopy imaging. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 1519–1522. IEEE, 2019. See pages 4, 29, and 30.

[44] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading, 2016. See page 5.

[45] Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016. See page 5.

[46] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018. See page 5.

[47] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019. See page 7.

[48] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019. See page 8.

[49] Jiuk Hong, Chaehyeon Lee, Soyoun Bang, and Heechul Jung. Fair comparison between efficient attentions. *arXiv preprint arXiv:2206.00244*, 2022. See page 9.

[50] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR, 2020. See page 9.

[51] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12116–12128. Curran Associates, Inc., 2021. See pages 9, 11, 12, and 23.

[52] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. See page 13.

[53] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug. `https://github.com/aleju/imgaug`, 2020. Online; accessed 01-Feb-2020. See page 13.

[54] Aurélien Lucchi, Kevin Smith, Radhakrishna Achanta, Graham Knott, and Pascal Fua. Supervoxel-based segmentation of mitochondria in em image stacks with learned shape features. *IEEE transactions on medical imaging*, 31(2):474–486, 2011. See page 15.

[55] Vincent Casser, Kai Kang, Hanspeter Pfister, and Daniel Haehn. Fast mitochondria detection for connectomics. In *Medical Imaging with Deep Learning*, pages 111–120. PMLR, 2020. See pages 15, 16.

[56] Narayanan Kasthuri, Kenneth Jeffrey Hayworth, Daniel Raimund Berger, Richard Lee Schalek, José Angel Conchello, Seymour Knowles-Barley, Dongil Lee, Amelio Vázquez-Reina, Verena Kaynig, Thouis Raymond Jones, et al. Saturated reconstruction of a volume of neocortex. *Cell*, 162(3):648–661, 2015. See page 16.

[57] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. See pages 20, 33.

[58] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. See page 20.

[59] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. See page 20.

[60] Francois Chollet et al. Keras, 2015. See page 20.

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. See pages 21, 34.

[62] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2014. See page 23.

[63] Ryan Conrad and Kedar Narayan. Cem500k, a large-scale heterogeneous unlabeled cellular electron microscopy image dataset for deep learning. *eLife*, 10:e65894, apr 2021. See page 27.

[64] Hao Guan and Mingxia Liu. Domain adaptation for medical image analysis: A survey. *IEEE Transactions on Biomedical Engineering*, 69(3):1173–1185, 2022. See page 29.

[65] Jialin Peng, Jiajin Yi, and Zhimin Yuan. Unsupervised mitochondria segmentation in EM images via domain adaptive multi-task learning. *IEEE Journal of Selected Topics in Signal Processing*, 14(6):1199–1209, 2020. See pages 29, 32.

[66] Rafael C Gonzalez and Richard E Woods. *Digital Image Processing*. Pearson, Upper Saddle River, NJ, 3 edition, 2008. See page 30.

[67] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, volume 11006, pages 369–386. SPIE, 2019. See pages 31, 33.

[68] Karel Zuiderveld. Contrast limited adaptive histogram equalization. *Graphics gems*, pages 474–485, 1994. See page 32.

[69] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. See page 34.

[70] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. See page 34.

[71] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. See page 34.