Error-Free Semantic Segmentation Inference of Images Larger than GPU Memory

Michael Majurski[1] and Peter Bajcsy[1]

National Institute of Standards and Technology
Information Technology Lab
Gaithersburg, MD 20899, USA
{michael.majurski,peter.bajcsy}@nist.gov

Abstract. We address the problem of performing error-free out-of-core semantic segmentation inference of arbitrarily large images using fully convolutional neural networks (FCN). FCN models have the property that once a model is trained it can be applied on arbitrarily sized images, though it is still constrained by the available GPU memory. This work is motivated by overcoming the GPU memory size constraint without numerically impacting the final result. One can select a tile size which will fit into GPU memory with a halo border of half the network receptive field. Stride across the image by the tile size without the halo. The input tile halos will overlap while the output tiles join exactly at the seams. This enables inference of whole slide microscopy images generated by a slide scanner, for example.
We documented the formulas for determining tile size and stride and validated them on U-Net and FC-DenseNet architectures.
In addition, we document the errors due to tiling configurations which do not satisfy the constraints, and explore using architecture effective receptive fields to estimate the tiling parameters.

1 Introduction

The task of semantic segmentation, assigning a label to each image pixel, is often performed using deep learning based convolutional neural networks (CNNs) [2, 23], for instance, by using a special type of CNN which only uses convolutional layers. These so-called "fully convolutional neural networks" (FCN) have a very useful property which allows the alteration the input image size. Both U-Net [23] and the original FCN network [18] are examples of FCN type CNNs. FCNs enable training the network on images much smaller than those of interest at inference time as long as the resolution is comparable. For example, one can train a U-Net model on 512×512 pixel tiles and then perform GPU-based inference on arbitrarily sized images provided the GPU memory can accommodate the model coefficients, network activations, application code, and an input image tile. This decoupling of training and inference image sizes means the semantic segmentation models can be applied to images much larger than the memory available on current GPUs.

The ability of FCN networks to inference arbitrarily large images differs from other types of CNNs where the training and inference image sizes must be identical. Usually this static image size requirement is not a problem since the input image size is expected to be static, or images can be resized within reason to fit the network. For example, if one trained a CNN on ImageNet [24] to classify pictures into two classes: {Cat, Dog}, the content of the image does not change drastically if the cat photo is resized to 224×224 pixels before inference provided the resolution is not altered considerably. Convolutional networks are not yet capable of strong generalization across scales [13, 17] so the inference time pixel resolution needs to approximately match the training time resolution or accuracy can suffer.

In contrast, there are applications where resizing (re-scaling or down-sampling) the image is not acceptable due to loss of information. For example, in digital pathology, one cannot take a whole slide microscopy image generated by a slide scanner (upwards of 10 Gigapixels) and fit it into GPU memory; nor can one reasonably resize the image as too much image detail would be lost.

Our work is motivated by the need to design a methodology for arbitrarily large image inference on GPU memory constrained hardware in those applications where the loss of information due to image resizing is not acceptable. This method can be summarized as follows. Select a tile size which will fit into GPU memory with a halo border of half the network receptive field size. Stride across the image by the tile size without the halo. The input tile halos will overlap while the output tiles join exactly at the seams. The original U-Net paper [23] briefly hinted at the feasibility of an inference scheme similar to the one we present in this paper but did not fully document and explain the inference scheme. The novelty of our work lies in presenting a methodology for error-free inference over images that are larger than available GPU memory for processed data. This work leverages FastImage [3], a high-performance accessor library for processing gigapixel images in a tile-based manner.

2 Related Work

Out of core, tile-based processing is a common approach in the high performance parallel computing field where any local signal processing filter can be applied to carefully decomposed subregions of a larger problem [5]. The goal is often computation acceleration via task parallelization. However, a side effect of the tile-based processes is the potential for reducing the active working memory required at any given point in the computation, trading increased runtime for a reduced memory requirement [5].

It has been known since the initial introduction of FCN models that they can be applied via shift-and-stitch methods as if the FCN were a single filter [18, 27]. The original U-Net paper by Ronneberger et al. [23] also hints at inference of arbitrary sized images in its Figure 2. However, none of the past papers mentioning shift-and-stitch discuss the methodology for performing out-of-core arbitrary sized image inference.

There are two common approaches for applying CNN models to large images: sliding window (overlapping tiles) and patch-based. Sliding window (i.e. overlapping tiles) has been used for object detection [26, 28] and for semantic segmentation [16, 29]. Patch-based inference also supports arbitrarily large images, but it is very inefficient [29, 20].

Huang et al. and Iglovikov et al. both propose sliding window approaches [9, 11]. Huang et al. directly examines the problem of operating on images which cannot be inferred in a single forward pass. The authors focus on different methods for reducing but not eliminating the error in labeling that arises from different overlapping tile-based processing schemes [9]. They examine label averaging and the impacts of different tile sizes on the resulting output error and conclude that using as large a tile as possible will minimize the error [9]. Huang et al. also examine the effects of zero-padding, documenting how much error it introduces [9]. At no point do they produce error-free tile-based inference. Iglovikov et al. remark upon error in the output logits at the tile edges during inference and suggest overlapping predictions or cropping the output to reduce that error [11].

Patch based methods for dealing with large images predict a central patch given a larger local context. Volodymyr Mnih predicts a 16×16 pixel patch from a 64×64 pixel area for road segmentation from aerial images [21]. This approach will scale to arbitrarily large images and if a model architecture with a receptive field of less than 48 pixels is used, it will have zero error from the tiling process. Saito et al. use a similar patch based formulation to Mihn, predicting a center patch given a large local context, however they add a model averaging component by predicting eight slightly offset versions of the same patch and then combining the predictions [25].

Our ability to perform error-free tile-based inference relies on the limited receptive field of convolutional layers as explored by Luo et al. [19]. Luo et al. [19] discuss the receptive fields of convolutional architectures, highlighting that for a given output pixel, there is limited context/information from the input that can influence that output[1]. Input data outside the receptive field cannot influence that output pixel [19]. The receptive field of a model is highly dependent upon architecture and layer connectivity. We us the theoretical underpinning of receptive fields to identify a sufficient size of an image tile for error-free inference.

To the best of our knowledge, no published method fully explores a methodology for error-free tile-based (out-of-core) inference of arbitrarily large images. While tile-based processing schemes have been outlined, the past publications do not provide a framework for achieving error-free tile-based inference results. Our approach does not handle layers with dynamic inference or variable receptive fields like stand alone self-attention [22], squeeze and excitation layers [8], deformable convolutions [6], or non-local layers [30].

[1] Article on "Computing Receptive Fields of Convolutional Neural Networks" [1] https://distill.pub/2019/computing-receptive-fields

3 Methods

Inferencing arbitrarily large input images requires that we only inference a small enough tile to fit in GPU memory for any single forward pass and then operate tile-by-tile. To form a tile, the whole image being is broken down into non-overlapping regions. This is equivalent to striding across the image with a fixed stride. Then if one includes enough local context around each tile to cover the theoretical receptive field of the network, each pixel will have all of the local context it requires. Therefore, while the full image was broken into tiles for inference, each pixel individually had all of the information required to be predicted as if the whole image were passed through the network as one block of memory.

There are three important concepts required for this tile-based (out-of-core) processing scheme.

1. Zone of Responsibility (ZoR): a rectangular region (block, partition, zone, or area) of the output image currently being computed, a.k.a. the output tile size.
2. Halo: minimum horizontal and vertical border around the ZoR indicating the local context that the FCN requires to accurately compute all pixels within the ZoR. This value is equal to half of the receptive field size of the model architecture being used.
3. Stride: the stride across the source image used to create tiles. The stride is fixed at the ZoR size.

Each dimension of the square input tile is then defined as $inputTileSize = ZoR + 2 \times Halo$. Figure 1 shows an example where a 832×832 pixel zone of responsibility is shown as a square with a 96 pixel halo surrounding it. Since the local context provided by the pixels in the halo is required to correctly compute the output tile, the GPU input is $832+2\times96 = 1024$ pixels per spatial dimension.

In other words, the image is broken down into the non-overlapping ZoR (i.e. output tiles). For each ZoR, the local context defined by the halo (where that context is available) is included in the input tile to be passed through the network. For a specific output pixel, if an input pixel is more than half the receptive field away, it cannot influence that output pixel. Therefore including a halo of half the receptive field ensures that pixels on the edge of the ZoR (output tile) have all the required context.

After passing the input tile through the model, the ZoR within the prediction (without the halo) is copied to the output image being constructed in CPU memory. The halo provides the network with all the information it needs to make correct, deterministic predictions for the entirety of the output zone of responsibility. Hence the name, since each ZoR is responsible for a specific zone of the output image.

This tile-based inferencing can be thought of as a series of forward passes, each computing a subregion (ZoR) of the feature maps that would be created while performing inference on the whole image in one pass. In summary, each tile's feature maps are created (forward pass), its ZoR output extracted, and then the GPU memory is recycled for the next tile. By building each ZoR in a

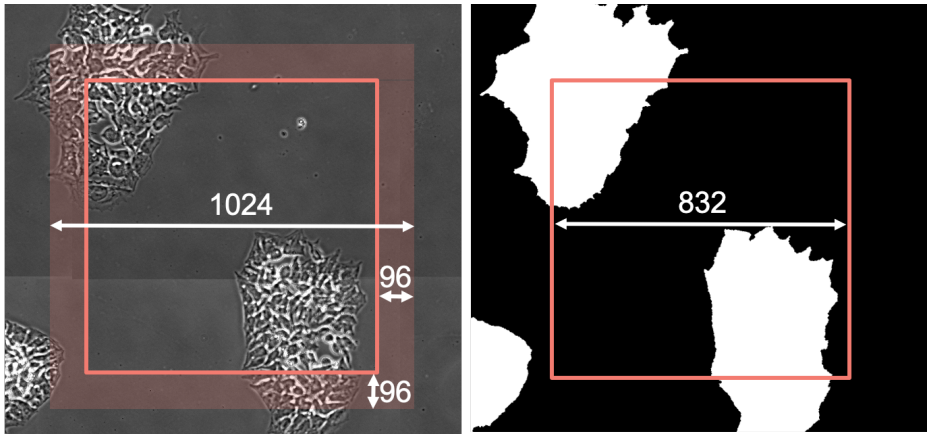

Fig. 1. Left: ZoR (832 × 832 pixel square) with a 96 pixel surrounding halo (shaded area) which combine to make the 1024 × 1024 pixel input tile required to infer the 832 × 832 pixel output tile (ZoR). Right: segmentation output tile showing the ZoR contribution to the final segmented image.

separate forward pass we can construct the network output within a fixed GPU memory footprint for arbitrarily large images.

3.1 U-Net Case Study

We use U-Net [23] as the case study FCN. Nonetheless, the presented methodology applies to any FCN network, just the specific numerical values will be different.

Note: for the purpose of brevity, we will use 'up-conv' (as the U-Net paper does) to refer to fractionally strided convolutions with a stride of $\frac{1}{2}$ which doubles the feature map spatial resolution [7].

Determining The Halo The halo must be half the receptive field. The general principle is to sum along the longest path through the network, the product of half the receptive field for each convolutional kernel and the stride that kernel has across the input image. The stride a specific convolution kernel has across the input image is a combination of that kernels stride with respect to its feature map and the downsampling factor between the input image size and the spatial size of that feature map. The downsampling factor is determined by the number of spatial altering layers between the input image and a specific layer.

Let U-Net be described by an ordered sequence of convolutional layers $c = 0, ..., N-1$ with each layer being associated with a level l_c and a square kernel $k_c \times k_c$. For the network, N defines the number of convolutional layers along the longest path from input to output.

Let us define the level l_c of an encoder-decoder network architecture as the number of max-pool[2] layers minus the number of up-conv layers between the input image and the current convolutional layer c along the longest path through the network. Levels start at 0; each max pool encountered along the longest path increases the level by 1 and each up-conv reduces the level by 1.

General Halo Calculation The required halo can be calculated according to the Equation 1 for a general FCN architecture.

$$Halo = \sum_{c=0}^{N-1} 2^{l_c} \lfloor \frac{k_c}{2} \rfloor \tag{1}$$

The halo is a sum over every convolutional layer index c from 0 to $N-1$ encountered along the longest path from the input image to the output. Equation 1 has two terms. The 2^{l_c} term is the number of pixels at the input image resolution that correspond to a single pixel within a feature map at level l_c. Therefore, at level $l_c = 4$ each pixel in the feature map equates to $2^4 = 16$ pixels at the input image resolution. This 2^{l_c} term is multiplied by the second term $\lfloor \frac{k_c}{2} \rfloor$ which determines, for a given c, how many pixels of local context are required at that feature map resolution to perform the convolution.

U-Net Configuration We have made two modifications to the published U-Net.

1. Normalization: Batch normalization [12] was added after the activation function of each convolutional layer as it is current good practice in the CNN modeling community.
2. Convolution Type: Convolutional type was changed to `SAME` from `VALID` as used in the original paper [23].

The use of batch normalization should prevent the out-of-core tiling scheme from achieving numerically identical results when performing inference of the whole image in a single pass because the batch statistics used for normalization will be different. However, as we will show later (Table 1), this appears to contribute very little to the observed error coming from using the tiling scheme.

The original U-Net paper uses `VALID` type convolutions which shrink the spatial size of the feature maps by 2 pixels for each layer [7][3]. Switching to `SAME` type convolutions preserves feature map size. See Appendix D for additional explanation.

[2] Convolutions with a stride of 2 can also be used to halve the spatial size of the feature maps but they will affect the receptive field.

[3] For an excellent review of convolutional arithmetic, including transposed convolutions (i.e., up-conv), see "A guide to convolutional arithmetic for deep learning by Dumoulin and Visin" [7].

There is one additional constraint on U-Net that needs to be mentioned. Given the skip connections between the encoder and decoder elements for matching feature maps, we need to ensure that the tensors being concatenated together are the same size. This can be restated as requiring the input size be divisible by the largest stride across the input image by any kernel. For U-Net this stride is 16 (derivation in Appendix D). Another approach to ensuring consistent size between the encoder and decoder is to pad each up-conv layer to make its output larger and then crop to the target feature map size. That is a less elegant solution than enforcing a tile size constraint and potentially padding the input image.

Halo Calculation for U-Net The published U-Net (Figure 1 from [23]) has one level per horizontal stripe of layers. The input image enters on level $l_{c=0} = l_{c=1} = 0$. The first max-pool layer halves the spatial resolution of the network, changing the level. Convolution layers $c = \{2, 3\}$ after that first max-pool layer up to the next max-pool layer belong to level $l_{c=2} = l_{c=3} = 1$. This continues through level 4, where the bottleneck of the U-Net model occurs. In U-Net's Figure 1 [23], the bottleneck is the feature map at the bottom which occurs right before the first up-conv layer. After the bottleneck, the level number decreases with each subsequent up-conv layer, until level $l_{N-1} = 0$ right before the output image is generated.

The halo computation in Equation 1 can be simplified for U-Net as shown in Appendix A. Appendix B shows a numerical example for computing the halo size using Equation 1.

Following Equation 1 for U-Net results in a minimum required halo of 92 pixels in order to provide the network with all of the local context it needs to predict the outputs correctly. This halo needs to be provided both before and after each spatial dimension and hence the input image to the network will need to be $2 \times 92 = 184$ pixels larger. This value is exactly the number of pixels the original U-Net paper has the output being shrunk by to avoid using `SAME` convolutions; a 572 pixel input shrunk by 184 results in the 388 pixel output [23]. However, this runs afoul of our additional restriction on the U-Net input size, which requires images to be a multiple of 16. So rounding up to the nearest multiple of 16 results in a halo of 96 pixels. Since one cannot just adjust the ZoR size to ensure $(ZoR + Halo)\%16 = 0$ due to convolutional arithmetic, we must explore constraints on image partitioning.

3.2 Constraints on Image Partitioning

Our tile-based processing methodology operates on the principle of constructing the intermediate feature map representations within U-Net in a tile-based fashion, such that they are numerically identical to the whole image being passed through the network in a single pass. Restated another way, the goal is to construct an input image partitioning scheme such that the zone of responsibility is building a spatial subregion of the feature maps that would exist if the whole image were passed through the network in a single pass.

Stride Selection To properly construct this feature map subregion one cannot stride across the input image in a different manner than would be used to inference the whole image. The smallest feature map in U-Net is spatially 16× smaller than the input image. Therefore, 16 pixels is the smallest offset one can have between two tile-based inference passes while having both collaboratively build subregions of a single feature map representation. Figure 2 shows a simplified 1D example with a kernel of size 3 performing addition. When two applications of the same kernel are offset by less than the size of the kernel they can produce different results. For U-Net, each 16×16 pixel block in the input image becomes a single pixel in the lowest spatial resolution feature map. A stride other than a multiple of 16 would result in subtly different feature maps because each feature map pixel was constructed from a different set of 16×16 input pixels.

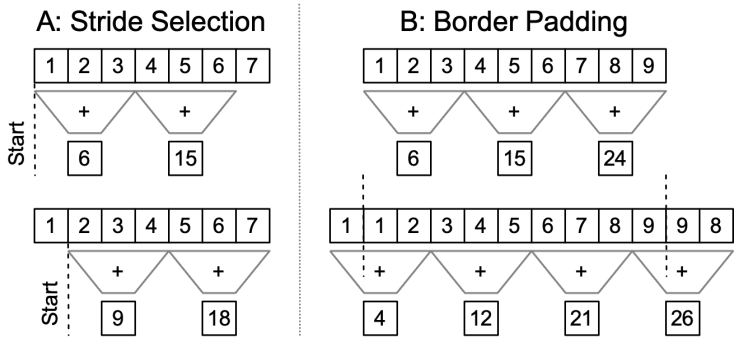

Fig. 2. A (left): Simplified 1D example of an addition kernel of size 3 being applied at an offset less than the kernel size, producing different results (compare top and bottom sums of 1, 2, and 3 or 2, 3, and 4). B (right): Simplified 1D example of reflection padding (reflected through dotted line) causing a different stride pattern across a set of pixels. The altered stride prevents the tile-based processing from collaboratively building subregions of a single feature map.

This requirement means that we always need to start our tiling of the full image at the top left corner and stride across in a multiple of 16. However, this does not directly answer the question as to why we cannot have a non-multiple of 16 halo value.

Border Padding The limitation on the halo comes from the fact that if we have arbitrary halo values, we will need to use different padding schemes between the full image inference and the tile-based inference to handle the image edge effects. Figure 2 shows for a 1D case how reflection padding can (1) alter the stride across the full image which needs to be maintained as a multiple of 16 to collaboratively build subregions of a single feature map and (2) change the reflection padding required to have an input image whose spatial dimensions are

a multiple of 16. Reflection padding is preferred since it preserves image statistics locally.

ZoR and Halo Constraints Both problems, (1) collaboratively building feature maps and (2) different full image edge reflection padding requirements disappear if both the zone of responsibility and the halo are multiples of 16. Thus, we constrain the final values of ZoR and halo to be the closest higher multiple of the ratio F between the image size I and minimum feature map size (Equation 2) where $F = 16$ for the published U-Net,

$$\begin{aligned} F &= \frac{\min\{H_I, W_I\}}{\min_{\forall l_c}\{H_{l_c}, W_{l_c}\}} \\ Halo^* &= F\lceil\frac{Halo}{F}\rceil \\ ZoR &= F\lceil\frac{ZoR}{F}\rceil \end{aligned} \tag{2}$$

where H_I and W_I are the input image height and width dimensions, $Halo^*$ is the adjusted halo value to accommodate stride constraints, and H_{l_c} and W_{l_c} are the feature map height and width dimensions.

4 Experimental Results

4.1 Dataset

We used a publicly accessible dataset acquired in phase contrast imaging modality and published in [4]. The dataset consists of three collections, each with around 161 time-lapse images at roughly $20\,000 \times 20\,000$ pixels per stitched image frame with 2 bytes per pixels.

4.2 Error-Free Tile-Based Inference Scheme

Whether inferencing the whole image in a single forward pass or using tile-based processing, the input image size needs to be a multiple of 16 as previously discussed. Reflection padding is applied to the input image to enforce this size constraint before the image is decomposed into tiles.

Let us assume that we know how big an image we can fit into GPU memory, for example 1024×1024 pixels. Additionally, given that we are using U-Net we know that the required halo is 96 pixels. Then our zone of responsibility is $ZoR = 1024 - 2 \times Halo = 832$ pixels per spatial dimension. Despite inferencing 1024×1024 pixel tiles on the GPU per forward pass, the stride across the input image is 832 pixels because we need non-overlapping ZoR. The edges of the full image do not require halo context to ensure identical results when compared with a single inference pass. Intuitively, the true context is unknown since it's outside the existing image.

In the last row and column of tiles, there might not be enough pixels to fill out a full 1024×1024 tile. However, because U-Net can alter its spatial size on demand, as long as the tile is a multiple of 16 a narrower (last column) or shorter (last row) tile can be used.

4.3 Errors due to Small Halo

To experimentally confirm that our out-of-core image inference methodology does not impact the results we determined the largest image we could process on our GPU, performed the forward pass, and saved the resulting softmax output values as ground truth data. We then process the same image using our tiling scheme with varying halo values. We show that there are numerical differences (greater than floating point error) when using halo values less 96.

We trained our U-Net model to perform binary (foreground/background) segmentation of the phase contrast microscopy images. The largest image we could inference on our GPU with 24 GB of memory is 3584×3584 pixels. We created 20 reference inference results by cropping out $K = 20$ random 3584×3584 subregions of the dataset.

We performed tile-based out-of-core inference for each of the 20 reference images using a tile size of 512 pixels, meeting the multiple of 16 constraint. Halo values from 0 to 96 pixels were evaluated in 16 pixel increments.

The tiling codebase seamlessly constructs the softmax output in CPU memory as if the whole image had been inferred in a single forward pass. So our evaluation methodology consists of looking for differences in the output softmax produced by the reference forward pass (R) as well as the tile-based forward pass (T). We used the following two metrics for evaluation: Root Mean Squared Error (RMSE, Equation 3) of the softmax and Misclassification Error (ME, Equation 4) of the resulting binary segmentation masks. We also include Normalized Misclassification Error (NME, Equation 5), the ME normalized by the number of pixels; and Relative Runtime, where compute time to perform inference is shown relative by the runtime without the tiling scheme. This runtime highlights that there is a tradeoff to be made between the error introduced due to the out-of-core GPU inference and the compute overhead required to do so. The NME metric can be multiplied by 100 to compute the percent of pixels with errors due to the tiling scheme. All metrics are averaged across the $K = 20$ reference images.

$$RMSE = \frac{1}{K}\sum_{i=1}^{K}\sqrt{\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(R_{ij} - T_{ij})^2}{mn}} \tag{3}$$

$$ME = \frac{1}{K}\sum_{i=1}^{K}\left(\sum_{i=1}^{m}\sum_{j=1}^{n}[R_{ij} \neq T_{ij}]\right) \tag{4}$$

$$NME = \frac{1}{K}\sum_{i=1}^{K}\left(\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}[R_{ij} \neq T_{ij}]}{nm}\right) \tag{5}$$

The total inference error is a composite of model error (what is directly minimized by gradient descent during training) and tiling error. We demonstrate a zero error contribution from tiling in the case of trained and untrained U-Net models (or minimum and maximum inference errors due to a model).

For the trained model, the error metrics are shown in Table 1 with 512 pixel tiles. Once the required 96 pixel halo is met the RMSE falls into the range of floating point error and ME goes to zero. Beyond the minimum required halo, all error metrics remain equivalent to the minimum halo. The first row shows the data for the whole image being inferred without the tiling scheme. The ME metric is especially informative, because when it is zero, the output segmentation results are identical regardless of whether the whole image was inferred in a single pass or it was decomposed into tiles. Table 1 highlights the engineering tradeoff that must be made, obtaining zero inference error requires 2.69× the wall clock runtime. The normalized error (NME) with naive tailing is 6.1×10^{-4} or 0.06%. Depending on your application, this level of error might be acceptable despite the potential for edge effects between the non-overlapping tiles. One consideration, larger tile sizes are more compute efficient because the ratio of the ZoR area to the tile area increases.

Table 1. Error Metrics for Tile Size 512

TileSize	ZoR	Halo	RMSE	ME	NME	RelativeRuntime
3584	n/a	n/a	0.0	0.0	0.0	1.0
512	512	0	1.11×10^{-2}	7773.4	6.1×10^{-4}	1.08
512	480	16	6.35×10^{-3}	5455.4	4.2×10^{-4}	1.31
512	448	32	3.29×10^{-3}	2372.2	1.8×10^{-4}	1.36
512	416	48	1.95×10^{-3}	1193.7	9.3×10^{-5}	1.61
512	384	64	7.79×10^{-4}	434.1	3.4×10^{-5}	1.85
512	352	80	1.50×10^{-4}	71.6	5.6×10^{-6}	2.21
512	320	96	4.17×10^{-10}	0.0	0.0	2.58

For the untrained model, results are shown in Table 3 in Appendix C with 1024 pixel tiles. The results were generated using an untrained 4 class U-Net model, whose weights were left randomly initialized. Additionally, the image data for that result was normally distributed random noise with $\mu = 0, \sigma = 1$. The error coming from tile-based processing is zero once the required halo is met.

4.4 Errors due to Violation of Partitioning Constraints

To demonstrate how the inference results differ as a function of how the network strides across the input image we have constructed 32 overlapping, 2048×2048 pixel subregions of an image; each offset from the previous subregion start by 1 pixel. So the first subregion is $[x_{st}, y_{st}, x_{end}, y_{end}] = [0, 0, 2048, 2048]$, while the second subregion is $[1, 0, 2049, 2048]$, and so on. In order to compare the inference

results without any edge effects confounding the results, we only compute RMSE (Equation 3) of the softmax output within the area in common between all 32 images, inset by 96 pixels; $[128, 96, 1920, 1952]$. The results are shown in Figure 3 where identical softmax outputs only happen when the offset is a multiple of 16.

4.5 Application to a Fully Convolutional DenseNet

Up to this point we have shown that our ZoR and halo tiling scheme produces error-free out-of-core semantic segmentation inference for arbitrarily large images when using the published U-Net architecture [23]. This section demonstrates the tiling scheme on a Fully Convolutional DenseNet configured for Semantic Segmentation [14]. DenseNets [10] replace stacked convolutional layers with densely connected blocks where each convolutional layer is connected to all previous convolutional layers in that block. Jegou et al. [14] extended this original DenseNet idea to create a fully convolutional DenseNet based semantic segmentation architecture.

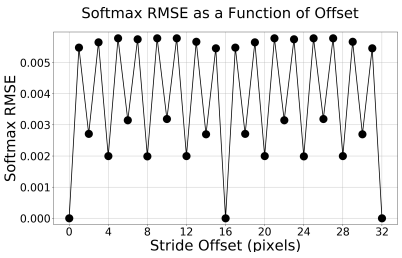

Fig. 3. Impact of the stride offset on the RMSE of the U-Net softmax output.

While the architecture of DenseNets significantly differs from U-Net, the model is still fully convolutional and thus our tiling scheme is applicable. Following Equation 1 for a FC-DenseNet-56 [14] model produces a required halo value of 377. This is significantly higher than U-Net due to the architecture depth. FC-DenseNet-56 also has a ratio between the input image size and the smallest feature map of $F = 32$. Therefore the inference image sizes need to be a multiple of 32, not 16 like the original U-Net. Thus, the computed 377 pixel halo is adjusted up to 384.

The error metrics for FC-DenseNet-56 as a function of halo are showing in Table 4 in Appendix C. This numerical analysis relies on versions of the same 20 test images from the U-Net analysis, but cropped to 2304×2304, which was the largest image we were able to inference using FC-DenseNet-56 on our 24 GB GPU in a single forward pass.

4.6 Halo Approximation via Effective Receptive Field

The halo value required to achieve error free inference increases with the depth of the network. For example, see Table 2 which shows the theoretical halo values (computed using Equation 1) for a few common semantic segmentation architectures. The deeper networks, like FC-DenseNet-103 require very large halo values to guarantee error free tile based inference.

Table 2. Theoretical Radii for Common Segmentation Architectures

Architecture	Halo
U-Net [23]	96
SegNet [2]	192
FCN-VGG16 [18]	202
FC-DenseNet-56 [14]	384
FC-DenseNet-67 [14]	480
FC-DenseNet-103 [14]	1120

Using the empirical effective receptive field estimation method outlined by Luo et al. [19] which consists of setting the loss to 1 in the center of the image and then back propagating to the input image, we can automatically estimate the required halo. For our trained U-Net [23] this method produces an estimated halo of 96 pixels, which is exactly the theoretical halo. This matches the data in Tables 1 and 3 where the ME metric did not fall to zero until the theoretical halo was reached. On the other hand, according to Table 4 for FC-DenseNet-56, there is no benefit to using a halo larger than 192 despite the theoretical required halo (receptive field) being much larger. This is supported by the effective receptive field estimated halo of 160 pixels; just below the empirically discovered minimum halo of 192. Thus, using the effective receptive field for estimating the required halo is not foolproof but it provides a good proxy for automatically reducing the tiliing error.

5 Conclusions

This paper outlined a methodology for performing error-free segmentation inference of arbitrarily large images. We documented the formulas for determining the tile-based inference scheme parameters. We then demonstrated that the inference results are identical regardless of whether or not tiling was used. These inference scheme parameters were related back to the theoretical and effective receptive fields of deep convolutional networks as previously studied in literature [19]. The empirical effective receptive field estimation methods of Luo et al. [19] were used to provide a rough estimate of the inference tiling scheme parameters without requiring any knowledge of the architecture. While we used U-Net and FC-DenseNets as example FCN models, these principles apply to any FCN model while being robust across different choices of tile size.

In this work we did not consider any FCN networks with dilated convolutions, which are known to increase the receptive field side of the network. We will include this extension in future work as well as tradeoff evaluations of the relationships among relative runtime, GPU memory size and maximum tile size.

6 Test Data and Source Code

The test data and the Tensorflow v2.x source code are available from public URLs[4]. While the available codebase in theory supports arbitrarily large images, we made the choice at implementation time to load the whole image into memory before processing it through the network. In practice this means the codebase is limited to inferencing images which fit into CPU memory. However, using a file format which supports reading sub-sections of the whole image would support inference of disk-backed images which do not fit into CPU memory.

7 Disclaimer

Commercial products are identified in this document in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the products identified are necessarily the best available for the purpose. Analysis performed [in part] on the NIST Enki HPC cluster. Contribution of U.S. government not subject to copyright.

[4] https://isg.nist.gov/deepzoomweb/data/stemcellpluripotency
https://github.com/usnistgov/semantic-segmentation-unet/tree/ooc-inference.

References

1. Araujo, A., Norris, W., Sim, J.: Computing receptive fields of convolutional neural networks. Distill (2019). https://doi.org/10.23915/distill.00021, https://distill.pub/2019/computing-receptive-fields
2. Badrinarayanan, V., Kendall, A., Cipolla, R.: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. In: IEEE transactions on pattern analysis and machine intelligence (2017). https://doi.org/10.1109/TPAMI.2016.2644615, http://arxiv.org/abs/1511.00561
3. Bardakoff, A.: Fast image (fi) : A high-performance accessor for processing gigapixel images (Aug 2019), https://github.com/usnistgov/FastImage
4. Bhadriraju, K., Halter, M., Amelot, J., Bajcsy, P., Chalfoun, J., Vandecreme, A., Mallon, B.S., Park, K.y., Sista, S., Elliott, J.T., Plant, A.L.: Large-scale time-lapse microscopy of Oct4 expression in human embryonic stem cell colonies. Stem Cell Research **17**(1), 122–129 (2016). https://doi.org/10.1016/j.scr.2016.05.012, http://linkinghub.elsevier.com/retrieve/pii/S1873506116300502
5. Blattner, T., Keyrouz, W., Bhattacharyya, S.S., Halem, M., Brady, M.: A Hybrid Task Graph Scheduler for High Performance Image Processing Workflows. Journal of Signal Processing Systems **89**(3), 457–467 (2017). https://doi.org/10.1007/s11265-017-1262-6
6. Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y.: Deformable Convolutional Networks. Proceedings of the IEEE International Conference on Computer Vision **2017-Octob**, 764–773 (2017). https://doi.org/10.1109/ICCV.2017.89
7. Dumoulin, V., Visin, F.: A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285 (2016)
8. Hu, J., Shen, L., Sun, G.: Squeeze-and-Excitation Networks. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition pp. 7132–7141 (2018). https://doi.org/10.1109/CVPR.2018.00745
9. Huang, B., Reichman, D., Collins, L.M., Bradbury, K., Malof, J.M.: Tiling and stitching segmentation output for remote sensing: Basic challenges and recommendations. arXiv preprint arXiv:1805.12219 (2019)
10. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. vol. 2017-Janua, pp. 2261–2269 (aug 2017). https://doi.org/10.1109/CVPR.2017.243, http://arxiv.org/abs/1608.06993
11. Iglovikov, V., Mushinskiy, S., Osin, V.: Satellite imagery feature detection using deep convolutional neural network: A kaggle competition. arXiv preprint arXiv:1706.06169 (2017)
12. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
13. Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K.: Spatial transformer networks. Advances in Neural Information Processing Systems **2015-January**, 2017–2025 (2015)
14. Jegou, S., Drozdzal, M., Vazquez, D., Romero, A., Bengio, Y.: The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops **2017-July**, 1175–1183 (2017). https://doi.org/10.1109/CVPRW.2017.156
15. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436 (2015)

16. Lin, H., Chen, H., Graham, S., Dou, Q., Rajpoot, N., Heng, P.A.: Fast ScanNet: Fast and Dense Analysis of Multi-Gigapixel Whole-Slide Images for Cancer Metastasis Detection. In: IEEE Transactions on Medical Imaging. vol. 38, pp. 1948–1958 (2019). https://doi.org/10.1109/tmi.2019.2891305
17. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature Pyramid Networks for Object Detection. In: Computer Vision and Pattern Recognition (2017)
18. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2015). https://doi.org/10.1109/CVPR.2015.7298965
19. Luo, W., Li, Y., Urtasun, R., Zemel, R.: Understanding the effective receptive field in deep convolutional neural networks. Advances in Neural Information Processing Systems (Nips), 4905–4913 (2016)
20. Maggiori, E., Tarabalka, Y., Charpiat, G., Alliez, P.: Fully convolutional neural networks for remote sensing image classification. In: International Geoscience and Remote Sensing Symposium (IGARSS). IEEE (2016). https://doi.org/10.1109/IGARSS.2016.7730322
21. Mnih, V.: Machine Learning for Aerial Image Labeling. PhD Thesis p. 109 (2013)
22. Ramachandran, P., Parmar, N., Vaswani, A., Bello, I., Levskaya, A., Shlens, J.: Stand-Alone Self-Attention in Vision Models. In: Neural Information Processing Systems (2019), http://arxiv.org/abs/1906.05909
23. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241 (may 2015)
24. 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**, 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y, http://dx.doi.org/10.1007/s11263-015-0816-y
25. Saito, S., Yamashita, T., Aoki, Y.: Multiple object extraction from aerial imagery with convolutional neural networks. IS and T International Symposium on Electronic Imaging Science and Technology **60**(1), 1–9 (2016). https://doi.org/10.2352/ISSN.2470-1173.2016.10.ROBVIS-392
26. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229 (2013)
27. Sherrah, J.: Fully convolutional networks for dense semantic labelling of high-resolution aerial imagery. arXiv preprint arXiv:1606.02585 (2016)
28. Van Etten, A.: Satellite imagery multiscale rapid detection with windowed networks. In: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 735–743. IEEE (2019). https://doi.org/10.1109/WACV.2019.00083
29. Volpi, M., Tuia, D.: Dense semantic labeling of subdecimeter resolution images with convolutional neural networks. In: IEEE Transactions on Geoscience and Remote Sensing. vol. 55, pp. 881–893. IEEE (2016). https://doi.org/10.1109/TGRS.2016.2616585
30. Wang, X., Girshick, R., Gupta, A., He, K.: Non-local Neural Networks. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2018). https://doi.org/10.1109/CVPR.2018.00813

8 Appendix A: Derivation of Simplified Halo Formula

Let us assume that in the entire U-Net architecture the kernel size is constant $k_c = k = const$ and each level has the same number of convolutional layers on both decoder and encoder sides $n_l = n = const$. If these constraints are satisfied, then the general formula for determining halo can be simplified as follows:

$$\begin{aligned} Halo &= \sum_{c=1}^{N} 2^{l_c} \lfloor \frac{k_c}{2} \rfloor \\ &= \lfloor \frac{k}{2} \rfloor \times \sum_{c=1}^{N} 2^{l_c} \\ &= \lfloor \frac{k}{2} \rfloor \times (2 \times \sum_{m=0}^{M-1} (2^m \times n) + 2^M \times n) \\ &= \lfloor \frac{k}{2} \rfloor \times n \times (2 \times \frac{1 \times (1 - 2^M)}{1 - 2}) + 2^M) \\ &= \lfloor \frac{k}{2} \rfloor \times n \times (3 \times 2^M - 2) \end{aligned} \tag{6}$$

where M is the maximum U-Net level $M = \max_{\forall c}\{l_c\}$.

For U-Net architecture, the parameters are $k = 3$, $n = 2$ and $M = 4$, and the equation yields $Halo = 92$.

For DenseNet architecture, the parameters are $k = 3$, $n = 4$ and $M = 5$, and the equation yields $Halo = 376$. This value differs by one from the value computed according to Equation 1 because the DenseNet has asymmetry between the first encoder layer with a kernel size $k = 3$ and the last decoder layer with a kernel size $k = 1$.

9 Appendix B: Example U-Net Halo Calculation

Following Equation 1 for U-Net results in a required halo of 92 pixels in order to provide the network with all of the local context it needs to predict the outputs correctly. With $k = 3$, $\lfloor \frac{k_c}{2} \rfloor$ reduces to $\lfloor \frac{3}{2} \rfloor = 1$. The halo computation for U-Net thus reduces to a sum of 2^{l_c} terms for each convolutional layer encountered along the longest path from input to output as shown in Equation 7.

$$Halo = \sum_{c=0}^{17} 2^{l_c} \tag{7}$$

By substituting the level numbers for each convolutional layer from 0 to 17 as shown in Equation 8, one obtains the minimum halo value of 92 pixels.

$$\begin{aligned} l_c &= \{0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 3, 3, 2, 2, 1, 1, 0, 0\} \\ 92 &= 2^0 + 2^0 + 2^1 + 2^1 + 2^2 + 2^2 + 2^3 + ... \end{aligned} \tag{8}$$

Similarly, according to Equation 6, the calculation simplifies to:

$$\begin{aligned} M &= \max_{\forall c} l_c = 4 \\ k_c &= k = 1 \\ n_l &= n = 2 \\ 92 &= 1 \times 2 \times (3 \times 2^4 - 2) \end{aligned} \tag{9}$$

10 Appendix C: Error Metrics

Table 3. UnTrained U-Net Error Metrics for Tile Size 1024

TileSize	ZoR	Halo	RMSE	ME	NME	RelativeRuntime
3584	n/a	n/a	0.0	0.0	0.0	1.0
1024	1024	0	2.36×10^{-4}	21856.9	1.7×10^{-3}	1.08
1024	992	16	1.05×10^{-6}	234.8	1.8×10^{-5}	1.23
1024	960	32	1.92×10^{-7}	49.7	3.9×10^{-6}	1.30
1024	928	48	5.47×10^{-8}	13.2	1.0×10^{-6}	1.35
1024	896	64	1.44×10^{-8}	2.8	2.1×10^{-7}	1.31
1024	864	80	5.87×10^{-9}	1.4	1.1×10^{-7}	1.5
1024	832	96	3.54×10^{-10}	0.0	0.0	1.58

Table 4. Error Metrics for FC-DenseNet Tile Size 1152

TileSize	ZoR	Halo	RMSE	ME	NME	RelativeRuntime
2304	n/a	n/a	0.0	0.0	0.0	1.0
1152	1152	0	5.40×10^{-3}	821.5	1.5×10^{-4}	1.15
1152	1088	32	1.81×10^{-3}	376.1	7.1×10^{-5}	1.42
1152	1024	64	9.16×10^{-4}	168.0	3.2×10^{-5}	1.54
1152	960	96	3.36×10^{-4}	51.6	9.7×10^{-6}	1.59
1152	896	128	5.63×10^{-5}	5.3	1.0×10^{-6}	1.67
1152	832	160	4.97×10^{-6}	0.2	4.7×10^{-8}	1.76
1152	768	192	2.44×10^{-7}	0.0	0.0	2.32
1152	704	224	1.92×10^{-8}	0.0	0.0	2.22
1152	640	256	1.37×10^{-8}	0.0	0.0	2.33
1152	576	288	1.44×10^{-8}	0.0	0.0	2.39
1152	512	320	1.37×10^{-8}	0.0	0.0	2.52
1152	448	352	1.45×10^{-8}	0.0	0.0	4.65
1152	384	384	1.37×10^{-8}	0.0	0.0	5.89

11 Appendix D: `SAME` vs `VALID` Convolution Padding

The original U-Net paper uses `VALID` type convolutions which shrink the spatial size of the feature maps by 2 pixels for each layer [7][5]. Figure 2.1 from Dumoulin and Visin "A guide to convolutional arithmetic for deep learning" [7] shows an illustration showing why `VALID` causes the feature maps to shrink. The effect of `VALID` convolutions can also be seen in the first layer of the original U-Net where the input image size of 572×572

[5] For an excellent review of convolutional arithmetic, including transposed convolutions (i.e., up-conv), see "A guide to convolutional arithmetic for deep learning by Dumoulin and Visin" [7].

pixels shrinks to 570×570 [23]. Switching to `SAME` type convolutions requires that within each convolutional layer zero padding is applied to each feature map to ensure the output has the same spatial size as the input. Figure 2.3 from Dumoulin and Visin [7] shows an illustration where a input 5×5 remains 5×5 after the convolution is applied. While `VALID` type convolutions avoid the negative effects of the zero padding within `SAME` type convolutions, which can affect the results as outlined by Huang et al. [9], it is conceptually simpler to have input and output images of the same size. Additionally, our tiling scheme overcomes all negative effects that zero padding can introduce, justifying the choice of `SAME` type convolutions.

The change to `SAME` type convolutions introduces an additional constraint on U-Net that needs to be mentioned. Given the skip connections between the encoder and decoder elements for matching feature maps, we need to ensure that the tensors being concatenated together are the same size. This can be restated as requiring the input size be divisible by the largest stride across the input image by any kernel. Another approach to ensuring consistent size between the encoder and decoder is to pad each up-conv layer to make its output larger and then crop to the target feature map size. We feel that is a less elegant solution than enforcing a tile size constraint and potentially padding the input image.

The feature map at the bottleneck of U-Net is spatially $16\times$ smaller than the input image. Therefore, the input to U-Net needs to be divisible by 16. As we go deeper into a network we trade spatial resolution for feature depth. Given a 512×512 pixel input image, the bottleneck shape will be $N \times 1024 \times 32 \times 32$ (assuming `NCHW`[6] dimension ordering with unknown batch size). Thus the input image height divided by the bottleneck feature map height is $\frac{512}{32} = 16$. However, if the input image is 500×500 pixels, the bottleneck would be (in theory) $N \times 1024 \times 31.25 \times 31.25$. When there are not enough input pixels in a feature map to perform the 2×2 max pooling, the output feature map size is the floor of the input size divided by 2. Thus, for an input image of 500×500 pixels the feature map heights after each max pooling layer in the encoder are: $[500, 250, 125, 62, 31]$. Now following the up-conv layers through the decoder, each of which doubles the spatial resolution, we end up with the following feature map heights: $[31, 62, 124, 248, 496]$. This results in a different feature map spatial size at the third level; encoder 125, decoder 124. If the input image size is a multiple of 16 this mismatch cannot happen.

[6] NCHW Tensor dimension ordering: N (batch size), Channels, Height, Width