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

Michael Majurski and Peter Bajcsy
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 inference of arbitrarily large images for semantic segmentation using fully convolutional neural networks (FCNN). FCNN 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 memory (RAM) of the GPU executing the inference. This work is motivated by overcoming the GPU memory size constraint via a tile-based inferencing methodology which does not numerically impact the final result.

We developed mathematical formulas for determining the tile size and stride of tiles created from an input image too large to inference on a GPU. The formulas are validated on multiple configurations of an FCNN U-Net model and tiling parameters. The numerical accuracy is evaluated by executing the forward (inference) pass of the U-Net network with each parameter setting on $20\,000 \times 20\,000$ pixel grayscale images.

This method decomposes the full inference image into small, overlapping tiles each of which fit into GPU memory for the forward (inference) pass of the network. This tiling scheme produces a segmented result as if the whole image had been inferenced in a single pass. The primary contribution of this work lies in demonstrating that one can achieve error-free inference using a tile-based (out-of-core) approach if the tiling parameters are chosen according to the mathematical analysis of the FCNN model and GPU specification. In addition, we document the segmentation errors due to tiling configurations that do not satisfy the formulas.

Introduction

The task of semantic segmentation, assigning a label to each image pixel, is often performed using deep learning based convolutional neural networks (CNNs) (Badrinarayanan, Kendall, and Cipolla 2017; Ronneberger, Fischer, and Brox 2015), for instance, by using a special type of CNN which only uses convolutional layers. These so-called ”fully convolutional neural networks” (FCNN) have a very useful property which allows altering the input image size. Both U-Net (Ronneberger, Fischer, and Brox 2015) and the original FCN network (Long, Shelhamer, and Darrell 2015) are examples of FCNN type CNNs. FCNNs enable training the network on images much smaller than those of interest at inference time. For example, one can train a U-Net model on 512×512 pixel tiles and then perform inference on many $20\,000 \times 20\,000$ pixel images. 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 GPU cards.

The ability of FCNN networks to inference of 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 images can be resized to fit the network. For example, if one trained a CNN on ImageNet (Russakovsky et al. 2015) 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.

In another example, performing semantic segmentation of the scene acquired by a self driving car's front camera, one can infer which surfaces are drivable at the native camera resolution of 720p (1280×720 pixels - $1\,\mathrm{Mpixel}$). Alternatively, one can inference images after down-samplings if compute time must be decreased in such a time-critical application.

In contrast to the above two examples, 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 ($100\,000 \times 50\,000$ pixels) and fit it into GPU memory; nor can one reasonably downsample the image to fit 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. The original U-Net paper (Ronneberger, Fischer, and Brox 2015) 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 large image tiling which enables error-free inference on GPUs with limited memory. This work draws on FastImage, a high-performance accessor library for processing gigapixel images in a tile-based manner (Bardakoff 2019).

Related Work

It has been known since the initial introduction of fully convolutional neural networks that they can be applied via shift-and-stitch methods as if the FCNN were a single filter (Long, Shelhamer, and Darrell 2015; Sherrah 2016). The original U-Net paper by Ronneberger et al. (Ronneberger, Fischer, and Brox 2015) 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 inference. Sliding window (i.e. overlapping tiles) has been used for object detection (Sermanet et al. 2013; Van Etten 2019) as well as for semantic segmentation (Lin et al. 2019; Volpi and Tuia 2016) inference. Patch-based inference also supports arbitrarily large images, but it is very inefficient (Volpi and Tuia 2016; Maggiori et al. 2016).

Huang et al. directly examine the problem of operating on images which cannot be inferenced in a single forward pass. However, the authors focus on different methods for reducing the error in labeling that arises from different overlapping tile-based processing schemes (Huang et al. 2019). 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 (Huang et al. 2019). Huang et al. also examines the effects of zero-padding, documenting how much error it introduces (Huang et al. 2019). At no point do they produce error-free tile-based inference. Iglovikov et al. also remark upon error in the logits near the edge of tiles during inference and suggest overlapping predictions or cropping the output to reduce that error (Iglovikov, Mushinskiy, and Osin 2017).

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.

Methods

To explain out-of-core image inference we use U-Net (Ronneberger, Fischer, and Brox 2015) as the case study FCNN. The structure of U-Net can be seen in Figure 4. Nonetheless, the presented methodology applies to any FCNN network, just the specific numerical values will be different.

U-Net Configuration

Before delving into the out-of-core inference methodology details, we clarify two modifications of U-Net.

1. Normalization: Batch normalization (Ioffe and Szegedy 2015) 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 (Ronneberger, Fischer, and Brox 2015).

The original U-Net paper uses VALID type convolutions which shrink the spatial size of the feature maps by 2 pixels for each layer. Figure 1 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 pixels shrinks to 570×570 (Ronneberger, Fischer, and Brox 2015). At the original U-Net output, the feature maps have shrunk down to 388×388 pixels.

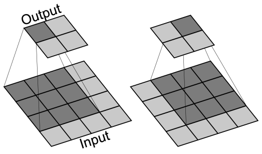

Figure 1: Illustration of VALID type convolution where no padding is applied to the feature maps, resulting in the feature maps shrinking from 4×4 to a 2×2. In this case, all convolution input values must be present to generate valid outputs.

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 shows an illustration where an input 4×4 remains 4×4 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. (Huang et al. 2019), users prefer 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. For an excellent review of convolutional arithmetic, including transposed convolutions (i.e., up-conv), see ”A guide to convolutional arithmetic for deep learning” (Dumoulin and Visin 2016).

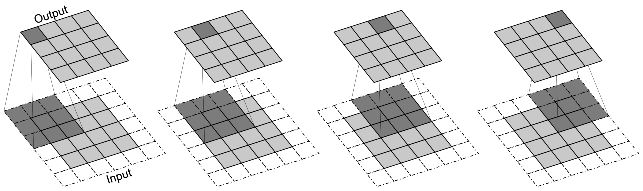

Figure 2: Illustration of SAME type convolutions where zero padding (empty white squares) is applied to ensure the output feature map has the same spatial dimensions as the input.

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. The feature map at the bottleneck of U-Net is spatially 16× smaller than the input image. 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` [1] 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 (fractionally strided/transposed convolution (Dumoulin and Visin 2016)) 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.

To ensure the input image is always a multiple of 16, we pad the end of each spatial dimension via reflection to meet the size requirement. So a 500×500 pixel image will be padded on the right and bottom with 12 pixels to bring its size to 512×512 pixels. Conversely, a 512×512 pixel image will be inferenced unmodified. Reflection padding is used to ensure that the summary statistics of the whole image are not unduly skewed since z-score normalization is used during inference.

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 double the feature map spatial resolution (Dumoulin and Visin 2016).

Conceptual Framework

Given an FCNN model architecture and a GPU with enough memory to inference at least a 512×512 pixel image, we can construct a scheme for inferencing arbitrarily sized images. There are two important concepts required for this tile-based (out-of-core) processing scheme.

1. Zone of Responsibility (ZoR): a rectangular region (partition, zone, or area) of the output image currently being computed.
2. Radius: minimum horizontal and vertical border size around the ZoR indicating the local context that the FCNN requires to accurately compute all pixels within the ZoR.

Each dimension of a square tile is then defined as $TileSize = ZoR + 2 \times Radius$. Figure 3 shows an example where a 832×832 pixel zone of responsibility is shown as a red square with a 96 pixel radius surrounding it. Since the pixels within the ZoR and the radius need to be passed through the network to compute the output, one tile of GPU input is $832 + 2 \times 96 = 1024$ pixels per spatial dimension.

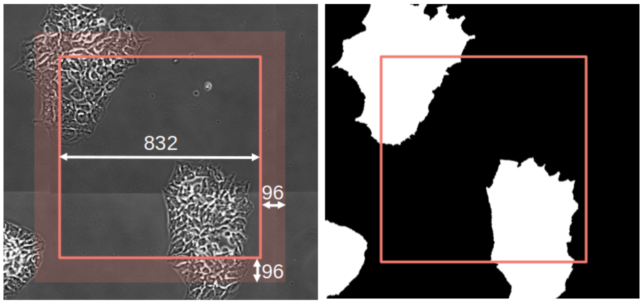

Figure 3: Left: ZoR (832×832 pixel square) with a 96 pixel surrounding radius (shaded area) for tile-based inference of stem cell colonies. Right: segmentation output showing the ZoR contribution.

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 inferenced is broken down into non-overlapping zones of responsibility. For each ZoR, the local context defined by the radius (where available) is included and passed through the network. The ZoR within the inferenced tile result (without the radius) is copied to the output image being constructed in CPU memory. The radius provides the network with all the information it needs to make correct, deterministic predictions for the entirety of the zone of responsibility. Hence the name, since each ZoR is responsible for a specific zone of the output. Therefore, while the full image was broken into tiles for inference, each pixel had all of the local context required to be predicted as if the whole image were passed through the network as one block of memory.

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 inferencing the whole image in one pass. In summary, each tile's feature maps are created (inferenced), its ZoR output extracted, and then the GPU memory is recycled for the next tile. By building each ZoR in a separate forward pass we can construct the network output within a fixed GPU memory footprint for arbitrarily large images.

Determining The Radius

Let U-Net be described by an ordered sequence of convolutional layers $c = 1, ..., N$ with each layer being associated with a level l_c and a square kernel $k_c \times k_c$. A convolutional layer (1) convolves a kernel and its input feature maps to create a set of output feature maps, (2) applies an element-wise a non-linearity (ReLu (LeCun, Bengio, and Hinton 2015)), and (3) performs batch normalization (Ioffe and Szegedy 2015) on its output feature maps. For the network, N defines the number of convolutional layers along the longest path from input to output. The index of c for each convolutional layer is written on each blue arrow in Figure 4.

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

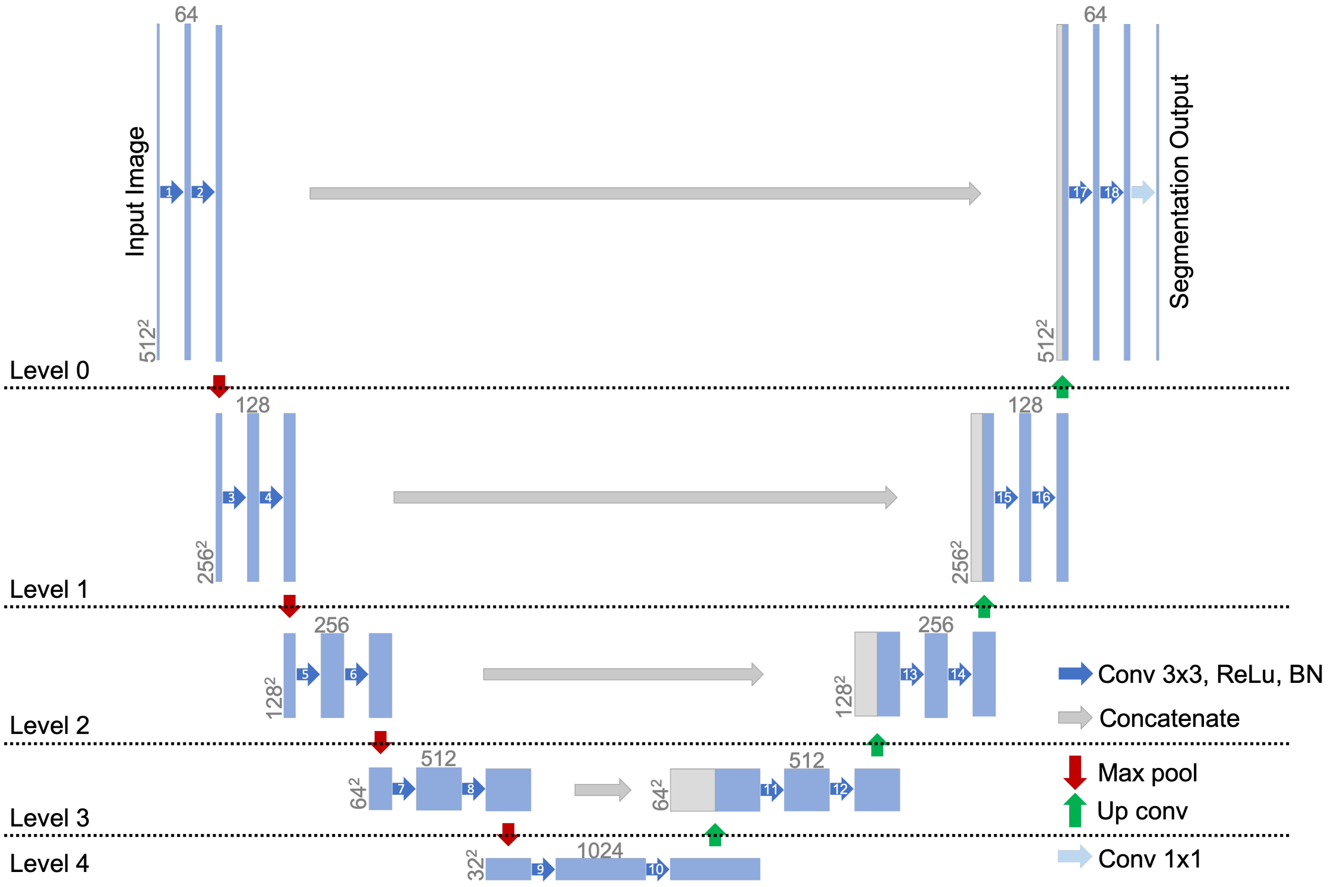

Figure 4: U-Net model architecture showing the different convolutional layers (blue arrows) and their respective levels. Each blue box is a multi-channel feature map with the channel count denoted on top of the box and the spatial dimension at the lower left edge. White boxes represent copied and concatenated feature maps. Each convolutional layer is numbered sequentially along the longest path through the network from input image to output segmentation result.

Let us define the level l_c of an encoder-decoder network architecture as the number of max-pool 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 Radius Calculation The minimum required radius can be calculated according to the Equation 1 for a general FCNN architecture.

$$Radius = \sum_{c=1}^{N} 2^{l_c} \lfloor \frac{k_c}{2} \rfloor \qquad (1)$$

The radius is a sum over every convolutional layer index c from 1 to N 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, if a 3×3 convolution is applied at level $l_c = 4$ then $2^4 = 16$ pixels of context are needed 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.

Radius Calculation for U-Net Let us assume that the U-Net architecture satisfies the following design constraints: $k_c = k = const$ and each level has the same number of convolutional layers on both decoder and encoder sides $n_l = n = const$. These constraints are satisfied for the published U-Net where $k = 3$ and $n_l = 2$.

In this case, the minimum required radius can be calculated according to the Equation 2.

$$Radius = \lfloor \frac{k}{2} \rfloor \times n \times (3 \times 2^M - 2) \qquad (2)$$

Where M is the maximum level value l_c over all values of c (i.e., $M = max_{\forall c}(l_c)$). The radius is linearly proportional to the kernel size k and to the number of convolutional layers per level n and exponentially proportional to the maximum level M. The derivation of Equation 2 from Equation 1 is provided in Appendix A.

The published U-Net (Figure 4) has one level per horizontal stripe of layers. The input image enters on level $l_{c=1} = l_{c=2} = 0$. The first max-pool layer halves the spatial resolution of the network, changing the level. Convolution layers $c = \{3, 4\}$ after that first max-pool layer up to the next max-pool layer belong to level $l_{c=3} = l_{c=4} = 1$. This continues through level 4, where the bottleneck of the U-Net model occurs. In Figure 4 the bottleneck is the feature map at the bottom with dimensions: $N \times 1024 \times 32 \times 32$ (`NCHW` dimension ordering), this occurs right before the first up-conv layer. After the bottleneck, the level number decreases with each subsequent up-conv layer, until level $l_N = 0$ right before the output image is generated. This is summarized in Figure 4 where the levels each convolutional layer are indicated.

Following Equation 1 or 2 for U-Net results in a minimum required radius of 92 pixels in order to provide the network with all of the local context it needs to predict the outputs correctly. See Appendix B for details on applying Equation 1 to U-Net. This radius 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 (Ronneberger, Fischer, and Brox 2015). 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 radius of 96 pixels. Unfortunately, one cannot just adjust the ZoR size to ensure $(ZoR + Radius)\%16 = 0$ because of how convolutional arithmetic works.

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 5 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.

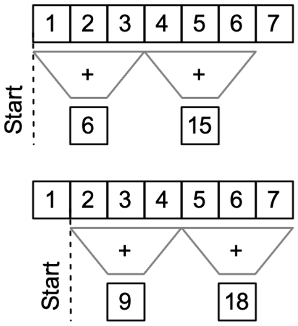

Figure 5: Simplified 1D example of an addition kernel of size 3 being applied at an offset less than the kernel size, producing different results.

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 radius value.

Border Padding The limitation on the radius comes from the fact that if we have arbitrary radius 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 6 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.

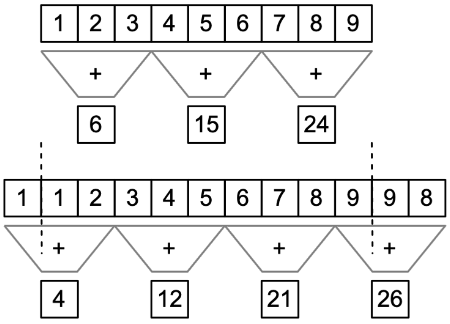

Figure 6: Simplified 1D example of reflection padding (reflected through dotted line) causing a different stride pattern across a set of pixels. The altered stride prevents the tile-based processing from collaboratively building subregions of a single feature map.

ZoR and Radius 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 radius are multiples of 16. Thus, we constrain the final values of ZoR and radius to be the closest higher multiple of the ratio F between the image size I and minimum feature map size (Equation 3) where $F = 16$ for the published U-Net.

$$\begin{aligned} F &= \frac{\min\{H_I, W_I\}}{\min_{\forall l_c}\{H_{l_c}, W_{l_c}\}} \\ Radius &= F\lceil\frac{Radius}{F}\rceil \\ ZoR &= F\lceil\frac{ZoR}{F}\rceil \end{aligned} \tag{3}$$

Where H_I and W_I are the input image height and width dimensions, and H_{l_c} and W_{l_c} are the feature map height and width dimensions.

Experimental Results

Dataset

We used a publicly accessible dataset acquired in phase contrast imaging modality and published in (Bhadriraju et al. 2016). 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.

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 radius is 96 pixels. Then our zone of responsibility is $ZoR = 1024 - 2 \times Radius = 832$ 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. Figure 7 shows a full image with 6 non-overlapping ZoR in alternating colors (red and blue) with each radius as a shaded region surrounding each ZoR. The edges of the full image do not require radius context to ensure identical results when compared with a single inference pass. Intuitively, the the true context is unknown since its outside the existing image.

Image tiling starts at $[x_{st}, y_{st}, x_{end}, y_{end}] = [0, 0, 832, 832]$ and proceeds across the image with stride 832. 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.

Errors due to Small Radius

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

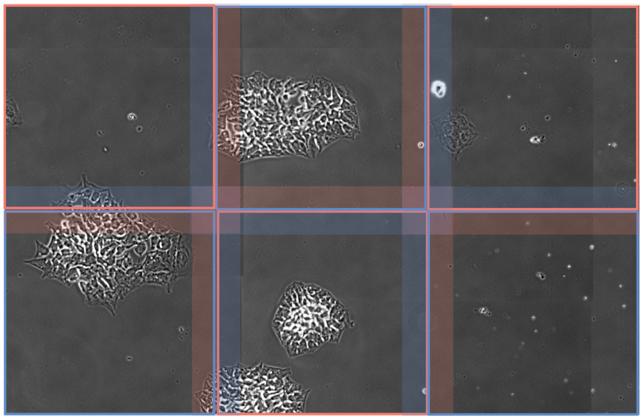

Figure 7: Non-overlapping ZoR with radius shown in the interior of a large image being inferenced. Each ZoR (denoted with a line) has a shaded radius region outside the square.

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 3824×3824 pixels. We created 20 reference inference results by cropping out $K = 20$ random 3824×3824 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. Radius values from 0 to 128 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 inferenced as 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 4) of the softmax and Misclassification Error (ME Equation 5) of the resulting binary segmentation masks. ME is the number of misclassified pixels, which provides a very intuitive understanding of the error since it directly measures how many output pixels were incorrect. Both metrics are averaged across the $K = 20$ reference images.

$$RMSE = \frac{1}{K}\sum_{i=1}^{K}\sqrt{\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(R_{ij} - T_{ij})^2}{mn}} \tag{4}$$

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

The error metrics are shown in Table 1 for 512 pixel tiles.

Once the required 96 pixel radius is met the RMSE falls into the range of floating point error and ME goes to zero. Beyond the minimum required radius, all error metrics remain equivalent to the minimum radius. The ME metric is especially informative, because when it is zero, the output segmentation results are identical regardless of whether the whole image was inferenced in a single pass or it was decomposed into tiles.

Table 1: Error Metrics for Tile Size 512

TileSize	ZoR	Radius	RMSE	ME
512	512	0	1.30e-2	6052.6
512	480	16	5.85e-3	1555.0
512	448	32	2.89e-3	432.1
512	416	48	9.32e-4	77.4
512	384	64	1.75e-4	10.8
512	352	80	1.17e-5	0.4
512	320	96	0.0	0.0
512	288	112	2.61e-8	0.0
512	256	128	0.0	0.0

These error metrics only evaluate the error coming from the tiling scheme. There is no evaluation of how accurately U-Net itself is performing. To demonstrate this, results for 1024 pixel tiles (Table 2) 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 $\mu = 0, \sigma = 1$. No doubt the segmentation results from that U-Net were nonsense, but the error coming from tile-based processing is 0 once the required radius is met.

Table 2: UnTrained U-Net Error Metrics for Tile Size 1024

TileSize	ZoR	Radius	RMSE	ME
1024	1024	0	2.70e-4	55796.3
1024	992	16	1.32e-6	532.4
1024	960	32	2.49e-7	113.8
1024	928	48	7.71e-8	43.5
1024	896	64	2.00e-8	8.5
1024	864	80	8.71e-9	3.7
1024	832	96	0.0	0.0

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 4) 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 8 where identical softmax outputs only happen when the offset is a multiple of 16.

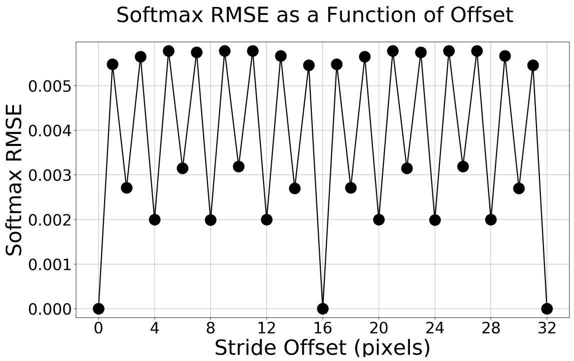

Figure 8: Impact of the stride offset on the RMSE of the U-Net softmax output.

Application to a Modified U-Net Architecture

Up to this point we have shown that our ZoR and Radius tiling scheme produces error-free out-of-core semantic segmentation inference for arbitrarily large images when using the published U-Net model architecture. This section demonstrates the tiling scheme on a modified U-Net as a proxy for any modification to a FCNN that someone might want to make. First, the number of convolutional layers between each spatial resolution changing layer is increased to $n_l = 3$. Second the lowest level $l = 4$ is removed, changing $M = 3$. The modified U-Net is shown in Figure 9. Following Equation 1 or 2 for this modified U-Net model produces a required radius value of 66.

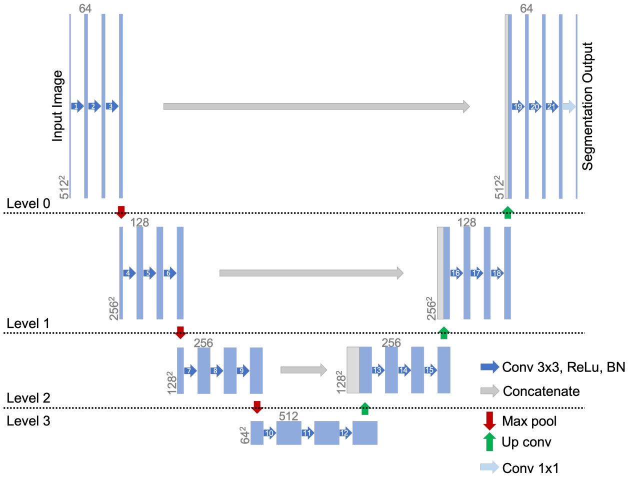

Figure 9: Modified U-Net model architecture with $n_l = 3$ and $M = 3$.

This modified U-Net has one additional difference from the published U-Net. With $M = 3$ the smallest feature map is $[N \times 512 \times 64 \times 64]$ which according to Equation 3 means the size ratio between the input image and the smallest feature map is $F = 8$. Therefore the inference image sizes need to be a multiple of 8, not 16 like the original U-Net. Thus, the computed 66 pixel radius is adjusted to 72.

Table 3 shows the error metrics for a tile size of 1032 pixels (multiple of 8) over a range of radius values from 0 to 72 with a step of 8.

Table 3: Modified U-Net Error Metrics for Tile Size 1032

TileSize	ZoR	Radius	RMSE	ME
1032	1032	0	7.51e-3	2688.4
1032	1016	8	5.18e-3	1678.0
1032	1000	16	3.81e-3	947.5
1032	984	24	2.33e-3	545.8
1032	968	32	1.31e-3	203.1
1032	952	40	4.16e-4	61.6
1032	936	48	6.50e-5	7.9
1032	920	56	1.81e-6	0.1
1032	904	64	3.70e-8	0.0
1032	888	72	1.20e-8	0.0

Conclusions

This paper outlines a methodology for performing error-free out-of-core semantic segmentation inference of arbitrarily large images. We provide formulas for determining the tile-based inference scheme parameters and demonstrated the inference results are identical whether or not tiling was used. While we used U-Net (and its modifications) as an example FCNN model for this work, the same principles apply to any FCNN model while being robust across different choices of tile size.

Test Data and Source Code

The test data used in this paper are available at https://isg.nist.gov/deepzoomweb/data/stemcellpluripotency.

The U-Net Tensorflow v2.0 source code used in this paper is available at https://github.com/usnistgov/semantic-segmentation-unet/tree/ooc-inference.

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.

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.

References

Badrinarayanan, V.; Kendall, A.; and Cipolla, R. 2017. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. In *IEEE transactions on pattern analysis and machine intelligence*.

Bardakoff, A. 2019. Fast image (fi) : A high-performance accessor for processing gigapixel images.

Bhadriraju, K.; Halter, M.; Amelot, J.; Bajcsy, P.; Chalfoun, J.; Vandecreme, A.; Mallon, B. S.; Park, K.-y.; Sista, S.; Elliott, J. T.; and Plant, A. L. 2016. Large-scale time-lapse microscopy of Oct4 expression in human embryonic stem cell colonies. *Stem Cell Research* 17(1):122–129.

Dumoulin, V., and Visin, F. 2016. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Huang, B.; Reichman, D.; Collins, L. M.; Bradbury, K.; and Malof, J. M. 2019. Tiling and stitching segmentation output for remote sensing: Basic challenges and recommendations. *arXiv preprint arXiv:1805.12219*.

Iglovikov, V.; Mushinskiy, S.; and Osin, V. 2017. Satellite imagery feature detection using deep convolutional neural network: A kaggle competition. *arXiv preprint arXiv:1706.06169*.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature* 521(7553):436.

Lin, H.; Chen, H.; Graham, S.; Dou, Q.; Rajpoot, N.; and Heng, P.-A. 2019. Fast ScanNet: Fast and Dense Analysis of Multi-Gigapixel Whole-Slide Images for Cancer Metastasis Detection. In *IEEE Transactions on Medical Imaging*, volume 38, 1948–1958.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Maggiori, E.; Tarabalka, Y.; Charpiat, G.; and Alliez, P. 2016. Fully convolutional neural networks for remote sensing image classification. In *International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 234–241.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115:211–252.

Sermanet, P.; Eigen, D.; Zhang, X.; Mathieu, M.; Fergus, R.; and LeCun, Y. 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.

Sherrah, J. 2016. Fully convolutional networks for dense semantic labelling of high-resolution aerial imagery. *arXiv preprint arXiv:1606.02585*.

Van Etten, A. 2019. Satellite imagery multiscale rapid detection with windowed networks. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 735–743. IEEE.

Volpi, M., and Tuia, D. 2016. Dense semantic labeling of subdecimeter resolution images with convolutional neural networks. In *IEEE Transactions on Geoscience and Remote Sensing*, volume 55, 881–893. IEEE.

Appendix A: Derivation of Simplified Radius 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 radius can be simplified as follows:

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

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

Appendix B: Example U-Net Radius Calculation

Following Equation 1 for U-Net results in a required radius 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 radius 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.

$$Radius = \sum_{c=1}^{18} 2^{l_c} \tag{7}$$

By substituting the level numbers for each convolutional layer from 1 to 18 as shown in Equation 8, one obtains the minimum radius 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 2, the calculation simplifies to:

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