Convolutional Models for Biomedical Image Segmentation

Repository Link: github.com/frits-vp/cs230_finalproject

Alex Haigh, Frits van Paasschen, Joey Murphy {haighal, fritsvp, murphyjm}@cs.stanford.edu

***Abstract*—Identifying cell nuclei in microscopy images is vital to new pharmaceutical developments. However, biologists lack a robust and efficient way to detect nuclei due to natural variation in their appearances as well as differences in image capturing methods. The Kaggle 2018 Data Science Bowl—the latest installation in an annual online competition that leverages advanced data science for social good—challenges participants to deliver a deep learning model to the pharmaceutical research community that is able to identify nuclei across a diverse and complex training set. A successful implementation will aid researchers immensely in their fight to find pharmaceutical solutions to medical crises while saving both valuable research time and funding.**

I. Introduction and Related Work

For more than two decades, researchers have recognized the potential impact of machine learning to areas such as biomolecular informatics and drug discovery (Achanta et al. 1995). But only recently, with the advent of deep learning and its broadening set of applications, has there been significant interest and attention given to the development of practical tools for the pharmaceutical industry (Gawehn et al. 2016). Furthermore, the advances in deep convolutional neural networks for computer vision–and specific to this project, the development of the so-called "U-Net" by Ronnenberger et al. in 2015–have made tractable the process of biomedical image segmentation. The confluence of these accomplishments has set the stage for the emergence of a robust and efficient biomedical image classification tool. Nuclei identification gives researchers the ability to track the effects of different drugs on afflicted cells, making such a network highly valuable to the pharmaceutical industry. With the Kaggle 2018 Data Science Bowl, participants will tackle the problem of using deep convolutional networks to successfully identify cell nuclei across a diverse set of images. Our submission to the competition will build off of the successes of the U-Net architecture to deliver an accurate, generalizable model for nuclei identification in medical images.

II. Dataset

The training and testing data used is a collection of cell images that vary across cell type, image magnification, and imaging methods. For stage 1 of the competition, the training data consists of 670 unique raw images, while the test set contained 65 images. The test images are unlabeled (we submit our predictions to Kaggle for evaluation), but, for each training example, there exists the raw image taken of a cell or a group of cells, and a set of pixel masks that each display the exact position of one cell nucleus in the original picture. Each mask is unique, meaning that there is no overlap between nuclei masks. An example of a training pair can be seen in Figures 1 and 2.

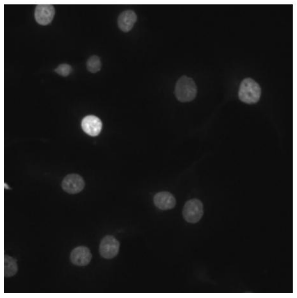

Fig. 1: Cell Microscopy Training Example[5]

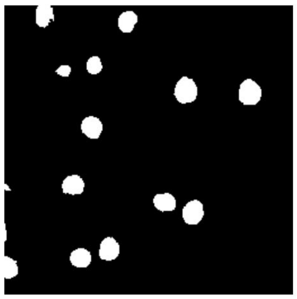

Fig. 2: Cell Microscopy Mask[5]

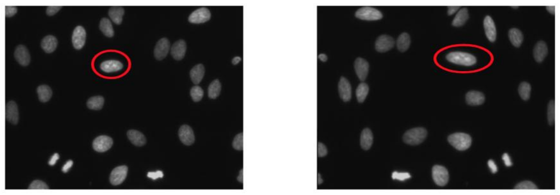

Fig. 3: Data augmentation. Left: Original image. Right: Image after horizontal flipping and piecewise affine transformation.

Copies of this dataset can be downloaded on kaggle [5]. In addition, a comma-separated-value file of run-encoded pixel values for each nucleus' mask is included in the dataset. The term run-encoded means that, for some image, the run-encoded mask 1, 3 means that pixels (1,2,3) are to be included in the final nucleus mask. As output, we are tasked with generating these values for an unseen set of data.

III. Approach

A. Data Manipulation

To train the model, we identified the smallest image in the dataset (256 x 256) and resized all images to that shape to ensure homogeneous inputs and allow faster training; similarly, we converted color images to grayscale to ensure that all training images had the same format. While the majority of images had white (or light, if color) cells on top of a black (dark) background, some images had the reverse. So, if an image had an average intensity We also used the `imgaug` python library to augment our training dataset.[6] in particular, we used three main kinds of transformations (see Fig. 3 for example):

- Random horizontal flips ($p = 0.5$)
- Random vertical flips ($p = 0.5$)
- Random piecewise affine transformations. This locally distorts images by placing a regular grid of points on an image and randomly moving the neighborhood of each point on the grid via affine transformations. In particular, each point on the grid is moved with a distance v (percent relative to image size), where v is sampled per point from $N(0, z)$, where $z \in [0.01, 0.04]$

Finally, we converted the greyscale input image to a `FloatTensor` with range $[0, 1]$ and combined all of the individual image masks into a single binary image file. The first has the effect of normalizing the input, and the second allows us to compute the Binary Cross-Entropy Loss across an entire image's segmentation map to train the model, rather than based on individual masks.

At test time, we resized the image to 256 x 256, converted to grayscale, and (depending on the image) inverted it before inputting it to our model. To generate an output segmentation map the same size as the cell, we resize the predicted map to the original size of the image, then threshold. While we have some success with this method, it is a significant source of information loss that we should address in future models.

B. Model

For the milestone, we began by extending an open-source implementation of a U-Net in PyTorch that was originally developed for the Carvana image segmentation challenge.[4] As discussed in the introduction, U-Nets are an appropriate model for semantic segmentation because they combine high resolution, local features from the "contracting path" with more global features from the upsampled expansion layer.

Figure 3 displays the base architecture of this U-Net. The layers of the U-Net are as follows. We begin with a 3x3 Convolution and ReLU later, we then repeat a 2x2 max-pool and 3x3 convolution/ReLU layer three times, finishing with a final 3x3 convolution and ReLU layer. Then, we repeat this process, replacing each 2x2 max-pool operation with a 2x2 up-convolution that reduces the number of feature channels, as described in Ronneberger et al. Finally, a 1x1 convolution is used as the final layer to decrease the number of channels to give us our binary pixel mask.

While very similar to the U-Net introduced by Ronneberger et al., our model architecture differs in a few ways:

- We assume an input image size of 256x256 (though we have the same number of layers in the U-Net
- We use "same" 3x3 convolutions instead of valid convolutions, allowing us to produce an output segmentation map with the same size as the input. To the same end, we pad our upsampled input to match the shape of the corresponding contracting path layer, rather than crop the contracting path layer.
- Rather than treating the learning task as a multi-class classification problem, our algorithm instead treats it as binary classification (1 = cell, 0 = not cell). So, the final layer is a single 1x1 convolution with sigmoid activation and a Binary Cross-Entropy Loss function, rather than a soft-max cross-entropy.
- In our initial model, we don't use a weighting function to enforce boundaries on nuclei.

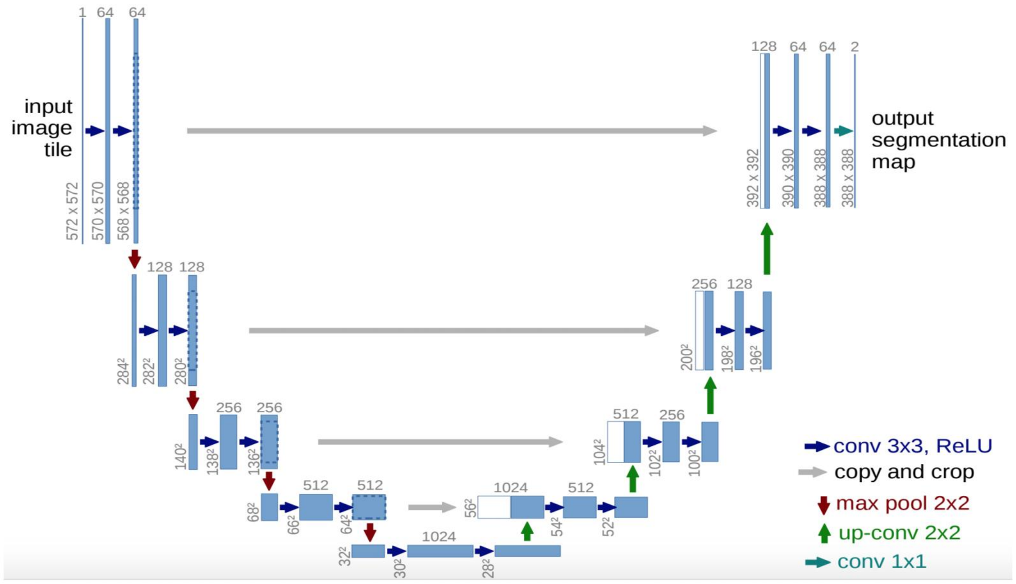

Fig. 4: Overview of the Architecture of the original U-Net [2]

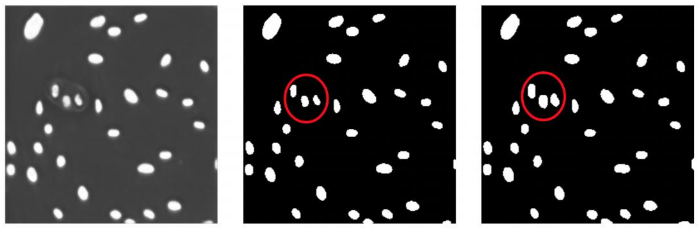

Fig. 5: Image post-processing using Otsu thresholding. Left: Model output (with pixel values [0, 1]). Middle: Output after applying Otsu thresholding. Right: Ground-truth mask. Example of difference between output and ground-truth circled in red.

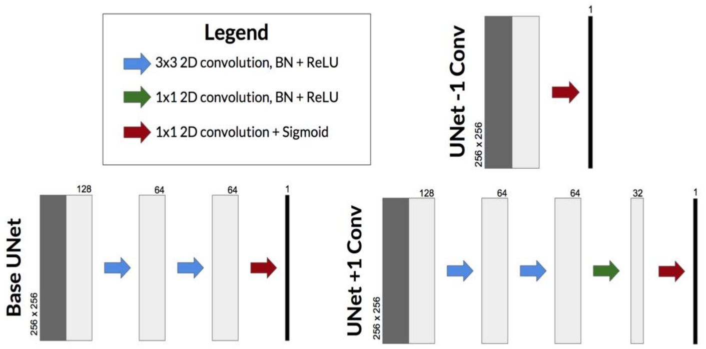
Legend
3x3 2D convolution, BN + ReLU
1x1 2D convolution, BN + ReLU
1x1 2D convolution + Sigmoid
UNet -1 Conv
Base UNet
UNet +1 Conv
256 x 256

Fig. 6: U-Net architecture modifications.

Additionally, the UNet produced a global probability distribution - pixel $[i, j]$ of the output segmentation map represents the probability that pixel $[i, j]$ in the input image was part of a cell; however, the Kaggle competition required the submission to be a discretized set of cell masks. In order convert our segmentation map into a set of masks, we first converted the map to a binary global mask using Otsu's thresholding method.[8] Otsu thresholding (Fig. 5) converts images with bimodal pixel intensity distributions into binary masks by finding the threshold t that minimizes the weighted sum of intra-class variance between pixels greater than t and pixels less than t, then classifies every $M[i, j] > t$ as 1 (part of a cell) and every $M[i, j] \leq t$ as 0 (not part of a cell). To produce individual labels, we performed a simple BFS over the binarized segmentation map and return every connected component of 1's as its own cell.

Finally, we extended the basic UNet with two different architecture modifications to explore the response of the final convolution layer in the expanding path (Fig. 6). By modifying the final layer of the network, we hoped to determine if our model's learning was either too shallow or too deep. Especially due to the heterogeneous nature of the training and test data, it is important that our model learns the general characteristics of nuclei, but not the systematic remnants related to standardizing the input images.

We train two additional models that alter the UNet's final convolutional layer. The first model adds an additional two-dimensional convolution and ReLU activation before the sigmoid output. The second removes the two convolutions and ReLU activations that immediately precede the sigmoid output.

Before evaluating any of the three models using Kaggle's evaluation metric, we conducted a hyperparamter search over learning rates, training each model for 20 epochs. To search for a successful learning rate, we randomly sampled for α in log-space and evaluated each of the three models on the same draws. For every learning rate, we recorded the best IoU scores of each of the models after each epoch. We determine our best model to be the one whose global IoU score is lowest compared to other learning rates and number of epochs. The learning rate used for our final models is of the order $\alpha \approx 10^{-5}$.

IV. Results and Evaluation

A. Evaluation Metric

We evaluate our model on several metrics that quantify model performance over our train, test, and development datasets. The Kaggle competition refers to its evaluation metric as an LB score, which quantifies model performance by measuring the mean across all test images of the average precision for each cell nucleus detection. This precision can be calculated over given a set of IoU thresholds and averaged across the set of threshold values. For a threshold t in a list of thresholds $[0.5 \rightarrow 0.95]$ (with intervals of 0.05), we consider our model to be accurate at that threshold t if the intersection over union (IoU) score:

$$IoU(\hat{Y}, Y) = \frac{Y \cap \hat{Y}}{Y \cap \hat{Y}}$$

meets or exceeds that threshold value. The Kaggle competition defines a true positive as the model predicting a specific nucleus mask with an IoU score that is greater than or equal to the threshold. Kaggle defines the precision over a set of thresholds T of a predicted set of masks $\hat{Y}$ with regards to a ground truth set of nucleus masks $masks$ as

$$P(\hat{Y}, masks) = \frac{1}{|T|} \sum_t \frac{TP(t)}{TP(t) + FP(t) + FN(t)}$$

where the label is a set of masks $masks$ and the entire score is averaged over all threshold values.

$$TP(\hat{Y}, mask, t) = \sum_{nucleus \in \hat{Y}} \mathbb{1}(IoU(nucleus, mask) > t)$$

$$FN(\hat{Y}, mask, t) = |mask| - (|\hat{Y} - FP(t))$$

$$FP(\hat{Y}, mask, t) = \sum_{nucleus \in \hat{Y}} \mathbb{1}(IoU(nucleus, mask) < t)$$

B. Results

To evaluate each of our three models we used the results from our hyperparameter search over learning rate and number of training epochs that led to the highest dev LB score. We then ran the selected models to predict masks on the test set images and submitted these masks to Kaggle for evaluation.

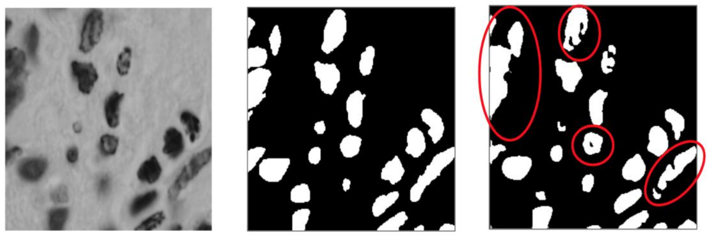
Fig. 7: Errors in our model's mask prediction. Left: Pre-processed training image. Middle: Ground-truth mask. Right: predicted mask with notable systematic errors circled in red.

Using the metrics described above, we achieved the performance shown in Table I aggregated over the entire dataset.

Model	Dev				Test
	Accuracy	Precision	IoU	LB	LB
Base U-Net	0.860	0.478	0.804	**0.342**	**0.277**
U-Net +1	0.931	0.451	0.786	**0.337**	**0.265**
U-Net -1	0.913	0.460	0.797	**0.321**	**0.274**

TABLE I: Evaluation metrics

C. Analysis

Our results show significant promise, and at a basic level, the algorithm "works," as shown by the success of the first prediction in the figure about. However, there are certainly flaws in our model that can be seen in the second image, which has several artifacts. These include masks that are conjoined because the margin separating them is so thin (as well as questionable labeling by the human since it's difficult to see even with the human eye where the boundary is) as well as masks that leak information. Some have holes in the middle, while others have finger-like projections emanating from them or have been split into two masks.

We think our errors come from a few major sources:

1) *Input Data Processing* and data heterogeneity: by resizing input images to 256 x 256, we by definition lose information about the image that could be valuable in predicting cell masks, especially since some of the input images have size up to 608x512. Converting 3-channel images to grayscale also lost information and may have

been too naive of an approach, especially since the input data came from a variety of different microscopy techniques.

2) *Image Postprocessing*: Applying a single threshold across the whole segmentation map was too simplistic of an approach because it ignores the fact that the probability of a pixel being a part of a cell is highly conditional upon whether its neighbors are. Additionally, the fact that images had to be upsampled from 256 x 256 to their original size at test time led to less accurate predictions because of information loss (interpolation is less accurate than predicting the mask outright).
3) *Model Expressivity*. For this task, the U-Net is inherently limited by the fact that we need to postprocess the output so heavily. Even though we attempted different model architectures, their relative efficacy (and therefore how expressive our model is) is limited by the extent to which we are able to successfully process our data - both pre- and post- feeding the data into our model.
4) *Differing dev and test distributions* / model bias. As the terms of the Kaggle competition stated, the test set contained images using microscopy techniques not seen in the (which, critically, had different cell shapes and sizes as well as camera perspective), and our algorithm struggled to accurately predict masks for these images.

V. Next Steps

Moving forward, we propose a few major revisions to our model, targeted to address each of the shortcomings cited.

1) *Input Data Processing*: To compensate for the loss of information caused by shrinking cells to 256x256, we propose a few potential solutions. One is to not resize images and train a model with a batch size of 1, using a high $\beta_1 (\approx 0.99)$ for the Adam optimizer (an approach used by Ronneberg et al.). To accelerate training, we could also group images by size, though this would result in highly correlated train batches since images usually with the same size were often taken using the same microscope, or randomly crop 256x256 segments of larger images. Since the model is fully convolutional, we could also simply not resize inputs images at test time. Finally, in terms of dealing with color vs. grayscale input images, we plan to train a different input convolution layer for 3-channel images, then use the same remainder of the model.
2) *Image Postprocessing*: To address the artifacts in our output segmentation map, we need more sophisticated postprocessing than a single threshold applied globally. In particular, we need to account for interdependencies between neighboring pixels, which our U-Net does not explicitly do. We plan to explore the Watershed Algorithm, which handles the image as a topographical map (intensity of each pixel corresponds to its height) and finds the lines that run along the tops of ridges (i.e. cell boundaries); the boundaries can then be filled in. We also would attempt hysteresis, which applies a "double threshold" (s, t) to the image, where every pixel (i, j) with intensity $> t$ maps to 1, (i, j) with intensity $< s$ maps to 0, and the mapping of intermediate pixels is determined based on a BFS from other "1" pixels. Simpler techniques such as filling in holes and removing masks with size ¡ 10 pixels could also help. Finally, running our U-Net on full-size images would avoid the information loss caused by upsampling.
3) *Model Expressivity*. With more sophisticated postprocessing, we might see the U-Net extensions we made have a positive impact. We can also make our model more expressive by trying larger filter sizes at the highest level (which could help fill some of the "holes" seen in cells in our output image masks), learning the interpolation weights when we upsample in the expansion path, and by modifying our loss function to overweight border pixels, which would force the algorithm to learn the boundaries correctly/prevent conjoined masks [the latter approach was used in the original U-Net paper]. We would also experiment with modifying the learning task to a 3-class softmax classification problem where we classify each pixel as background, cell, or border; postprocessing would be simpler since we know the boundary of the image. Finally, we propose using an entirely different model architecture that is designed to instance segmentation (Mask-RCNN) to avoid postprocessing entirely and directly address the competition goal.
4) *Differing dev and test distributions* / model bias. To address the bias of our model towards the dev set, we propose using heavier data augmentation (e.g. stronger shifts in our affine transformations) as well as acquiring additional data from other nuclei segmentation datasets and microscopy techniques to help the model generalize. Not resizing test images will also compensate for data loss.

VI. Contributions

- *Alex Haigh*: Cleaned dataset, debugged and adapted CS230 PyTorch vision code + Open-Source U-Net to current task, and trained the first model. Implemented data augmentation and postprocessing, visualized U-Net output. Wrote dataset, model, and next steps section; made first pass at poster.
- *Frits van Paasschen*: Set up project requirements on AWS and managed source control. Debugged CS230 PyTorch vision code and setup GPU training. Wrote evaluation code. Wrote part of model section.
- *Joey Murphy*: Authored or edited various paper sections including abstract, introduction, model extensions, hyperparameter search, and results. Helped add figures and text to poster. Wrote and integrated two U-Net model extension architectures, conducted hyperparameter searching, and trained the final models that were evaluated by Kaggle.

References

[1] Achanta, A. S., J. G. Kowalski, and C. T. Rhodes. "Artificial neural networks: implications for pharmaceutical sciences." Drug Development and Industrial Pharmacy 21.1 (1995): 119-155.

[2] Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation (2015), arXiv:1505.04597 $[cs.CV]$

[3] Gawehn, Erik, Jan A. Hiss, and Gisbert Schneider. "Deep learning in drug discovery." Molecular informatics 35.1 (2016): 3-14.

[4] Pytorch implementation of the U-Net for image semantic segmentation, with dense CRF post-processing: https://github.com/milesial/Pytorch-UNet

[5] Kaggle Data Science Bowl 2018. https://www.kaggle.com/c/data-science-bowl-2018/.

[6] imgaug: Image Augmentation for Machine Learning Experiments, Alexander Jung. https://github.com/aleju/imgaug

[7] PyTorch: An Optimized Tensor Library for Deep Learning using GPUs and CPUs, http://pytorch.org/docs/master/

[8] scikit-image: Image Processing in Python. http://scikit-image.org/

[9] Hand Signs Recognition with PyTorch (CS230 Torch Vision Code). https://github.com/cs230-stanford/cs230-code-examples/tree/master/pytorch/vision