RICE UNIVERSITY COMP540

deton8: Detector of Nuclei

Will LeVine & Gabriel Vacaliuc

April 17, 2018

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

1 Introduction

Modern medical research generates an incredible amount of data that requires significant time to be processed in some way. Often some of this processing involves tedious manual hand labeling of data, such as images, simulations, or video. To this end, it is desirable for medical research to develop automated processes so as to allow trained professionals to focus on more challenging problems than rote labeling. An example of one of these problem ripe to be solved is the labeling of microscopic cell nuclei in image data. As such, the 2018 Data Science Bowl is focused on solving this problem, or at least advancing the current state of the art. We've included the corresponding kaggle competition's description in Appendix Section A.1.

To begin understanding the task at hand, we're going to dive in and check out some images from the dataset. We'll continue by explaining exactly what needs to be done for a given image, followed by some commentary on the specific challenges of this competition.

1.1 Dataset Examples

The dataset contains an assortment of images of sizes ranging from (256, 256) to (1024, 1024), however most images are on the smaller end. Upon reading in our images, we reshape them all to (256, 256) for simplicity. Observe a sample of our dataset in Figure A.1.

1.2 Formal Description of Task

Here is a mathematical description of our task. Given a dataset $\mathcal{D}$ composed of images, $x \in \mathbb{R}^d$, we'd like to learn a function $f : \mathbb{R}^d \mapsto \mathbb{N}^d$ mapping each pixel in an input image to a discrete nucleus label in the natural numbers. It is key to understand that this task is not binary classification, but rather a task of semantic segmentation. We wish to uniquely identify each nucleus in an image.

1.3 Challenges

While semantic segmentation is itself a very challenging task given any dataset, there are some key aspects of this competition which make our task especially challenging.

1. size of training dataset

   While the recent state of the art for semantic and instance segmentation has been dramatically advanced in recent years, MS-COCO, one of the most popular datasets for such a task contains more than 330K images, with more than 200K of them labeled[1]. In contrast, our dataset contains a mere 670 labeled images coupled with a 67 image validation set, and a 3019 image test set.

2. composition of dataset

   As seen in Figure A.1, there exist a variety of image types. More specifically, the dataset was constructed using a number of different microscope types. In addition, some images contain a rather dense assortment of nuclei while some include only a few sparsely distributed.

3. mislabelings

[1] MS-COCO Dataset Website

A significant portion of the training set has mislabeled masks. That is, the reported "true" mask incorrectly represents the ground truth of the image. Several attempts have been made by the community to produce better labels.[2][3]

1.4 Metric

Common metrics for object and semantic segmentation include average precision (AP) at various intersection over union (IoU) thresholds, or mean average precision (mAP) which averages the AP for each semantic class. This competition mostly uses AP, albeit with a somewhat modified precision definition, detailed below.

$$cP(t) = \frac{TP(t)}{TP(t) + FP(t) + FN(t)} \tag{1.1}$$

where $cP(t)$ indicates the competition precision at IoU $= t$. A true positive at threshold t is when we identify an object which has an IoU of at least t with a true object, false positive when a predicted object doesn't have a corresponding true object, and a false negative if we fail to identify a true object. The full competition score can then be computed as

$$AP = \frac{1}{10} \sum_{t=0}^{9} cP(0.5 + 0.05t) \tag{1.2}$$

2 Background

2.1 Semantic Segmentation

2.2 Similar Work

3 Methods

Our pipeline consists of a few distinct steps. We begin with aggressive preprocessing, intended to collapse the data modalities into one. The processed outputs are concatenated with a set of hand-designed features resulting in multi-channel images. The result is a new, highly cleaned dataset ripe for modeling.

We then pose an intermediary binary classification problem: given a pixel, does it belong to a nucleus? Ignoring the spatial relationship of the pixels, we flatten our images into a large training set of individual d dimensional pixels, where d is the number of channels in the original image, and the corresponding label $y \in \{0, 1\}$ represents whether the pixel belongs to a nucleus. However, we frame this as a regression problem, with our outputs in the range $[0, 1]$, so as to predict the probability of a pixel belonging to a nucleus. To binarize our images, we depend on a type of Deep Neural Network (DNN) called a U-Net.

With our predicted binary mask, we proceed with the segmentation problem. This is accomplished using a Watershed Segmentation, with specific care to perform non-maximum supression on the proposed markers so as to limit oversegmentation.

[2] https://github.com/lopuhin/kaggle-dsbowl-2018-dataset-fixes
[3] https://github.com/ibmua/data-science-bowl-2018-train-set

3.1 Preprocessing & Data Whitening

Acknowledging that the data is composed of several modalities, our first step is to effectively preprocess our images into a single unimodal distribution. Our approach can be broken into 3 steps:

1. reshape & rescale pixel values to $[0, 1]$
2. invert images with a white background and dark nuclei
3. transfer a desired color distribution onto all images by whitening

The first step, reshaping and scaling is somewhat self-explanatory. We require that all images are of one size for a few reasons. Firstly, it greatly simplifies implementation and speeds up computation, as it allows us to use fixed-size arrays n-dimensional arrays rather than using a slower dynamic data structure. It also affects scale-dependent hyperparameters. If we use a range of image sizes, then the best hyperparameters might be different for each size class, increasing the number we must choose.

We follow this by an inversion of all images with a white background. In images with white backgrounds, the nuclei are dark, while in images with a black or dark background, the nuclei are white or light. Naively training a classifier on the raw data can result in examples which contribute conflicting information to our loss landscape. To rectify this, we simply invert images with a white background, so that the background becomes dark and the nuclei become light. The binary classification problem between background and nuclei becomes less a problem of local structure in an image and more one of the individual pixel content compared to neighbors.

Over the past few months and years, there's been a rise in publications involving neural style transfer, many of which who employ deep neural networks to transfer the style of one image to the content of another. We initially tried this approach, attempting to transfer the dark background and light nuclei style of several of the training images to some of the more difficult training images with lighter back grounds. Unfortunately, in practice this proved be to be far too resource demanding and time consuming.

However, applying neural style transfer to this problem is a **sin of overengineering**. Note that the color of an image is simply defined as the ratio of the red green and blue color channels of an image. We approach this problem as a problem of finding the right basis space with which to represent our data. Assuming that our data naturally exists in a latent space with unknown basis vectors, we posit that we can return our data to this space through simple linear transformations, completely unsupervised. Given a flattened d-channel image with n pixels, $X_j \in \mathbb{R}^{n \times d}$, we propose that its content was colored according to the simple matrix equation

$$X_j = S_j X_j^* \tag{3.1}$$

where S_j is the coloring matrix, and X_j^* is our original content in the latent space. Of course, provided S_j is an invertible matrix, we can succeed in full recovery of our data. However, we must be careful to transform each image to the *same* latent space. Of course, we're not the first to do this.[**whiten-stanford**][**whiten-mit**] This practice is known as whitening the data, in which we uncorrelate the features. As seen in Figure 3.1, the channels in our raw images are highly correlated. Since we'd prefer our images (at least in this initial first pixel classification model) to be greyscale, with as much variance as possible, this problem becomes incredibly tractable. Our task reduces to decorrelating the images channels (recovering our latent space), and taking the channel corresponding to the highest variance. This is accomplished using PCA whitening.

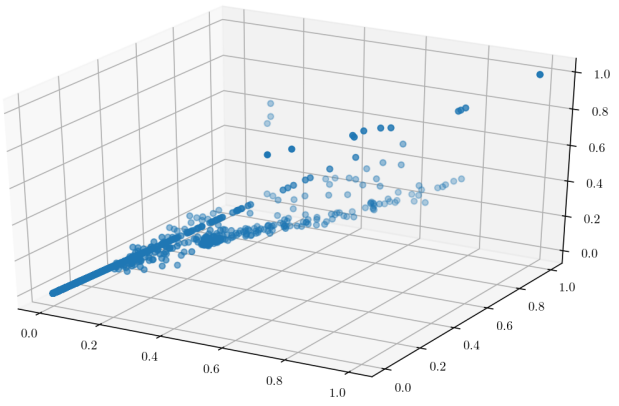

Figure 3.1: A scatter plot of a subset of pixels from our training data. Notice how clear lines form, representing various modalities, all with highly correlated features.

3.2 Hand Designed Features

After whitening the data, we mapped our data into a feature-space with goals of extracting as much information as possible and creating as much contrast between foreground and background as possible.

We came accross these features by extracting over 10 features that could encode further information, fitting the regressors, and then checking the regressors' weight vectors to see if each feature was important. After performing this process several times, we ended up with 5 features. Below, we describe the information each feature encodes and the intuition behind including that information:

1. Raw RGB Values: We first decided the include the RGB values themselves. Several other existing pipelines relied solely on the RGB values and performed quite well, meaning the RGB values must encode valuable information, so we included it.
2. Bilateral Filter: A Bilateral Filter convolves the image with a weighted Gaussian kernal. This denoises the image, while still preserving the edges. We included it to get rid of background salt and pepper.
3. 50/99 Image Rescaling: Image Rescaling widens the data distribution and increases contrast. We grabbed the 50th percentile and rescaled everything below and including that value to be 0, and we grabbed the 99th percentile and rescaled everything above that value to be 1. We included this to increase contrast between foreground and background, and to filter out salt and pepper.
4. Adaptive Histogram Equalization: Adaptive Histogram equalization increases contrast locally. We included this so that the regressors could recognize smaller nuclei among salt and papper.
5. Dilation: A dilation performs a uniform kernel convolution accross the image, thus setting each pixel to be the average of its neighbors and itself. This increases the area of each nucleus. We included this so that the regressors could pick out small nuclei.

After extracting these feature, we perform standard feature normalization on each feature.

3.3 Linear Model for Individual Pixel Probabilities

This stage of our pipeline is tasked with taking an input vector $x \in \mathbb{R}^d$ and assigning to it a probability of belonging to a nucleus. Mathematically, we wish to learn a function $f : \mathbb{R}^d \mapsto [0, 1]$ which minimizes the binary cross-entropy loss

$$\mathcal{L}(\mathcal{D}) = -\sum_{x \in \mathcal{D}} yf(x) + (1 - y)(1 - f(x)) \tag{3.2}$$

While we discussed several techniques for performing soft-classification, the largest factor of consideration for us is computational tractability. While our training dataset is rather small in terms of individual images[4], after decomposing these sets into pixels our dataset becomes quite large. In our implementation we resized all images to 256×256, which results in a 65536x increase in the number of training examples. One trick to reduce this, due to the sparsity of many training examples and the over-abundance of negative examples, is to simply filter out some of the number of negative examples. In practice, training on the entire dataset limits us to algorithms using some form of stochastic gradient descent (SGD), as it's infeasible to perform matrix operations to calculate an exact least squares solution on the full dataset.

We use two regression models in our model pipeline, both implemented in Scikit-Learn[**scikit-learn**]: the `SGDRegressor` and the `PassiveAggressiveRegressor`[**paregression**]. These models were chosen as they are fast, tested, and available implementations for linear regression. In practice, the two models focused on different subsets of our features mentioned above, and their learned outputs have considerable variance. We chose to include both as it improved our individual pixel classification precision and recall.

We'll continue by explaining in detail the mathematical problems each of these implementations solve as well as the hyperparameters involved.

3.3.1 `SGDRegressor`

This regressor solves linear regression using the standard squared loss, so as to approximate the Ordinary Least Squares (OLS) solution to the system of equations. Regularization is induced by an elastic net with a mixing coefficient to combine ℓ_1 and ℓ_2 regularization.

$$\hat{\theta} = \arg\min_{\theta} \left\{ \sum_{x \in \mathcal{D}} (y - \langle \theta, x \rangle)^2 + \alpha \left(\gamma \left| \theta \right|_1 + (1 - \gamma) \left| \theta \right|_2^2 \right) \right\}$$

where γ is our mixing coefficient, and α is our regularization term.

3.3.2 `PassiveAggressiveRegressor`

This regressor has roots in standard linear regression as well as Support Vector Machines (SVM).[**paregression**] It tackles standard binary classification as well as regression, however it's weight update is quite unique. At step t, the new weight vector w_{t+1} is defined as

$$w_{t+1} = \arg\min_{w \in \mathbb{R}^n} \frac{1}{2} \left| w - w_t \right|^2$$

such that $\ell(w; (x_t, y_t)) = 0$, where ℓ represents the hinge loss at step t. Let's explore what happens for a given w. Suppose that the hinge loss is already 0. Then the algorithm is completely *passive*, as it simply

[4]Dataset Sizes: Training (670), Validation (65), Testing (3036)

chooses $w_{t+1} = w$. Else, we *aggressively* force w_t to satisfy ℓ_t by projecting in onto the space of vectors such that $\ell_t = 0$.

In practice, much like fitting an SVM, we introduce a slack variable to control the *aggressiveness* of the regressor, as seen below.

$$w_{t+1} = \arg\min_{w \in \mathbb{R}^n} \frac{1}{2} |w - w_t|^2 + C\xi$$

such that $\ell(w; (x_t, y_t)) \leq \xi$ and $\xi \geq 0$.

3.4 UNet as a Learned Binarizer

Taking our predictions from our linear models and our 5 features, we complete our pipeline using a DNN functioning as a leared binarizer.

3.5 Semantic Segmentation using Watershed

4 Results

4.1 Preprocessing & Color Transfer

4.2 Hand Designed Features

4.3 Linear Model for Individual Pixel Probabilities

4.4 UNet as a Learned Binarizer

4.5 Semantic Segmentation using Watershed

5 Discussion

6 Conclusion

A Appendix

A.1 Description of the 2018 Data Science Bowl

Spot Nuclei. Speed Cures.

Imagine speeding up research for almost every disease, from lung cancer and heart disease to rare disorders. The 2018 Data Science Bowl offers our most ambitious mission yet: create an algorithm to automate nucleus detection.

We've all seen people suffer from diseases like cancer, heart disease, chronic obstructive pulmonary disease, Alzheimer's, and diabetes. Many have seen their loved ones pass away. Think how many lives would be transformed if cures came faster.

By automating nucleus detection, you could help unlock cures fasterfrom rare disorders to the common cold. Want a snapshot about the 2018 Data Science Bowl? View this video.

Why nuclei?

Identifying the cells' nuclei is the starting point for most analyses because most of the human body's 30 trillion cells contain a nucleus full of DNA, the genetic code that programs each cell. Identifying nuclei allows researchers to identify each individual cell in a sample, and by measuring how cells react to various treatments, the researcher can understand the underlying biological processes at work.

By participating, teams will work to automate the process of identifying nuclei, which will allow for more efficient drug testing, shortening the 10 years it takes for each new drug to come to market. Check out this video overview to find out more.

A.2 Dataset Example

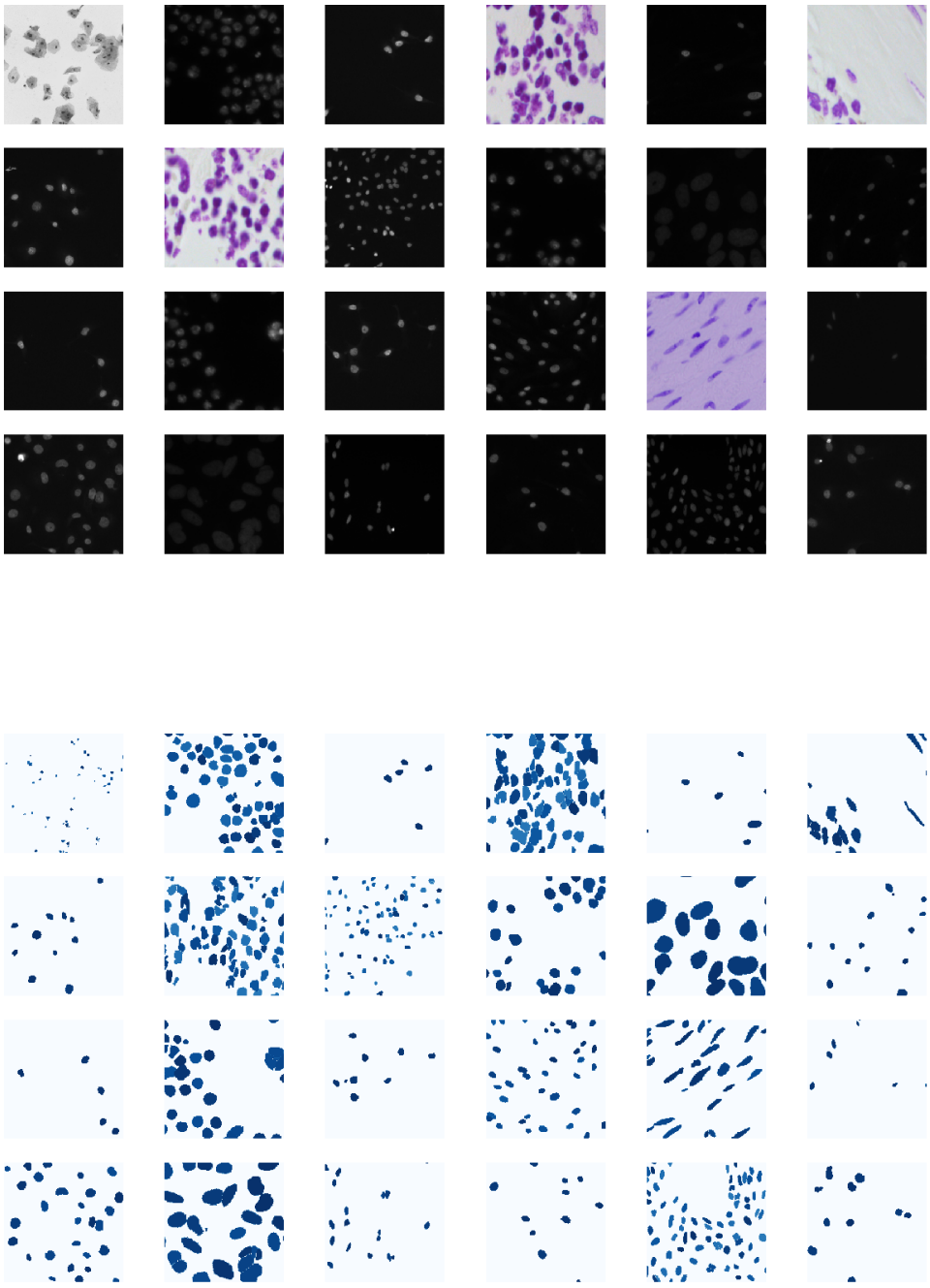

Figure A.1: An assortment of images from our dataset. All have been resized to 256 x 256. (top) raw image data (bottom) true masks