Machine Learning Engineer Nanodegree

Capstone Project

Christine D
April 17th, 2018

I. Definition

(approx. 1-2 pages)

Project Overview

The purpose of the capstone project is to create an algorithm that can automatically identify the nuclei in different types of cells as part of a competition organized by Kaggle, the 2018 Data Science Bowl (https://www.kaggle.com/c/data-science-bowl-2018).

Robots are able to mix new molecules with sick cells and take pictures of the cells in a fairly efficient way. An important part of the analysis process requires the visual examination of the cells as they are subjected to different treatments. The automation remains limited as biologists need to study the images in order to decide on the drugs' efficacy. The automatic recognition of cells' nuclei can speed up the drug discovery and testing process, which could shorten the time period needed to find new drugs (https://datasciencebowl.com/2018dsbtutorial/).

Research shows that nuclear segmentation is complicated by (1) the diversity of nuclei (in terms of size and nuclear type) and (2) the potential overlap of different nuclei [1]. Olaf Ronneberger, Philipp Fischer, and Thomas Brox [2], in a research paper dated 18th May 2005, concluded that the Unet combined with data augmentation achieved good performance. The data augmentation allowed a reduced number of training images and the model needed only about 10 hours on a NVidia Titan GPU (6 GB).

Problem Statement

The algorithm automates the low-value but necessary task of identifying cells' nuclei. The segregation of nuclei will help biologists focus on solutions instead of spending time manually flagging nuclei. The increased automation in the research process could shrink the time-to-market for drugs, leading to potentially less pricy drugs.

This project deals with a segregation problem whereby an algorithm takes as input a picture of cells containing nuclei to identify the nuclei. The output is the exact location of each nuclei in the input image. The project uses deep learning to classify

and segregate the nuclei. Two neural networks are used (please refer to the Algorithms and Techniques section)

The purpose of this capstone project is to create a Unet neural network, measure its performance and then improve its performance by augmenting data. A benchmark network made up of the three first layers of the Unet will be used to assess the added benefits of the additional layers as well as the contraction and upsampling used in the Unet architecture. I dropped the max pool layer from the benchmark architecture and replaced it by a convolutional layer in order to study the effect of the contraction and dilation.

The performance is measured using the mean IoU (intersection over Union) metric. The definition is explained in the section below.

The problem is quantifiable and measurable: the main metric is the proportion of nuclei which were correctly identified. A high-performing algorithm needs to correctly segregate nuclei from other types of structures within the cell. The problem is also replicable and observable: biologists regularly assess drugs and compounds in many laboratories in the world. The algorithm accuracy can be compared against human measurements.

Metrics

A score is computed to assess the precision of the algorithm as follows:

The competition is evaluated on the mean average precision at different intersection over union (IoU) thresholds.
The IoU of a proposed set of object pixels and a set of true object pixels is calculated as:

$$IoU(A, B) = \frac{A \cap B}{A \cup B}$$

The threshold values range from 0.5 to 0.95 with a step size of 0.05: (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95)

Intersection over Union is a valuable metric to measure the accuracy of a nuclei detector over the dataset. The IoU measures the ratio of the area of overlap over the area of union. The higher the number, the more overlap areas A and B have. IoU takes values between 0 and 1. When IoU is equal to 0, there is no overlap between the two areas. When IoU is equal to 1, A and B are the same. In this project A and B represent respectively the ground-truth mask (identified manually) and the prediction. An IoU greater than 0,5 is normally considered a good prediction, hence the range chosen above [0.5, 0.95].

For instance, at a threshold of 0.6, a predicted object is considered a hit if its intersection over its union with a ground truth is greater than 0.6

The average precision of a single image is calculated as the mean of the precision values at each IoU threshold *t*:

$$\frac{1}{|thresholds|}\sum_t \frac{TP(t)}{TP(t)+FP(t)+FN(t)}$$

where:

TP(t) = True positive which is counted when a single predicted object matches a ground truth with an IoU above the threshold t.

FP(t) = False positive which is counted when a predicted object has no associated ground truth project.

FN(t) = False negative which is counted when a ground truth object has no associated predicted object.

The higher the metric, the more accurate the model is as the number of true positives becomes much larger than the number of false positives (when a nucleus is falsely predicted) and the number of false negatives (when there is no nuclei detected while there is one in reality).

Because this metric measures the accuracy over a number of IoU threshold, it is well suited to assess the performance of the segregation models I used in this project.

II. Analysis

(approx. 2-4 pages)

Dataset and inputs

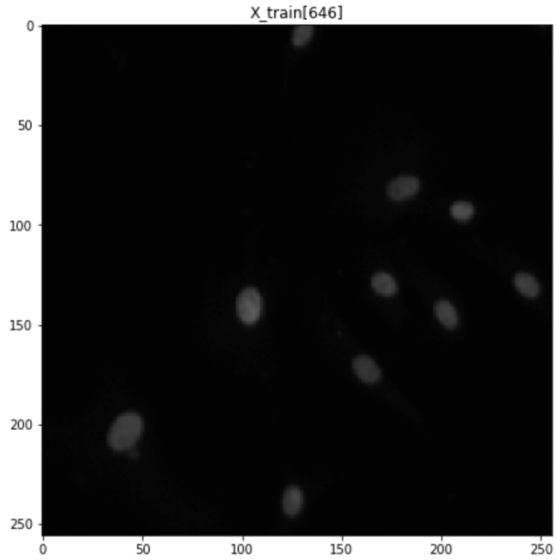
The Kaggle competition webpage (https://www.kaggle.com/c/data-science-bowl-2018/data) provides the needed training data and a separate set of images for testing. The training data contains the original image and a series of masks. The testing data only contains the original test image.

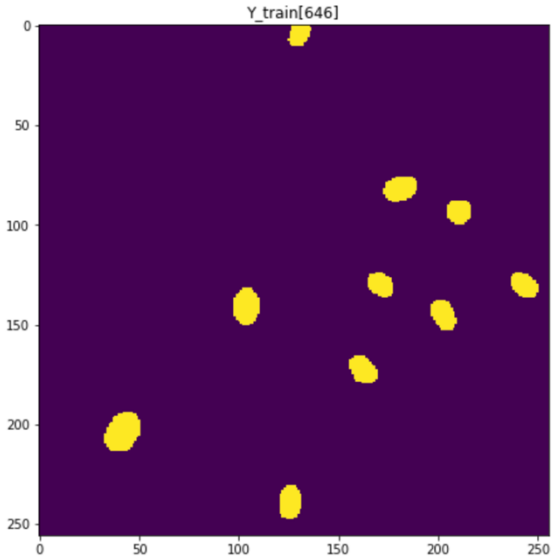
File Descriptions

- /stage1_train/* - training set images (images and annotated masks). The training set has 670 folders for 670 images in png format (one folder per image). Each folder has two sub-folders (*masks* and *images*). The sub-folder *images* has only one png file (the image) and the sub-folder *masks* contains as many files as nuclei.

- /stage1_test/* - stage 1 test set images (images only). The set has 65 images in jpg format.

Here is an example of a training image (number 646) and the corresponding image with delineated nuclei.





Data Exploration

There are 670 training images and 65 test images with varying sizes.

Training Set (total = 670 images)

	Size	Count
0	(1040, 1388, 3)	1
1	(360, 360, 3)	91
2	(520, 696, 3)	92
3	(256, 256, 3)	334
4	(603, 1272, 3)	6
5	(512, 640, 3)	13
6	(260, 347, 3)	5
7	(256, 320, 3)	112
8	(1024, 1024, 3)	16

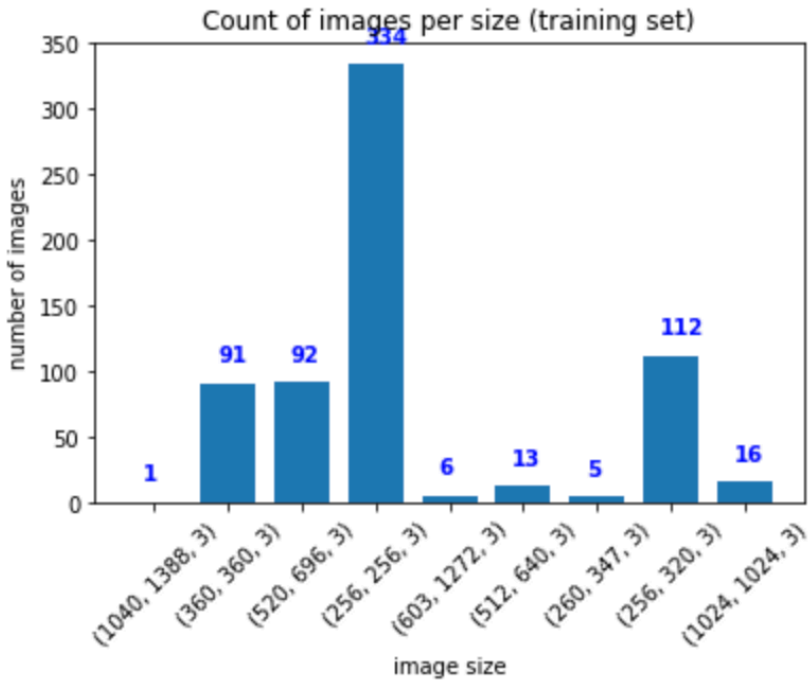
Test set (total = 65 images)

	Size	Count
0	(260, 347, 3)	4
1	(390, 239, 3)	1
2	(519, 253, 3)	4
3	(512, 640, 3)	8
4	(524, 348, 3)	4
5	(519, 161, 3)	2
6	(520, 348, 3)	4
7	(512, 680, 3)	8
8	(519, 162, 3)	2
9	(520, 696, 3)	4
10	(256, 256, 3)	24

I analyzed the size distribution of the images in order to determine the optimal size of the images to be used for training the model. Please refer to the exploratory section below for a more detailed analysis.

Exploratory Visualization

Training Set



Test Data

Both the training and test sets have images of varying size. The distribution for the training dataset shows that about half of the images (334 out of 670 or 49.8%) have a size equal to (256, 256, 3). All images in the training set have width or height greater than 256. The largest group of images in the test dataset comprises have a size equal to (256, 256, 3): there are 24 out of 65, (or 37% of the test set) images with that size. Only 10 images out of 65 (or 15%) have a width or height less than 256 (4 images have a height of 253, which is not very dissimilar to 256).

Because the largest group of images had a size equal to (256, 256, 3), I chose to resize all images to (256, 256, 3). If I had chosen a smaller size (to accommodate the images with a smaller height or width), I would have lost precious information for the training process. A larger size would have led to longer computations.

Algorithms and Techniques

The inputs to the models are the training images and test images mentioned in the last paragraph.

I built three models:

- Two were built on the Unet architecture but trained with two sets of data (the original data and the augmented data).
- One is the benchmark model (described below).

The two architectures (Unet and the benchmark) use convolutional layers because convolutional neural networks (CNNs) have exhibited great performance in the computer vision field. An image can be considered a matrix of integers. The grid of number is fed into a convolutional layer which uses a smaller sliding window (or filter) that computes a number saved in a new array (the size of the array cannot increase). CNNs use a combination of convolutional layers (like the one I have just described) and max pool layers among others. Max pool layers reduce the size of the input by calculating the maximum over a sliding window. This step amount to keeping the most important information in a portion of the image. Dropout layers are also used in CNNs to ensure that the model explores all nodes. Dropout layers ignore a percentage of the nodes (this percentage is specified by the developer), hence forcing the model to adjust the weight associated with other nodes.

I benchmarked the two first models against the benchmark model and quantified the enhanced performance of the model trained with the augmented data.

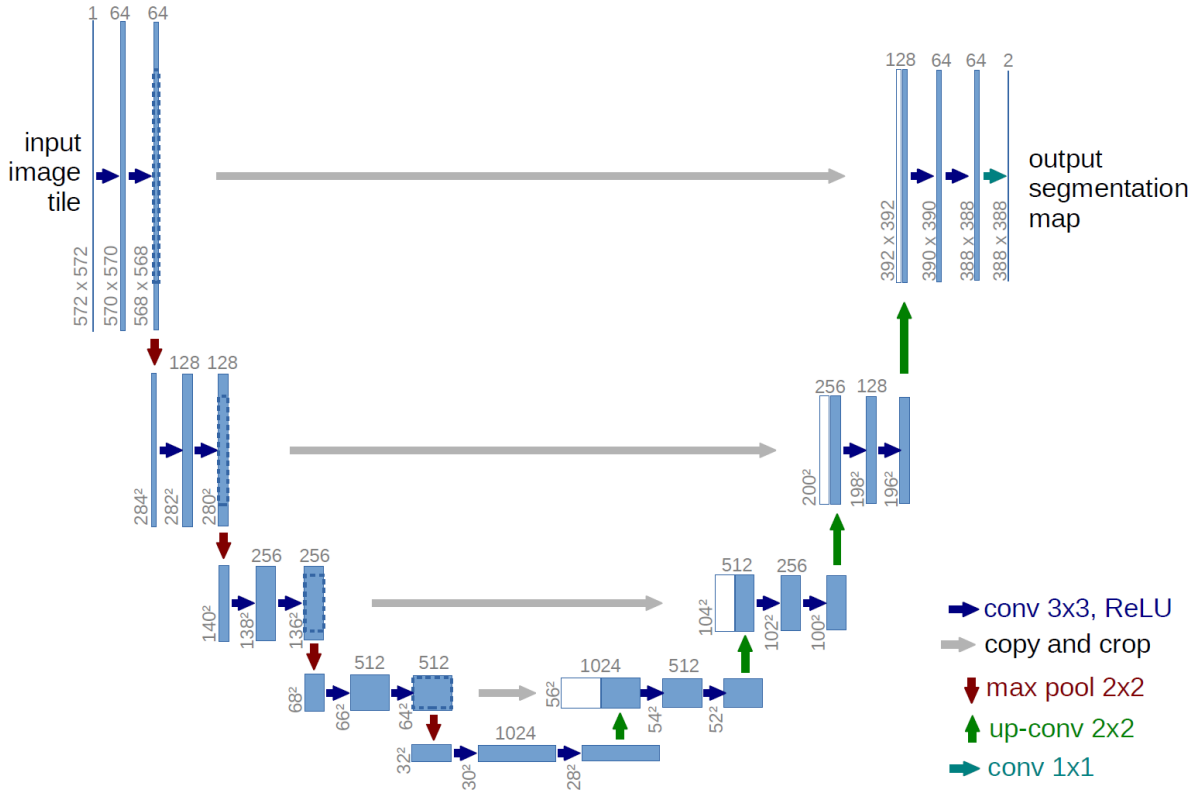
Unet

The Unet architecture contains 36 layers and was inspired from the research report released by Ronneberger, Fischer, and Brox [2]. The Unet avoids over-fitting by using dropout layers. The neural network used Keras while the IoU (defined in the Evaluation Metrics section) used Tensorflow.

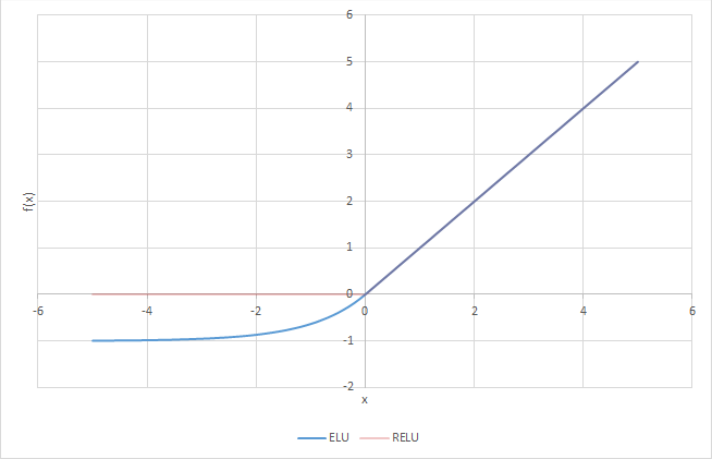
I trained the neural network with images smaller than the images used by Ronneberger, Fischer, and Brox. I chose a size equal to (256, 256, 3) instead of (572, 572,3)

The neural network uses 3 types of layers:

- Convolutional layers (16 (3x3) filters and one (1x1) filters are applied to the layers). The stride is (1,1) i.e., the filter jumps one pixel at a time as the filter slides around the image. The larger the stride the smaller the feature maps. The activation function is 'elu' (exponential linear unit – please refer to the graph below) and the kernel initializer is 'he_normal'. The kernel initializer draws from a uniform distribution within [-sqrt(6/fan_in), +sqrt(6/fan_in)] where fan_in is the number of inputs units in the weight tensor. The padding is "same" (because the stride is 1, the output size is the same as the output size). This choice of padding tries to pad evenly left and right, but if the amount of columns to be added is off, it adds the extra column to the right.
- Max pooling layers (with a (2x2) filter): the output is an image half the size of the input.
- Dropout layers (at a rate equal to 0.1, i.e. 10% of the inputs units to the layer are dropped for each epoch).
- Deconvolutional layers (or up conv layers): as the figure shows below, deconvolutional layers construct layers from an input in an upward direction. The output is an image twice as large in our case. The deconvolutional layers are built using the Conv2DTranspose function in Keras.



Source: Olaf Ronneberger, Philipp Fischer, and Thomas Brox [2]



ELU and RELU activation function

The model is compiled using the adam optimizer and binary_crossentropy for the loss function. The metrics is mean_iou (calculated from a predefined function in tensorflow).

The Unet is trained with the original data and the augmented data (please refer to the implementation for more details). The table below summarizes the main parameters.

	Unet with original data	Unet with augmented data
Nb of epochs	50	30
Optimizer	Adam	adam
Loss function	Binary_crossentropy	Binary_crossentropy
Batch size	16	16
Cross validation split	0.1	N/A (the data was split with the same ratio without using the cross validation split)
Metric	Mean_iou	Mean_iou
Earlystopping	Patience = 5 (i.e. the loss function does not decrease for 5 epochs)	Patience = 5 (i.e. the loss function does not decrease for 5 epochs)

The number of epochs was reduced after data augmentation because the network learns faster.

The benchmark network
Please refer to the paragraph on the benchmark model below

Benchmark

I used the following 4-layer neural network as a benchmark.

```
c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (inputs)

c1 = Dropout(0.1) (c1)

c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c1)

outputs = Conv2D(1, (1, 1), activation='sigmoid') (c1)
```

I have chosen a benchmark model made up of the first three layers of the Unet in order to understand the added value of the rest of the network. Please note that I mentioned the first four in my initial paper but I decided to drop the max pool layer to quantify the benefits of the contraction and then dilation found in the Unet architecture. The Unet architecture decreases the size of the output with max pool layers and then increases the resolution of the output by using upsampling layers. By comparing the benchmark model and the Unet architecture while using the same dataset, I will be able to quantify the added performance by the further contractions in the first half of the architecture and the upsampling operations in the second half of the architecture.

The same metric is used for both the Unet and the benchmark model.

In the refinement section, I compare the initial benchmark in the proposal and the final benchmark used in this project. The architecture shown above is the final benchmark.

III. Methodology

(approx. 3-5 pages)

Methodology

I used the following steps:

- Data visualization and preprocessing (load the data, data preparation)
- Design of the Convolutional Neural Network (CNN)
- Training of the Convolutional Neural Network (for each model)
- Application of the model to the testing data (for each model)

The outputs are:

- Model obtained with the original data and the benchmark neural network (model in h5 format)
- Csv file with encoded masks for the benchmark model (csv format)
- First model obtained with the original data and the Unet architecture (first model in h5 format)
- Csv file with encoded masks for the first model (csv format)
- Second model obtained with the augmented data and the Unet architecture (second model in h5 format)

- Csv file with encoded masks for the second model (csv format)

I used python 3.5 and the following libraries:
Anaconda: http://continuum.io/downloads

numPy: http://www.numpy.org/
pandas: http://pandas.pydata.org/
sys: https://www.scipy.org/
os: https://docs.python.org/2/library/os.html
random: https://docs.python.org/3/library/random.html
warnings: https://docs.python.org/2/library/warnings.html
time: https://docs.python.org/2/library/time.html
matplotlib.pyplot: http://matplotlib.org/
collections: https://docs.python.org/2/library/collections.html
skimage.io: http://scikit-image.org/docs/dev/api/skimage.io.html
skimage.transform: http://scikit-image.org/docs/dev/api/skimage.transform.html
skimage.morphology](http://scikit-image.org/docs/dev/api/skimage.morphology.html
keras.preprocessing.image](https://keras.io/preprocessing/image/
keras.models: https://keras.io/models/about-keras-models/
keras.layers: https://keras.io/layers/about-keras-layers/
keras.layers.convolutional: https://keras.io/layers/convolutional/
keras.layers.core: https://keras.io/layers/core/
keras.layers.pooling: https://keras.io/layers/pooling/
keras.layers.merge: https://keras.io/layers/merge/
keras.callbacks: https://keras.io/callbacks/
keras.backend: https://keras.io/backend/
sklearn.model_selection: http://scikit-learn.org/stable/model_selection.html)
tensorflow: https://www.tensorflow.org/
tqdm: https://tqdm.github.io/
jupyter: http://ipython.org/notebook.html

The calculations were run on Amazon Web Services (AWS). I created an instance using the following:
- image: Deep Learning AMI with Source Code (CUDA 8, Ubuntu): https://aws.amazon.com/marketplace/pp/B06VSPXKDX
  This AMI comes with TensorFlow 1.3.0 and Keras 2.0.8 with TensorFlow as default backend.
- Instance type: p2.xlarge
- Security Group:
 - Port range: 8888 (Custom TCP Rule, Source 0.0.0.0/0)
 - Port range: 22 (Custom TCP Rule, Source 0.0.0.0/0)
 - Port range: 8888 (ssh, Source 0.0.0.0/0)

The process used to set up the instance is summarized in the README.md file.

Data Preprocessing

Initial preprocessing

The images underwent the following transformations:

- resize to (256, 256, 3)
- gray (the process transforms a 3d matrix into a 1d matrix), which reduces the amount of data to manipulate while keeping enough information to segregate the nuclei
- normalize by dividing each pixel by 255 so that each final number in the numpy array representing the image is a float between 0 and 1

The original dataset was used on the benchmark network and then the Unet architecture.

Data augmentation

The same Unet was then trained on the augmented data. The data was augmented as follows:

- I split the training set (images and corresponding masks) into two sets (one used for training, one used for validation (even though the validation set is named using the word test)).
- I created four generators with the same seed for the set of training images, the set of training masks, the set of test images and the set of test masks. The parameters are detailed below:
    - shear_range=0.2 (Shear angle in counter-clockwise direction in degrees)
    - zoom_range=0.2 (the zoom range is [lower, upper] = [1-zoom_range, 1+zoom_range])
    - horizontal_flip=True (Randomly flip inputs horizontally)
    - vertical_flip=True (Randomly flip inputs vertically)
    - width_shift_range=0.2 (fraction of total width): the generator computes a range for random horizontal shifts.
    - height_shift_range=0.2 (fraction of total width): the generator computes a range for random horizontal shifts.
    - rotation_range=90 (Degree range for random rotations)
    - fill_mode='reflect' (points outside the boundaries of the input are filled according to the given mode: abcddcbalabcdldcbaabcd)

The data was augmented in 12.16 seconds.

Implementation

I used the following steps:

- Data visualization and preprocessing (load the data, data preparation)
- Design of the Convolutional Neural Network (CNN): benchmark and Unet
- Definition of mean_iou metrics using the mean_iou function from the metrics module in tensorflow.
- Training of the Convolutional Neural Networks with training/validation split equal to 0.1 in this order:

    - Benchmark model with original preprocessed data
    - Unet model with original preprocessed data
- Data Augmentation
- Training of the Unet model with augmented data
- Application of the model to the testing data (for each model) in order to generate csv files
- Comparison of the 3 models in the analysis.pdf file.

The mean_iou (mean intersection-over-union) function requires the labels, predictions and the number of classes (2 here as the predictions can take one of tow labels: nuclei or non-nuclei). Details are available on the tensorflow online documentation (https://www.tensorflow.org/api_docs/python/tf/metrics/mean_iou).

The training was executed using the following parameters:

	Benchmark Model	Unet with original data	Unet with augmented data
Nb of epochs	50	50	30
Optimizer	Adam	Adam	Adam
Loss function	Binary_crossentropy	Binary_crossentropy	Binary_crossentropy
Batch size	16	16	16
Cross validation split	0.1	0.1	N/A (the data was split with the same ratio without using the cross validation split)
Metric	Mean_iou	Mean_iou	Mean_iou
Earlystopping	Patience = 5 (i.e. the loss function does not decrease for 5 epochs)	Patience = 5 (i.e. the loss function does not decrease for 5 epochs)	Patience = 5 (i.e. the loss function does not decrease for 5 epochs)

Improvement and Refinement

I tried to use zca_whitening in the data generator during the data augmentation phase. I was not able to implement a solution because of a memory error message. This step seems to utilize too much memory.

I tried to decrease the size (128, 128, 3) but obtained the same memory error message. I finally decided against using zca_whitening because the contraction in the image size would lead to a loss of information that cannot be compensated by data augmentation. Because of the difficulty in segregating overlapping or contiguous nuclei, it is important to keep as much spatial information as possible in the input images.

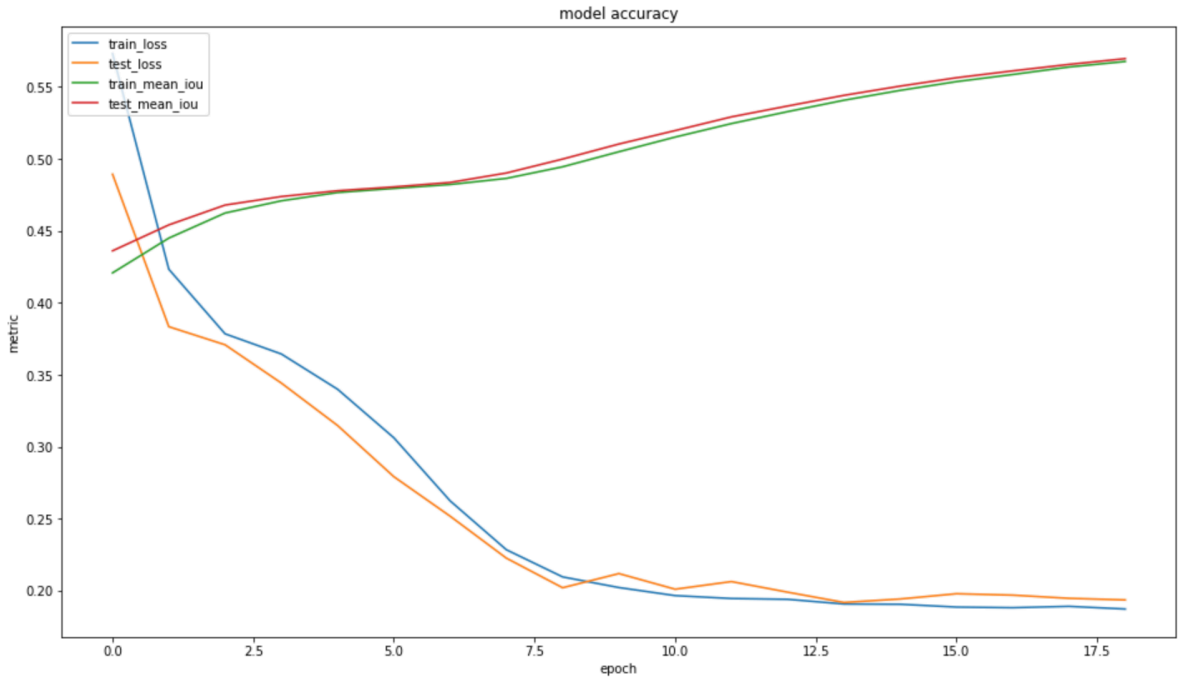
I modified the benchmark architecture mentioned in the proposal. Initially I wanted to keep only the first 4 layers (2 convolutional layers, 1 dropout layer and one max pooling layer). Because of the max pooling layer, I had to add the Conv2DTranspose layer to increase the size of the output.

```
inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH))
c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (inputs)
c1 = Dropout(0.1) (c1)
c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c1)
p1 = MaxPooling2D((2, 2)) (c1)
u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (p1)
outputs = Conv2D(1, (1, 1), activation='sigmoid') (c1)
```

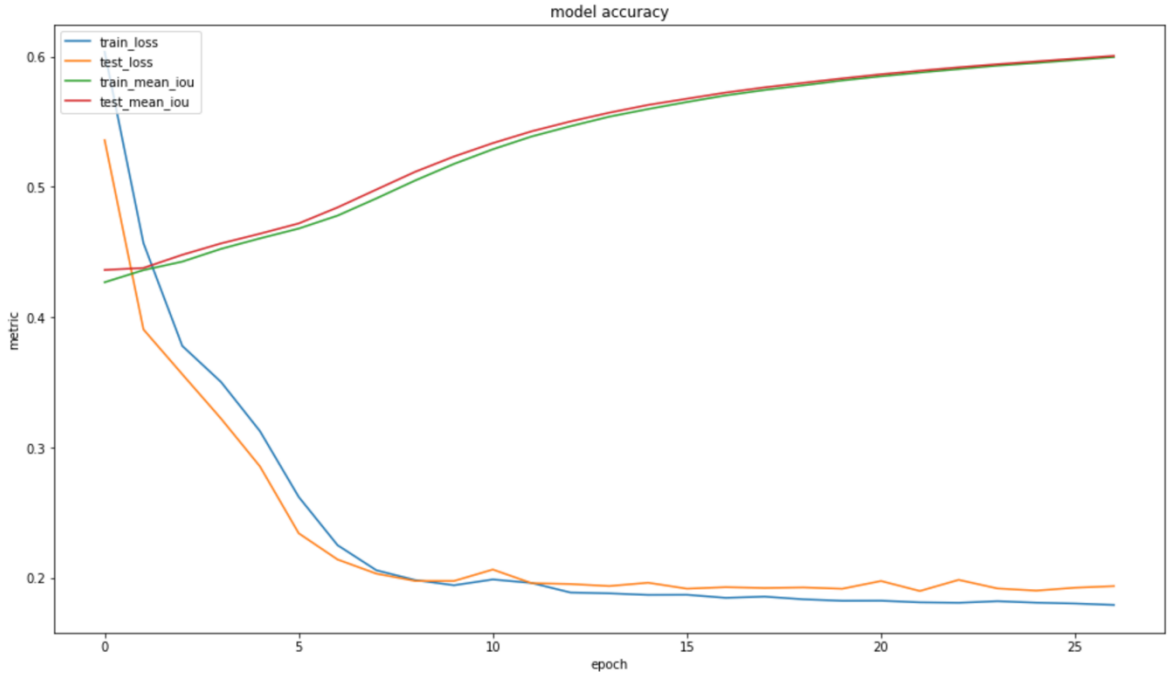
The table below summarizes the two benchmark options.

	As per proposal (2 Conv2D layers + 1 Dropout layer + 1 max pooling layer) + 1 Conv2DTranspose	Final implementation (detailed in the benchmark section)
Number of parameters	2,785	2,785
Number of epochs	50	50
Earlystopping	Epoch 27	Epoch 19
Final validation mean IoU	0.5696	0.6005
Time to train model (seconds)	916 (15 minutes 16 seconds)	1308 (i.e. 22 minutes)

Initial Benchmark Model



Final Benchmark Model



We can see that the initial benchmark has a lower mean IoU for the validation set (+5.4% increase over the initial benchmark model). It takes less time to train than the final model I used. I chose the modified benchmark because (1) it yields a higher mean IoU for the validation set and (2) it will help me compare the benefits of downsampling followed by upsampling that merges outputs from the contraction path.

Please note that the final benchmark model is simply called the benchmark model in the rest of this paper.

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

The main results are summarized in the table below

	Benchmark Model	Unet with original data	Unet with augmented data
Total number of parameters	2,785	1,941,105	1,941,105
Time to augment the data (seconds)	N/A	N/A	12.16
Time to train model (seconds)	1,308 (i.e. 22 minutes)	11271.82 (i.e. 3 hours 8 minutes)	28258.51 (i.e., 7 hours 51 minutes)
Final mean_iou (validation set)	0.6005	0.8075	0.8376
Time to predict results (seconds)	32.56	57.61	60.69

We can see that the large improvement of the Unet model over the benchmark model. The final IoU on the validation dataset increases by 34% (from 0.6019 to 0.8075) with the Unet architecture. It takes much more time to train the Unet than the benchmark model (3 hours 8 min vs 22 minutes). The training time more than doubles with the augmented data (up to 7 hours 51 minutes) for an increase in performance of 3.7%. This increase in the final mean IoU, while not being large, is still significant.

The time needed for the models to predict the nuclei also increased with the Unet (from 33 s to 58s). While the increase in prediction time is significant, it is still

reasonable (less than 1 minute for 65 images, i.e. less than 0.9s per image). This is still a significant improvement over a manual examination of pictures by humans. It is interesting to note that the data augmentation does not lengthen considerable the prediction time for the Unet trained with the augmented data. An image takes about 1s to be analyzed by the Unet trained on augmented data.

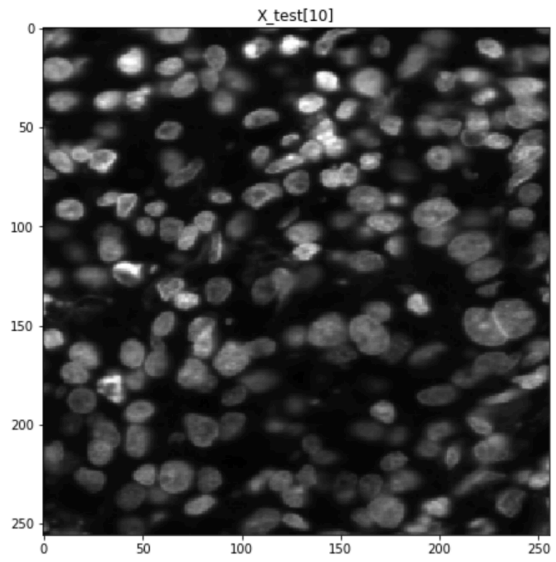
The model achieves a higher mean_iou with the augmented data. This can be explained by the use of perturbations on the training data. The model achieved with the Unet architecture and the augmented data should be more robust that the model obtained with the Unet and the original data. A visual examination shows that while the Unet with augmented data is able to spot nuclei, it is still not able to segment them correctly.

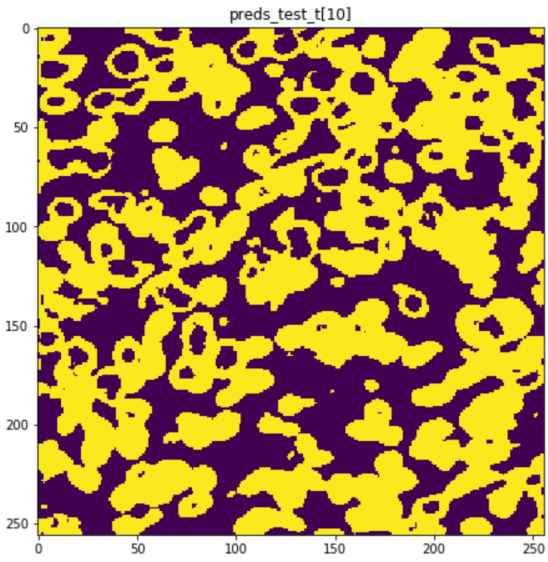
The model provided a low score (~0.256) when tested with the test dataset on Kaggle (submission on 9th April 2018). This low score shows that the model does not generalize well to unseen data despite the different sizes and shapes of nuclei in the training dataset.

Qualitative analysis

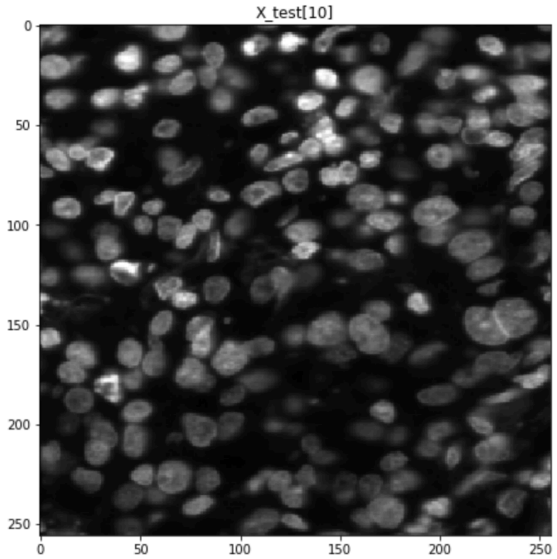
A visual examination shows that the model is not good at segregating contiguous or overlapping nuclei. I picked an image (image 10) that is complex enough to test the models.

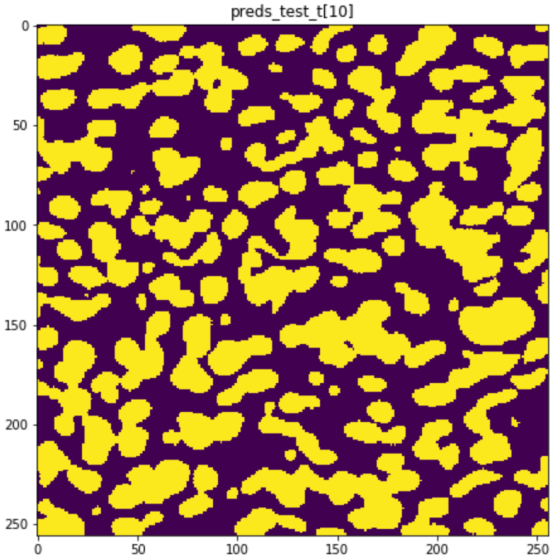
Benchmark



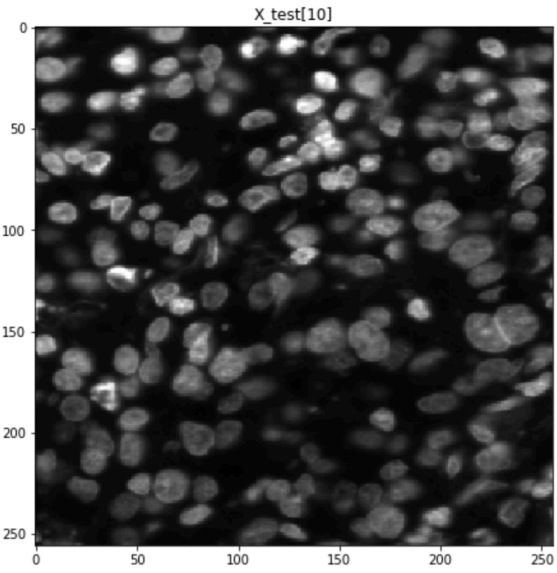


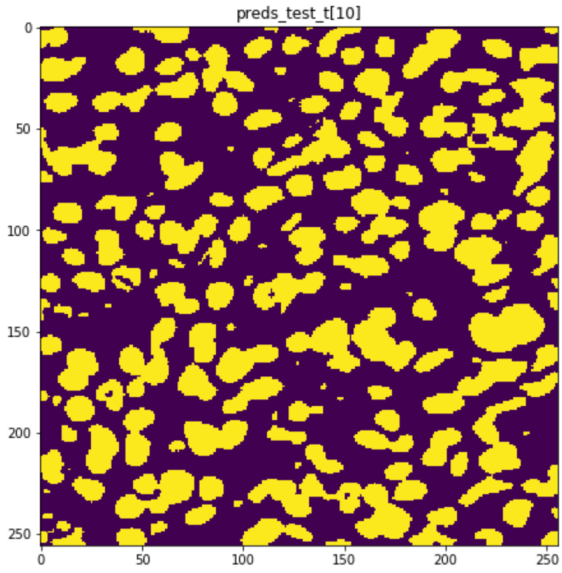
Unet with original data





Unet with augmented data





We can see a clear improvement in the nuclei localization task. The benchmark model identifies the nuclei but creates large clumps and negates the center of the nuclei. The Unet trained with the original data fixes that issue as it does not have

“missing centers”. But it tends to coalesce many nuclei together. The Unet trained on the augmented data is more selective and does not create the same clumps of nuclei. Despite this improvement, the Unet model is not able to segregate nuclei correctly.

Justification

We can attribute the improvement in performance from the benchmark and the Unet to the contraction path and upward sampling. The upward sampling integrates information from the contraction path. This linkage between the the contracting and dilating segments helps improve the performance of the Unet neural network.

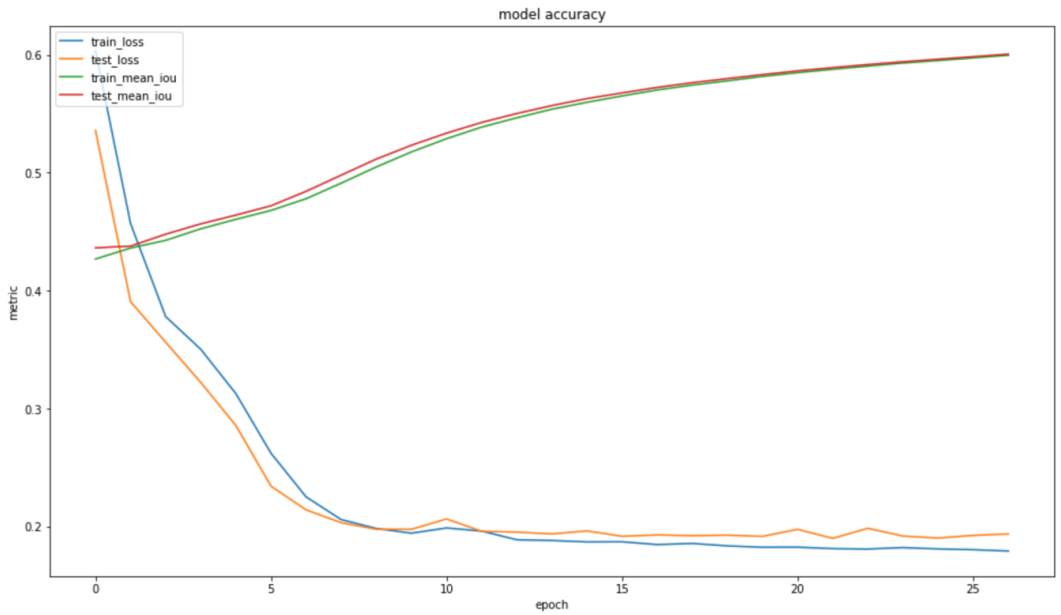
While the improvement in the mean IoU between the Unet trained with the original data and the Unet trained with the augmented data is not dramatic (+3.7%), it is still significant. The Unet model with augmented data benefits from the perturbations used in the data generator (rotations, shear angle, zoom, vertical and horizontal shifts...).

V. Conclusion

Free-Form Visualization

The qualitative analysis above combined with the study of the metrics improvement shows a clear improvement. The best model is the Unet trained with augmented. This model is still not able to replace biologists in the tedious task of identifying nuclei.

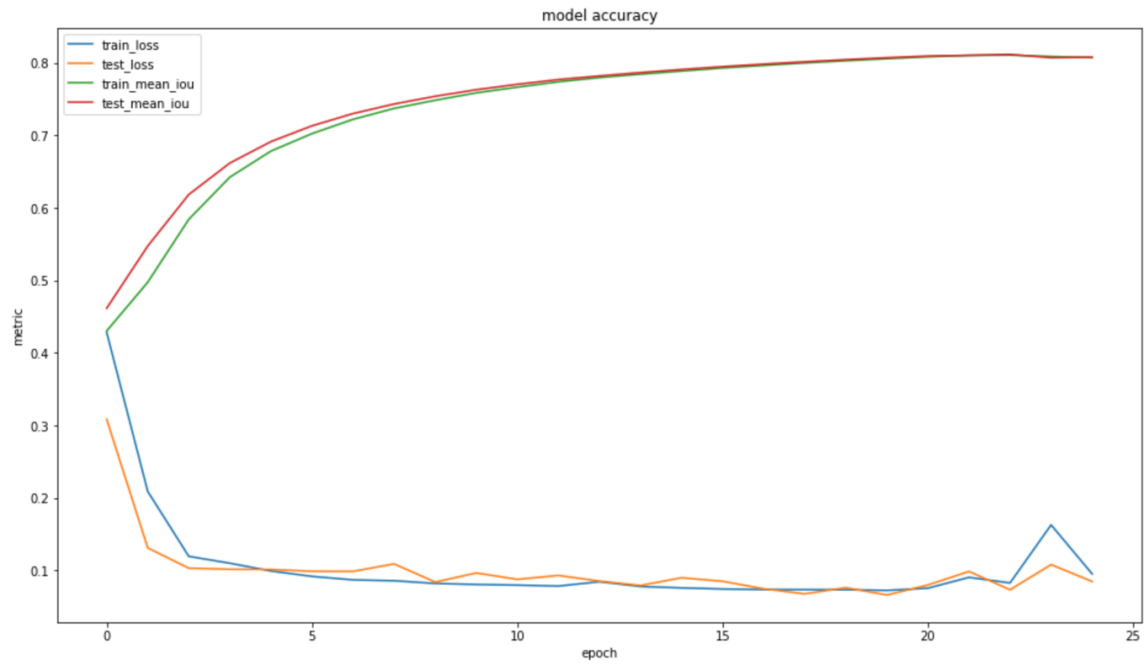
Benchmark Model



In the benchmark model, the IoU keeps on increasing at a steady rate but does not reach the values obtained with the Unet model. The lower IoU values in the benchmark values correlates well with the qualitative observation of more distinct nuclei (less clumps) in the Unet model. The Unet also does not take out the center of the nuclei, hence improving the IoU metric.

We can see that the Unet model starts with a similat IoU (as in the benchmark model) which keeps decreasing steadily in the first 3 epochs. The decrease rate slows down from then on. The mean IoU is a convex function (i.e. the rate of increase decreases over time). It reaches a plateau starting from epoch 19. We could have stopped training the model at epoch 19, hence saving training time.

Unet model (original data)



Summary

In this project I built a convolutional neural networks to segregate nuclei in images. This topic is of particular interest becausetThe automatic recognition of cells' nuclei can speed up the drug discovery and testing process, hence shortening the time period needed to find new drugs.

After resizing, graying and normalizing images, I built 2 neural networks, the Unet and a benchmark. The benchmark architecture is made up of convolutional layers and a dropout layers. I also augmented the data then conducted the following trainings:

- Benchmark model with original preprocessed data
- Unet model with original preprocessed data
- Unet model with the augmented data

I then compared the mean IoU yielded by all three networks. The Unet benefits from augmented data (3.7% increase in the validation mean IoU). The Unet outperforms the benchmark architecture when trained with the original data (34% increase in the validation mean IoU).

Reflection

In this project I was interested in tackling a real-world challenge despite not having the masks of the test data. I wanted to (1) familiarize myself with deconvolutional layers and Unet, (2) use data augmentation, and (3) experiment with bespoke metrics and run-length encoding. In the AIND and MLND, the data had always been prepared for the student and we were asked to create relatively simple neural networks. In this capstone project, I had to create an end-to-end solution, without the support of instructors.

Learning about the implementation of mean_iou, run-length encoding, data augmentation and the Unet architecture were the most interesting and the most difficult parts of the project. I used data generators for the very first time in this project and I had to pay attention to applying the same seed to both the generators on the training and test data. Understanding the convention used in the competition took some time. While the Unet clearly provides notable improvements over the benchmark, I was expecting a better result. The less than perfect result highlights the importance of pre-processing the data and augmenting it.

By generating an end-to-end solution, I understood the importance of “disciplined” coding (i.e. the appropriate use of comments, and helper functions) to make the code easier to read and amend.

Improvement

The model could certainly be improved in order to ensure a clear segregation of nuclei. I would consider the following methods:

Data pre-processing

- Increase the variety of shapes and size on the dataset
- Increase the size of the dataset
- Use an edge enhancement filter such as:
    - Canny edge detection from open cv to enhance edges
    - Pillow - the Python Image Processing Library
    - The issue with an edge enhancement filter is that it would require changing the masks (instead of a region, the masks will contain closed shapes).

Model implementation

- The use of a fixed split of the training data into a training dataset and a validation dataset could be improved. Ideally, I would like to apply K-folds cross validation split where I split the data into k different subsets. One subset is used for testing purposes while the other (k-1) subsetsa are used from training.
- Use a DenseNet (as presented in the following research paper: https://arxiv.org/pdf/1608.06993.pdf). According to Huang, Liu, van der Maaten and Weinberger [3], DenseNets can reduce the vanishing-gradient issue; DenseNets re-use features and reduce the number of parameters. This architecture requires less computational power while achieving a higher performance that other architectures. The code is available at https://github.com/liuzhuang13/DenseNet
- Use clustering in order to zoom in on the nuclei. This youtube video was shared with me by the last reviewer: https://www.youtube.com/watch?v=bZmJvmxfH6I&t=4679s

I would like to implement one of the solutions above with augmented data (obtained through a data generator) in the near future.

Research

[1] Mina Koshdeli, Bahram Parvi, Deep Learning Models Delineates Multiple Nuclear Phenotypes in H&E Stained Histology Sections. Submitted on 13 Feb 2018 (v1), last revised 14 Feb 2018, https://arxiv.org/abs/1802.04427

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, 18th May 2005, https://arxiv.org/pdf/1505.04597.pdf

[3] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, Densely Connected Convolutional Networks, 28th January 2018, https://arxiv.org/pdf/1608.06993.pdf