Assignment 2 - Machine Learning

Gianpiero Cea 1425458

March 14, 2018

Abstract

For this assignment we participated in the 2018 Data Science Bowl. The goal was to train an algorithm to automatically recognize the nuclei in various pictures of different kind of cells to help specialists reduce the time wasted on identifying the nuclei. The best score we obtained on Kaggle LB was of 0.305 when using a purely image analysis approach. The U-Net didn't score much since it perhaps required more sophisticated feature engineering and data augmentation.

1 Initial Data Exploration

Just from loading a few of the pictures in the training and test sets it seemed quite evident that the picture are quite sparsely distributed in terms of colors ,size and other parameters. Indeed the fact that the images distribution was multi-modal was explicitly stated in the problem specification and indeed the real challenge here is to be able to recognize nuclei over a very varied dataset of pictures. To explore more quantitatively this distribution we follow a similar initial analysis as in [1].
By a simple statistical analysis of the width, height and area we get the picture on the right. Also it seems like the R,G,B channels are each pairwise distributed bimodally. When considering mean intensity, this bimodality can be explained by the presence in the dataset of two distinct kind of pictures: brightfield and fluorescence's one. Numerically the sizes of the pictures have the following distributions: (256, 256) 334, (256, 320) 112, (520, 696) 92, (360, 360) 91, (1024, 1024) 16, (512, 640) 13, (603, 1272) 6,(260, 347) 5, (1040, 1388) 1. There were also pictures with really small mask for a singular cells: this is to be considered when downscaling pictures (for example to feed it to the MLP) as these mask would basically become invisible. Finally it was noted on various Kaggle discussions that some picture had clear mistakes with the manual annotations. In our code we ignored this and worked with the whole dataset since the percentage of errors seemed negligible. This analysis was inspired by [2].

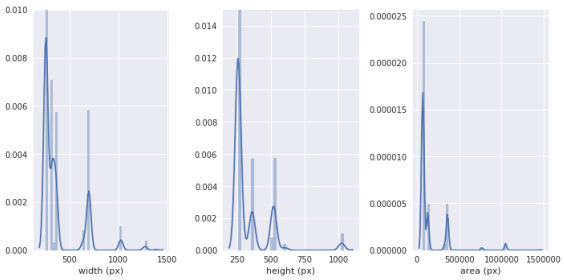

Figure 1: Distibution of width,heigth area

2 Feature Engineering

2.1 Histogram of Gradients

This technique consists of computing a particular kind of histogram for the picture by convolving the picture with filters that measure an approximate local gradient, for example one of the most used operator is the sobel operator that is capable of measuring a change in both the x and y dimension and

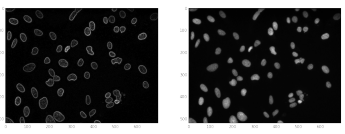

Figure 2: A comparison of the histogram of gradients and the original gray scale picture

is in general really good for example for edge detection. We used both skimage and OpenCV but the latter, using sobel, gave us a better solution so we used this. Overall the technique did not prove to be really good for segmentation even when we tried some morphological filters to try and "fill the holes" inside the boundaries detected by HoG.

2.2 Watershed segmentation

Watershed is in general a really good technique to do image segmentation since it tries to separate overlapping features and this a priori seems quite good for our problem. Out of the box it doesn't seem to perform really well on the cells pictures, possibly because the picture are too blurred: just from looking at the example on the right it is clear the the watershed algorithm tends to further subdivide masks that should be considered as one. The algorithm works by using the intensity of darkness as a distance metric, as we can see in the second picture on the right. Then we can pretend to place a water source at each maxima and "flood" the picture: the watershed lines gives our segmentation. Watershed segmentation was mentioned, together with other methods like K-Means clustering and Active contour as some of the best methods used for image segmentation in the following paper: [3].

Figure 3: Comparison of the picture,greyscale distance and watershed

2.3 Data Augmentation

Since we are expecting our test set to possibly be quite different from the train set, we think that adding more variance by generating new pictures by means of simple image transformations would be good. In particular it would make sense to colorize the pictures according to the most common clusters of colors in the picture and to somehow morph some of the pictures to take in account of the different sizes of cells. The effectiveness of data augmentation, in particular elastic modifications of the training data it is been well documented in the literature for example in [4] [5]. We took inspiration for the code from CITE: https://www.kaggle.com/shenmbsw/data-augmentation-and-tensorflow-u-net In our code we only implemented a basic data augmentation consisting of some random rotations of the pictures since they were quite easy to implement.

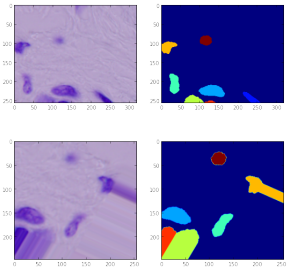

Figure 4: Distribution of width,height area

2.4 A model that does not use Neural Networks

As a part of the assignment we had to submit to the competition a model that did not use any neural networks. Indeed I used an algorithm that was completely based on pure image analysis and didn't do any learning on the train data. Following the kernel `https://www.kaggle.com/keegil/keras-u-net-starter-lb-0-277` I used otsu thresholding to separate the nuclei from the background and then used ndimage.label to generate the masks. This scored 0.256. I also then used a K-means clustering algorithm that got 0.305.

3 Artificial Networks and Deep Learning

Now we can assume we have different methods of feature engineering of the pictures. We know look if we can train some neural networks to automatically perform the segmentation task. As we have seen in the initial data exploration phase all image range through a very different array of sizes. Most likely it would be easier for us to either upscale or downscale all the pictures into a manageable size. Of course there are some sophisticated CNN models that allow for different kind of sizes for the train set but in this case we preferred to simply transform all the pictures into 128×128 pictures. The code to do this was taken from Keras U-Net starter - LB 0.277 [6] CITE.

3.1 Multilayer Perceptron

A multilayer perceptron is one of the first machine learning algorithm that gave popularity to the field. This is a feedforward network that takes a vector as an input and is passed to various hidden layers via activation function untils it reaches the output layer. The main characteristic of a MLP is that it is a highly parametric model that is differentiable. This is of key importance since it means that we can optimize it by means of a backpropagation algorithm We can quickly summarize a MLP as an input layer, an hidden layer and a final output layer and all these layers are fully connected with an activation function for each node. This can be see from the following diagram:

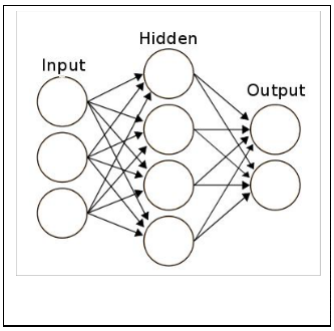

Figure 5: A general schematic for the Neural Network

I decided to use a very simple MLPClassifier from sklearn. Before doing this I wrote code to transform the various pictures into a vector form: this means flattening the picture array into a single vector with one dimension equal to 1. I tried a somewhat systematic grid search to try to fine tune all the hyperparameters but in general it seems like the performance Is really bad in general so I just used a standard MLPCLassifier. I think this is related to the fact that a lot of mask would just be very sparse and so it is difficult for the MLP to converge to a non trivial mask. This phenomenon was made evident from the fact that in most cases, If I tried to train the MLP on the full training data, I would just get the model that gave a trivial mask for all pictures while I could get a non trivial model by restricting the sample of training data. I have used three layers of size (20,20,10). The score is so bad that on the public leaderboard for Kaggle we only reach a score of 0.000. This was to be expected since when trying the network on the train set although we get some non trivial mask they don't seem to recover the actual mask so we think this method is not really apt since perhaps we miss too much information when we flatten the picture and so a Convolution Neural Network is required. It should also be noticed that by default MLPClassifier tries to minimise the log-loss and this is in no way related to the IoU metric which was of interest for the competition.

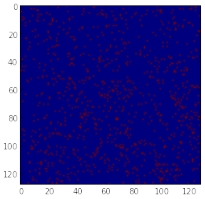

Figure 6: The mask obtained by our MLPClassifier when giving as a input a picture from the train data

3.2 Multilayer Perceptron with Feature engineering

Looking how bad the score was on the public leaderbord for the naive MLP taking as input the raw pixel of the picture flattened out, it is not surprising to see that non of the three methods mentioned above seemed to have done really well in terms of the score. I submitted my MLP applied to the pictures with the HoG and as expected this still had a score of 0.000. Perhaps I should have tried using some skimage morphological effect to fill the holes but it was already noticed that HoG was quite poor as a pure algorithm to do segmentation without training so I didn't investigate this further. I didn't submit the other two scores because just from testing them on my local machine I saw they would not have done much better. In particular we noticed how watershed seemd to have the effect of increasing the number of masks, so this would have made our model even more sparse. Data augmenting with rotation and color changes is more reasonable but instead we preferred it to use it together with the U-Net.

3.3 Convolutional Neural Network

REMARK: The version of keras was out of date. The codes were all tested on Kaggle notebooks with the most up to date version of the various packages.

A convolutional network is similar to a MLP in that it is also a feedforward network and it is also differentiable so we can use back-propagation to optimize the weigths. The main difference with convolutional neural network is that we don't loose anymore the structure of the picture since in various steps we can "convolve" the whole picture with much smaller filters.We also have some pooling layers that are a way of summarizing and reducing the data coming from various convolutional levels. In most cases it is impractical to use a fully connected network so this also distinguishes a CNN from MLP. Intuitively we can understand why a convolutional neural network would do better in image classification and segmentation tasks since it can be considered as an abstract version of what happens with retinal neurons, where higher and higher order details are learned from the net. From Kaggle it seems like a very popular structure for image segmentation in medical images is a U-Net. This should be the state of the art approach for medical image segmentation and it was firstly presented in the following paper: [**?**] U-Net: Convolutional Networks for Biomedical Image Segmentation. This CNN take its name from the U-Shape that the structure has. The reason why the U-Net is so successful is that the first "contracting" part of the U is really good at context recognition while the second symmetric "expanding" path is good for precise localisation of the nuclei. Going into the details of a the U-Net, in the way that has been described in the paper [**?**], on the descending path we have two consecutive applications of 3×3 convolution (each followed by a ReLU) followed by a downsampling by a 2x2 max pooling for stride 2. This means we downsample by only taking the maximum value over non-overlapping 2x2 convolutional filters. Max-pooling helps having simpler models that are less prone to overfitting by having less variance and also we can get more information on low level features from doing this. The number of feature channels

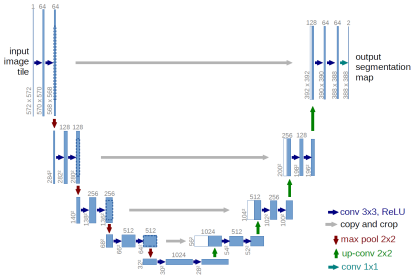

Figure 7: The U-Net structure

is double at every down sample step. A single step in the upscaling part of the U-Net consists of an upsampling, followed by an up-convolution with a 2x2 filter that halves the number of features. Then the result is concatenated with the corresponding result from the contracting path. This is a key step in the architecture since it helps keeping both information about low level context and localization. Finally we perform two consecutive 3x3 convolutions, each followed by a ReLU. To implement our convolutional neural network we used Keras with a tensorflow backend. The version installed in labs weren't up to date so they need to be updated before the code can be run. We take inspiration from [4] and [6] by creating a structure that goes up and down, where at each step we apply two convolutions followed by a max pooling. By running the net on the training data without data augmentation for only 10 epochs we could get a score on the Kaggle public leader board of 0.240 while running it for 100 epochs we increased the score to 0.263.

4 Progress graph and Final considerations

Submission	Description	Score
1	Threshold without watershed	0,2560
2	Threshold with watershed	0,2520
3	k-means clustering, k=3	0,2970
4	k-means clustering with watershed, k=3	0,2920
5	MLP	0,0000
6	U-Net, 128x128 image, DICE loss	0,2400
7	MLP trained on just 50 pics, layers =(20,10,20,10)	0,0000
8	Otsu thresholding+ morph opening	0,2560
9	Otsu thresholding+ morph opening+13 mask	0,2560
10	U-Net, 128x128 image, DICE loss, 100 epochs	0,2630
11	17 mask threshold	0,3050
12	U-Net,128x128 image, DICE loss, 100 epochs+imgaug	0,0000

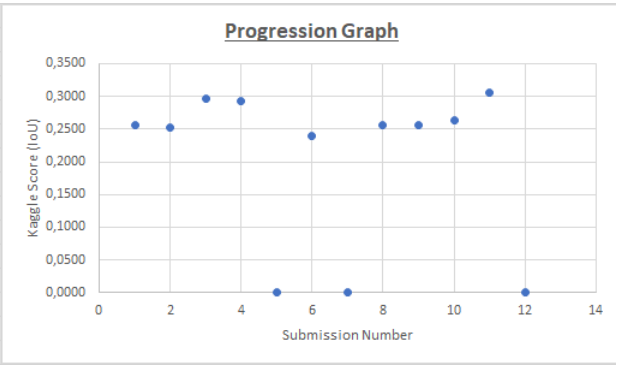

Figure 8: The progression graph of the submission on kaggle

As we can see in general our methods seemed to be fairly average and we weren't really able to push the IoU score in the Kaggle LB above the 0.305 mark. More surprisingly this result was not coming from an algorithm trained on the dataset but simply from a purely image analytic method, in this case k-means clustering with k=3. The CNN seemed promising but perhaps we should have tried playing more with some hyper parameters like batch sizes and sizes of the various convectional layers. In particular it seemed from working on this problem that the actual challenge was clearly identifying the borders for the mask of two overlapping cells. So a future approach to take could be to try implementing three different U-Nets, one to find nuclei, another one for background and another for borders. From the U-Net papers it seemed really important as well to perform some sophisticated data augmentation to significantly improve. In my 12th submission I tried to slightly modify my CNN by trying to use a basic data augmentation using the package `imgaug` Unfortunately I think the kind of modifications I applied must have deteriorated too much the dataset since after running for 3 hours I got a score of 0.000. I also whished to investigate other more sophisticated means of segmentation , for example active contour
It could have also been interesting to have done a transfer learning, perhaps layering an already trained u-net with some data augmented training data or even to try some ensemble methods, for example by taking union or intersection of the masks coming from different models.

Bibliography

1 https://www.kaggle.com/jerrythomas/exploratory-analysis

2 https://www.kaggle.com/jackvial/exploratory-data-analysis-dsb-2018

3 Irshad, Humayun, et al. "Methods for nuclei detection, segmentation, and classification in digital histopathology: a review—current status and future potential." IEEE reviews in biomedical engineering 7 (2014): 97-114.

4 Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

5 Dosovitskiy, Alexey, et al. "Discriminative unsupervised feature learning with convolutional neural networks." Advances in Neural Information Processing Systems. 2014.

6 https://www.kaggle.com/keegil/keras-u-net-starter-lb-0-277