CS342 Machine Learning: Assignment 2 Kaggle Competition: 2018 Data Science Bowl

Joe Tilley, MMath 4th year, `j.tilley@warwick.ac.uk`, 1403899

March 13, 2018

Abstract

Can deep feedforward networks be used to identify nuclei across varied conditions? Identifying nuclei allows researchers to identify each individual cell in a sample, thus allowing them to understand the underlying biological processes at work. This paper presents an analysis of findings for the Kaggle Competition: Data Science Bowl 2018. Multi Layer Perceptrons and Convolutional Neural Network models are employed to this problem, and we explore the challenges and successes of each.

Remark: All of the programmes attached to this report work on Kaggle's Kernel Notebooks, using the most recent version of Kaggle with TensorFlow backend.

1 Data Exploration, Feature Engineering, and Segmentation

1.1 Data Exploration

Exploring the images, we see that most are grayscale with black background and nuclei with varying white intensities. However, some images have white backgrounds with black nuclei and to follow the consistency of white nuclei on black background, we allow the option to invert colours in our models, as this often leads to improved learning. Moreover, some images are colour, often with pink and purple colours, which we explore further.

We analyse the colours of the entire dataset by performing an sns.pairplot analysis, as seen in Figure 1. On average the red, green and blue channels have similar intensities for all images (with some skewness towards greater intensity of blue, which could be exploited in feature engineering). The training set has a similar colour distribution to that of the test set, which means we do not need to concern ourselves with differences in training and test data colours.

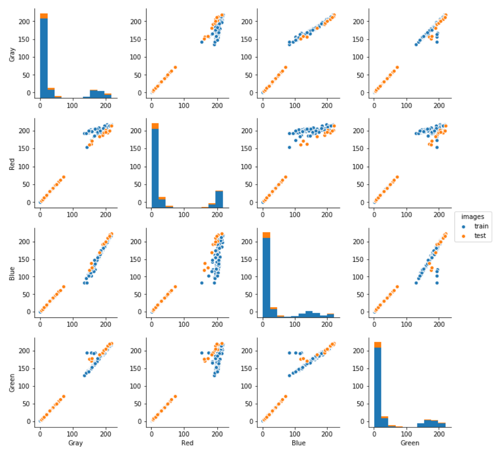


Figure 1: Pair plot of the average intensity of red, blue, green and gray across the dataset [3].

The distribution of image sizes and their respective frequencies are as follows - (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. When training our models, we should take this into consideration when deciding what dimensions to rescale our images to, so as to prevent the least information lost.

Other remarks to consider:

- Some of the masks have nuclei with holes and gaps in, which we should consider to be intentional.

- There are some labels/masks with nuclei smaller than 10 pixels large, so simply discarding nuclei predicted smaller than this is simply a quick technique to help generally rather than a good idea (as done in Bailey's kernel [4]).
- There are some mistakes in the dataset, and these are listed in [6].

1.2 Feature Engineering and Segmentation

1.2.1 Watershed Segmentation

Watershed segmentation is a technique used to separate overlapping objects. This is a particularly useful feature engineering technique for our task, as many nuclei overlap in predicted masks, which need segmenting. We implement this technique using Meyer's algorithm. This works by placing a water source in each local minima in the relief (where the minima is determined as the maxima of the distance to the background) to flood the entire relief from sources, and segment at watershed lines. An alternative (and, in some cases, more natural) idea would be to use the intensity of darkness in the grayscale images as our 'distance' function (so local minima become darkest points). However, due to the complexity of the images and their colours, this often produced unsatisfactory results, as in Figure 2.

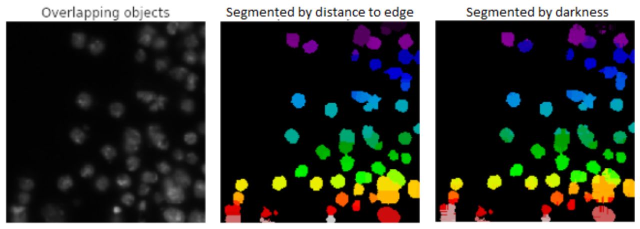


Figure 2: The original image (left), the segmented labels using watershed by distance to the background (centre), and the segmented labels using watershed by darkness (right).

1.2.2 Histogram of Orientated Gradients

Histogram of Orientated Gradients (HoG) is an object recognition technique, where the distribution of directions of gradients are used as feature descriptors. The directions are specified beforehand, though commonly x and y directions are used. This technique is applied through the use of a Sobel filter over small regions of the image. The gradients of an image are particularly important, as the magnitude of the gradients is large around edges where the colour intensity abruptly changes. These edges will provide valuable information in our segmentation problem, as they will often define the outline of the nuclei, thus making them easier to identify. Figure 3 displays the HoG of an image, showing the magnitude of the x and y gradient vectors. The method has been implemented using the Sobel method in the OpenCV library. This technique is not very effective for our problem; we cannot even apply a basic thresholding method (c.f. 1.3) on HoG images and 'filling the holes' of these labels, since, by the remarks made in 1.1 Data Exploration, some masks intentionally have holes in.

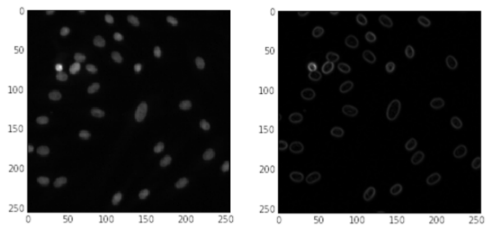


Figure 3: The original image (left) and its histogram of gradients (right).

1.2.3 k-means Clustering with Colours and Image Channel Scaling

We use k-means clustering to find the most common colours in an image (typically $k = 2$ for grayscale images, though $k = 3$ or more could be used for more complex coloured images, as seen in Figure 4). The colour with the largest cluster is typically the background, and this can be dropped from our label, and keeping the other colour clusters in our label. This feature

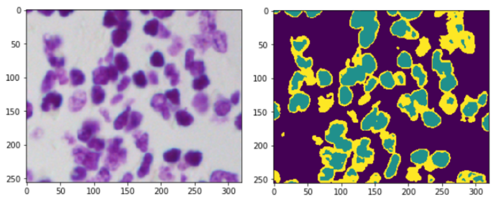


Figure 4: The original image (left) and 3-means clustered labelling by colour (right).

engineering technique is particularly applicable in our project, since the different colours in our images are known to be due to specific objects, i.e. the various colours are due to the nuclei, background, and other parts of the cell.

Furthermore, we see that some images have a high contrast, which are easier to identify nuclei on than images with a low contrast. Thus, to make full use of image contrasts, we scale each colour channel on each image to use the space between 0 and 255 optimally through the use of linear scaling. Figure 5 gives an example of this improved contrast.

As a potential extension, we could manipulate the non-background colour cluster channels determined in k-means clustering by rescaling their intensity to more strongly display these colours, instead of just rescaling all the RGB-image channels as we have done, thus making the nuclei easier to identify (this is also known as *colour deconvolution*, a method famously used for cell feature separation [1]).

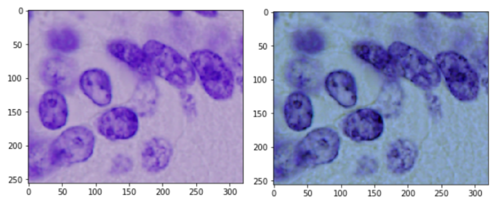


Figure 5: The original image (left) and the channel scaled image (right).

1.3 Applying non-ANN model to the competition

As a first submission to the competition, we follow Bailey's model [4], which converts the images to grayscale and uses a threshold value (via Otsu's method) to determine which pixels of the image are background and which are nuclei (a.k.a *image thresholding*). This method achieved an IOU score of 0.256. We also apply each of the feature engineering techniques on top of this method, with varying success. A successful attempt used 3-means clustering, dropping the most common colour cluster (i.e. the background) and using the other colours as a mask, achieving an IOU score of 0.301. An unsuccessful attempt used watershed segmentation, achieving a score of 0.064. This is most likely due to complexity of the overlapping nuclei in the images of the dataset, for which watershed assumes is far simpler.

2 Machine Learning Models: Artificial Neural Networks and Deep Learning

2.1 Multi-Layer Perceptrons

A *Multi-Layer Perceptron* (MLP) is a form of a feedforward network. The MLP has an input of a feature vector, which is fed forward to various hidden layers of a nodes, where the inputs are passed through an activation function before being fed to the next layer. The final layer is the output layer, which is the model's prediction. Optimal weights and bias parameters are obtain used a backpropogation algorithm with respect to a predefined loss function. An example of the network architecture of an MLP is described in Figure 6.

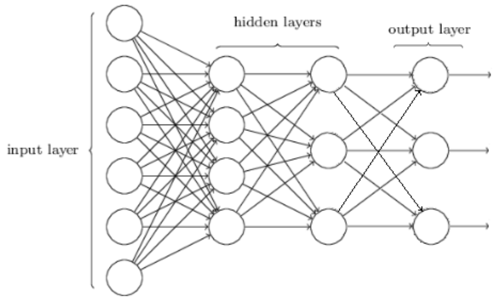


Figure 6: An example structure of an MLP.

We apply an MLP to our problem by taking feature vectors of the RGB-values of every pixel and an output layer giving the prediction of whether each pixel should be included in the mask (can be binary or probabilistic). Using this model, we lose the relationship between neighbouring pixels, as MLPs do not preserve the topological grid structure. This loss of information will undoubtedly lead to a worse model than a CNN, which preserves this information. We implement our MLP model through Sklearn's MLPClassifier method. The MLPClassifier uses log-loss function with which we minimise over, and we cannot modify. This will not provide the fastest rate of increase of IoU score (on which we are scored) as learning occurs, since IoU and log-loss do no not any close relationship.

As remarked in 1.1 Data exploration, we must rescale all of our images to the same size, so that they can be taken on by the MLP. Our choice of rescaling dimension is an important hyper parameter, which we discuss below.

2.1.1 Initial Observations

An immediate observation was that, when training on the entire training dataset and rescaling images to 128×128, almost no successful learning was achieved to produce an output remotely close to the desired masks, and in many cases producing noise, even on training image/masks it had allegedly learned (see Figure 7). The explanation for this was that the extreme sizes of the dataset and the neural network (more specifically, the size of the input layer due to the large images) led to an enormous number of parameters which the MLP struggled to minimise loss function for.

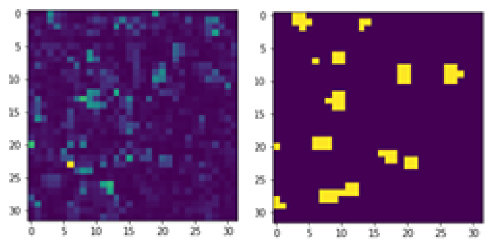


Figure 7: The MLP's probabilistic mask prediction (left) and the real mask (right).

Practising on a far smaller dataset (i.e. order of magnitude 10) and using smaller images (i.e 32×32) led to greater success; the MLP was able to successfully predict masks for training images/masks it had learned. However, this performed poorly on valdiation sets, as the MLP was massively overfitting.

2.1.2 Improving Learning and Feature Engineering

We implement a grid search combined with cross validation (via Sklearn's GridSearchCV) to find ideal values for the hyper parameters of our MLP, such as hidden layer structure, activation functions, method of minimisation (i.e. gradient descent), regularisation constant, batch size, learning rate decay type, initial learning rate, etc.

We can artificially expand the dataset by applying HoG and image channel scaling to every image to increase the size of our training data, thus helping prevent overfitting. Moreover, we could perform many data augmentation techniques to achieve the same result (c.f. 3 Progress Graph and Discussion). However, increasing the training dataset would only lead to an increased number of parameters in our loss function to minimise over, which we know the MLP struggled with. Furthermore, we need not concern ourselves with preventing overfitting on the entire training dataset, since we struggled to get any learning to occur in the first place, let alone overfit to it! Instead, we find greater success converting the images to grayscale, due to the fact we cut the number of parameters down by a factor of three.

For non-augmentation feature engineering techniques, we apply watershed segmentation to our model's predicted masks to segment the predicted overlapping nuclei. However, the MLP was so inaccurate and imprecise with its predictions that it almost never predicted masks resembling overlapping nuclei, thus rendering the technique useless; it is clearly too intricate of a technique to even be considered in this case, before we can even discuss its effectiveness (c.f. section 1.3).

With a basic MLP, we scored an IoU of 0.000 on a Kaggle submission. It was clear from this that it would be a waste of time to train the MLP on the entire training dataset with all our feature engineering techniques to get an 'optimal' model. Most likely, our IoU score would never have exceeded 0.15 with MLPs.

2.2 Convolutional Neural Network

Convolutional Neural Networks (CNN) are a type of feedforward network, similar to MLP. Like the MLP, the CNN relies on backpropogation to update weight and bias parameters. However, unlike the MLP, a

CNN takes advantage of the grid-like topological structure of images, allowing for more advanced learning through methods of convolving with filters to generate activation maps, typically alongside the user of pool layers, which progressively reduce the spatial size, thus reducing the amount of parameters and computation required by the network.

Following Kjetil Åmdal-Sævik's implementation [5] of the network architecture described in [2], we use a U-Net CNN structure, as shown in Figure 6. The U-Net is highy applicable to our problem, as in a very similar challenge for segmentation of neuronal structures in electron microscopic stacks, it outperformed the prior best method of a sliding-window convolutional network [2]. In our modified version of this, we use Keras with a TensorFlow backend to implement convolutional layers, which will exploit the preserved topological grid structure and relationships between nearby pixels, along with elu activation functions (advantageous over sigmoid functions as increasing the weighted input will never cause the neuron to saturate) sandwiched between dropout layers (which prevent overfitting by forcing robust learning in conjuction with other neurons). Additionally, we use pooling layers, as seen in the network architecture, involving both max-pooling and up-convolving (with which we then concatenate with previous layers).

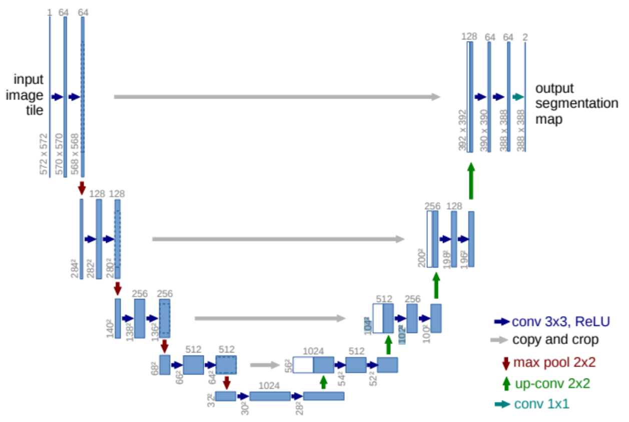


Figure 8: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations [2].

2.2.1 Training the CNN

Whilst Åmdal-Sævik trains using the Binary Cross-Entropy loss function, we use Mader's suggestion of the inverse dice function, based on Jaccard dice score coefficents, as our loss function, since we our scored on the IoU, which holds a close relationship with the dice index [4].

When training the model, it appears that downscaling the images to just 256×256 (c.f. 1.1 Data Exploration) gives a better performance than downscaling to 128 (unlike the initial method of scaling to 128 suggested by Åmdal-Sævik). In the original paper [2], it is stated that they do not wish to lose information by downscaling more than necessary, and hence downscale to just 256×256, which may be the cause of improved performance. Furthermore, following suggestions made by Raoul [3], upscaling images to 384×384 or 512×512 can lead to even greater performance. Again, this is most likely due to saving lost information by not downscaling, and no information is lost by upscaling small images. However, upscaling leads to more parameters in our loss function. Thus, we consider the upscaling dimension as a hyper-parameter to be optimised.

Like the MLP, we can use grid search cross validation to find optimal hyper parameters. However, due to the prominent use of U-Nets in this Kaggle challenge, we can use *transfer learning* of other kernels as a starting point for hyper parameters to optimise further, such as from [4].

We can apply our feature engineering techniques discussed in section 1 to the CNN:

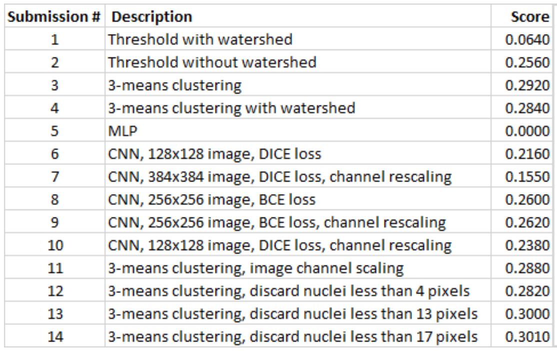
- We can augment all our training and test data images by applying image channel scaling and HoG, along with other data augmentation techniques (discussed in section 3).
- Instead of simply rounding the probabilistic outputs of our CNN to 0 and 1, we could apply our k-means clustering (or simply Otsu's method, since our predictions are 1-channel images and can be considered grayscale) and drop the largest cluster (i.e. the background) to get our mask prediction. This may improve IOU score, particularly in cases where the model has low confidence in pixels i.e. an

output around 0.5. However, the effects of this compared to rounding are negligible, as the threshold is typically 0.5.

- We do not apply our watershed segmentation technique, as this drastically reduced our IoU score in section 1.3, and applying this to our predicted mask certainly reduces our IoU score similarly.

3 Progression Graph & Discussion

- MLP's are clearly not an appropriate model for this task. This is most likely due to the importance of topological grid structure of our images and the relationships between neighbouring pixels, which is lost using MLP's. Moreover, the rate of learning was extremely low compared to CNN's.
- CNN's were effective models for this task, due to preservation of this topology, along with the fast rate of learning due to pooling layers. Other Kaggle users have achieved much higher scores with U-Net's, achieving scores of 0.4 upwards through the use of effective data augmentation, network architecture restructuring, hyper-parameter optimisation, and lengthy training times. The difficulty lies in finding these, as well as the effectiveness of CNN's once these have been found.
- More data augmentation techniques could have been explored in my scripts, thus artificially expanding the dataset, a known method to reduce overfitting, by cropping, flipping, rotating, jittering and modifying colours of images. However, we should use augmenting techniques appropriate to this task, such as generating artificial images of common colours found in 1.1 Data Exploration.
- We failed to find an effective way of segmenting overlapping nuclei, and this should be explored further. Watershed technique is proven to be ineffective due to its oversimplification for complex images. Instead, more sophisticated segmentation techniques are required, such as *active contour model*. Furthermore, we should explore more normalisation methods. For example, success has been found using *contrast limited adaptive histogram equalization* [7].
- Figure 9 describes our progression graph. Early success was found the k-means clustering, with struggling MLP and CNN models later. The final submissions attempted to boost early work.



Submission #	Description	Score
1	Threshold with watershed	0.0640
2	Threshold without watershed	0.2560
3	3-means clustering	0.2920
4	3-means clustering with watershed	0.2840
5	MLP	0.0000
6	CNN, 128x128 image, DICE loss	0.2160
7	CNN, 384x384 image, DICE loss, channel rescaling	0.1550
8	CNN, 256x256 image, BCE loss	0.2600
9	CNN, 256x256 image, BCE loss, channel rescaling	0.2620
10	CNN, 128x128 image, DICE loss, channel rescaling	0.2380
11	3-means clustering, image channel scaling	0.2880
12	3-means clustering, discard nuclei less than 4 pixels	0.2820
13	3-means clustering, discard nuclei less than 13 pixels	0.3000
14	3-means clustering, discard nuclei less than 17 pixels	0.3010

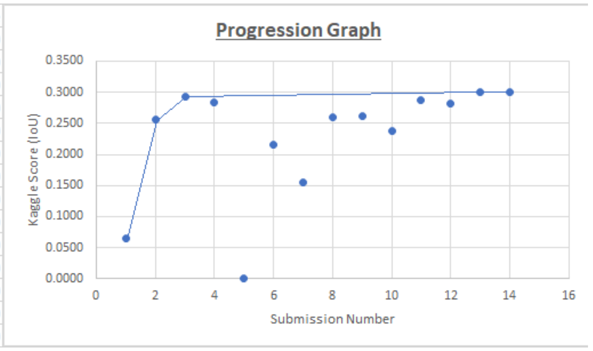


Figure 9: Progression graph and description of submissions.

References

[1] A. C. Ruifrok and D. A. Johnston. *Quantification of histochemical staining by color deconvolution.* Analytical and quantitative cytology and histology / the International Academy of Cytology [and] American Society of Cytology, vol. 23, no. 4, pp. 291-9, Aug. 2001. [2] Olaf Ronneberger, Philipp Fischer, Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* Medical Image Computing and Computer-Assisted Intervention (MICCAI), Springer, LNCS, Vol.9351: 234–241, 2015. [3] https://www.kaggle.com/raoulma/nuclei-dsb-2018-tensorflow-u-net-score-0-352. [4] https://www.kaggle.com/stkbailey/teaching-notebook-for-total-imaging-newbies. [5] https://www.kaggle.com/keegil/keras-u-net-starter-lb-0-277. [6] https://www.kaggle.com/c/data-science-bowl-2018/discussion/47770. [7] https://www.kaggle.com/kmader/normalizing-brightfield-stained-and-fluorescence/notebook.