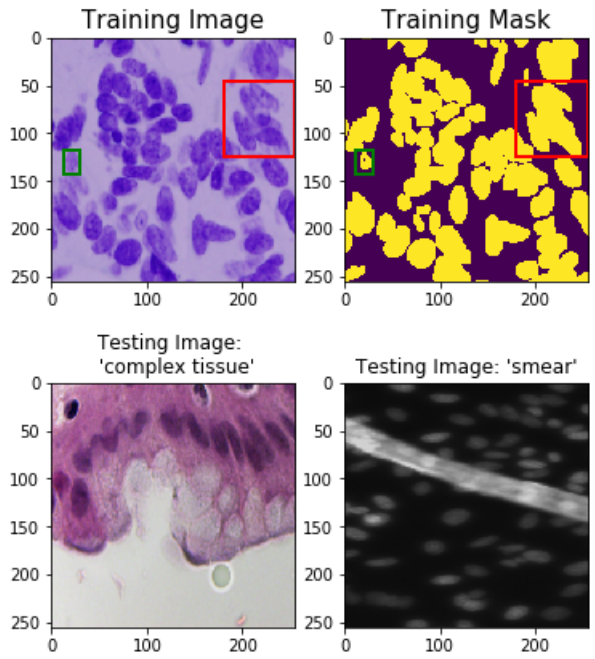
Machine Learning Assignment 2

Abstract

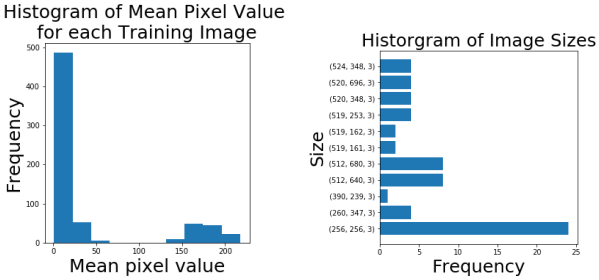
This report covers a nuclei segmentation task given by a Kaggle competition[1]. Watershed segmentation, Multilayer- perceptron, and convolutional neural network models are discussed and tested in this context. A top score of 0.337 (intersection over union) was recorded on Kaggle for a U-Net model.

1 Data Exploration, Feature Engineering and Segmentation

1.1 The Data, and its Quality



(a) Example images from the training set (top). Note the apparent merging of the ground truth masks (red box) and the missing mask pixels (green box). This low quality of target mask causes a supervised machine learning algorithm (using this mask) to learn to predict too few nuclei. Example images from the testing set (bottom) these are examples of Images that exhibited the 'complex tissue' structure (left) or 'smear' were empirically, poorly predicted due to little training examples.



(b) Size and pixel distributions for all the training images.

Figure 1

Immediately from figure 1 we see that there are several problems with the data set, which caused a learning algorithm to falter. Firstly the merging of masks in the training set (top of figure 1) caused the machine to learn to predict a single nucleus when many were actually present, resulting in lower accuracy. Secondly lost training set mask pixels caused the machine to learn to predict holes in masks, clearly in error. The lost mask pixels were (likely) a systematic error; even when 'fixed' these errors potentially remained within the set of true masks used by Kaggle for final evaluation. Images showing 'smears', similar to the testing image in the right of figure 1 b, also appeared in the testing set only, thus the machine could not learn to deal with such problems well on the training set alone, this empirically resulted in a large rectangular nucleus being predicted causing loss of accuracy. Similarly to this more complex images, such as the left image in figure 1 a, appeared little in the training set, again empirically these images are poorly predicted. Also of note were the distinct 'colour' groupings shown in figure 1 b, resultant from the two classes of images in the training set, stained (top left figure 1 a) and black and white (figure 1 a bottom right). The images also exhibited sizes in the range (256,256,3) to (1040,1388,3). Although the density of the image sizes was strongly concentrated on (256,256,3) which accounted for $\sim 50\%$ of the images. Many of these deficiencies were remedied (at least partially) by use of the feature engineering techniques discussed below. One notable exemption to this was the issue of overlapping and missing mask pixels, an attempt was made to improve mask separation by removing edges detected via an algorithm such as canny. This resulted in a IoU of ~ 0.2 between the original and 'corrected' masks, model performance was severely impacted by this fact.

1.1.1 Feature Engineering

Two main approaches were used for feature engineering: partitioning, and data/image augmentation.

Image augmentation is a feature engineering technique used to 'extend' an image data set by introducing a rich variety of transformed images from the original data set, for example rotations by e.g 90°/random angles, blurring using Gaussian noise, inverting colours etcetera. Here images were chosen randomly from the training data set and flipped over the horizontal/vertical axes with probability 0.5 each, then colour inversion was applied to all images. These choices are based on the discussion of augmentation in the context of medical image segmentation in [7] and implemented using [5]. Additionally to account for the 'smear' images shown in figure 1 a, images with rectangular-like over masks were added into the data set by use of the python imaging library (PIL), both the size and transparency of the rectangular masks was randomized. For the 'complex tissue' type images additional images were sourced from a new data set [9] (linked on the Kaggle external data thread). It was noted that image augmentation of the above forms produced the largest gains in performance of all feature engineering approaches used.

For the Multilayer-Perceptron (MLP) images were partitioned from shapes of $(256, 256)$ to sets of 256 images of size (16,16) each corresponding to one image. The partition size hyper-parameter was determined by a K-fold cross validation (K=5) over

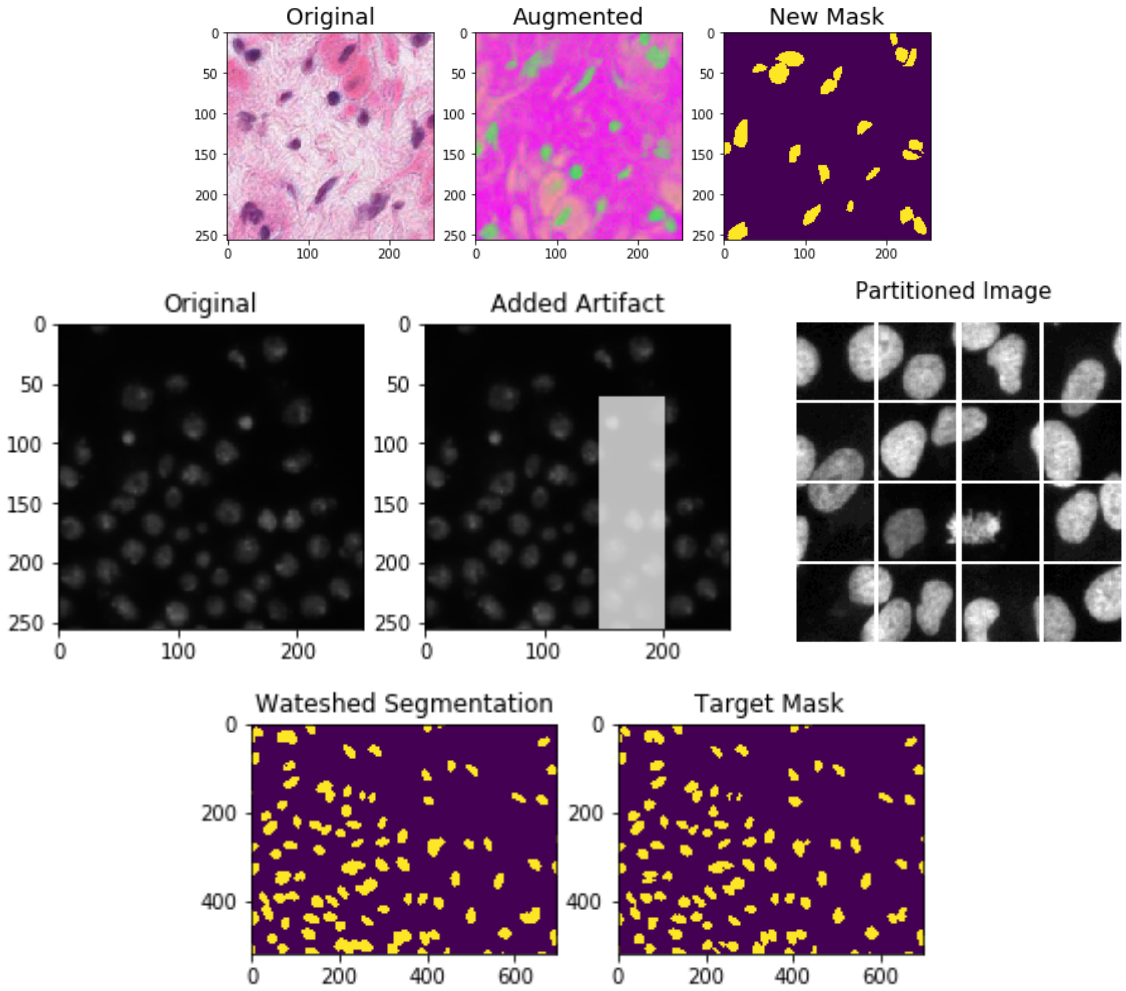


Figure 2: An example of image augmentation (top). The top left image is an example from the extra data set [9]. An example of adding in rectangular artifacts (middle). Second from bottom image partitioning into $(64, 64)$ for demonstration only. Finally an example of watershed segmentation.

the values $\{2^n : n \in \{0, 1, \ldots, 7, 8\}\}$. These new images were then passed as vectors of dimension $(256 = 16^2)$. The idea is to resemble the process of filtering in a convolutional layer to a small extent, in the hope that the MLP is able to retain some spatial information. See figure 3 for the result of the K-fold cross validation. Watershed segmentation

1.1.2 Watershed Segmentation

A simple model, without the use of artificial neural networks, was devised using the watershed transform. The intuitive idea behind the watershed transform is from geography where watershed lines are the dividing lines between domains of attraction of rain, that is boundaries of basins. Mathematically the basins are local minima and the watershed lines will correspond to object outlines [6].

The watershed algorithm was given edge magnitudes generated from a Scharr transform. Markers for the watershed transform were generated by thresholding the image, this thresholding parameter was found by maximizing the intersection over union (IoU) metric over the whole training set, this maximum was determined to be $1/7$. The watershed method achieved a score of 0.164 IoU on testing data, this was posted to Kaggle on 26/2/18. The reason for this low score was potentially due to high variance in predictions, images such as the middle left of figure 2 were predicted well (0.5-0.9 IoU), whilst images with coloured background or complex tissues structures etc performed extremely poorly ($\sim$ 0.01 IoU), often with the majority of the image being predicted as a nucleus.

2 Artificial Neural Networks

2.1 MLP Classifiers

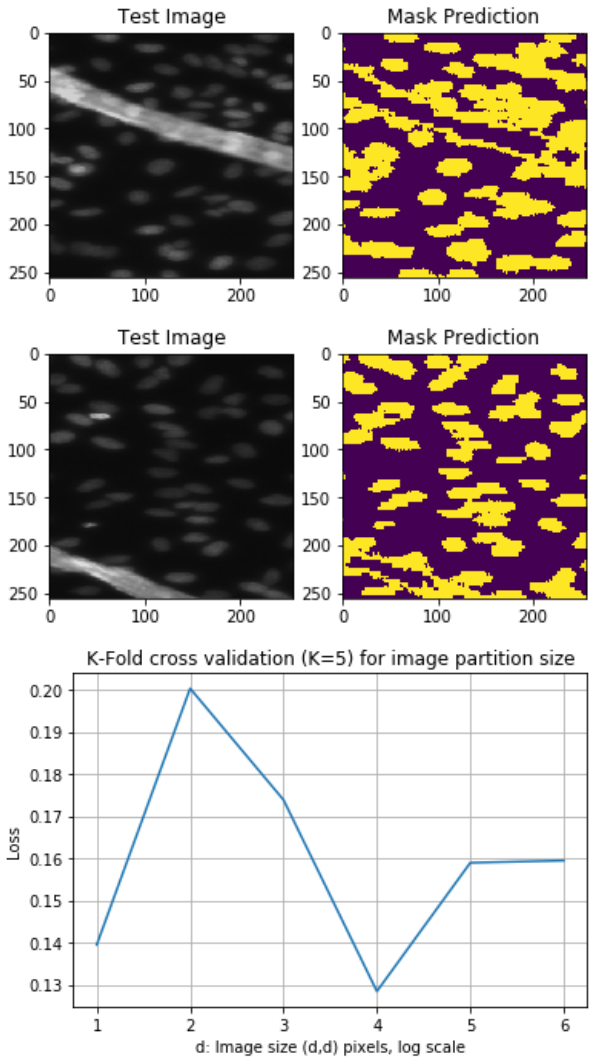


Figure 3: (Top) Example predictions from the MLP with no major data modifications. (Middle) example prediction of the MLP with partitioned and vectorised images. (Bottom) result of K-Fold cross validation on image partition size.

First an MLP classifier was trained with no modifications to the data except for gray-scaling each image and resizing all images to $(256, 256)$. The model architecture is simple, using one hidden layer and dropouts (probability 0.1) after the input and hidden layers. The main model consists of the three dense layers, these are full connected neural network layers. That is there are nodes corresponding to activations (dot products and kernels) and edges corresponding to weights between layers in an n-partite graph. Dropout layers work as a form of regularization by setting neurons to 0 at random, say with probability $1 - p = 0.1$. The activations, non-linearities to which the node values are passed, were Rectified Linear Unit (ReLu) in the input and hidden layers, and finishing with a sigmoid activation in the output layer.

The number of hidden layers, 1, was determined by training models with additional hidden layers on a subset of the images. The architecture used is similar to that of an auto-encoder network, that is the number of activations decreases from the input dimension down to a value and then climbs again. This style of architecture has seen success in general segmentation tasks in the form of encoder-decoder convolutional networks, SegNet [2] being a good example.

Trained on 50 epochs this simple model achieved an IoU of 0.085 on Kaggle (posted 9/3/18), an example of mask prediction can be seen in figure 3 (top image). Somewhat surprising is the models ability to deal with the large 'smear' artifact in this test image, especially since the model did not train on any similar images. Clearly the model struggles with separating nuclei which are close.

For the partitioned images and vectorised inputs the model parameters are decreased for each layer to match the new input dimension of (256), as apposed to $(256, 256)$. Just by modifying the network to take the partitioned images and vectorised an IoU of 0.096 was recored on Kaggle (posted 9/3/16). Most interesting was the use of image augmentation (460 new images) which was used for both the partitioned image network and the whole image network. Both these scores were lower on Kaggle, $0.071, 0.069$ respectively. This is perhaps because of what the MLP will perceive the image as, a set of high and low magnitude pixels, so for a colourful or 'busy' image such as the 'complex tissue' types, the MLP detects these regions as highly active and so predicts a mask. This is the major downfall of the MLP, as applied in this task, unlike the convolutional neural network the MLP cannot on its own learn the inter-relations between pixels.

2.2 Convolutional Neural Networks

Two convolutional neural networks (CNNs) were implemented, first a simple design based upon encoder-decoder networks such as SegNet [2], and a larger, well known network U-Net [7]. The former is designed for general image segmentation while the latter is specifically designed for biomedical image segmentation. For both CNN models the loss function used was the inverse mean IoU defined as simply 1−mean IoU.

2.2.1 Encoder-Decoder Architecture and Results

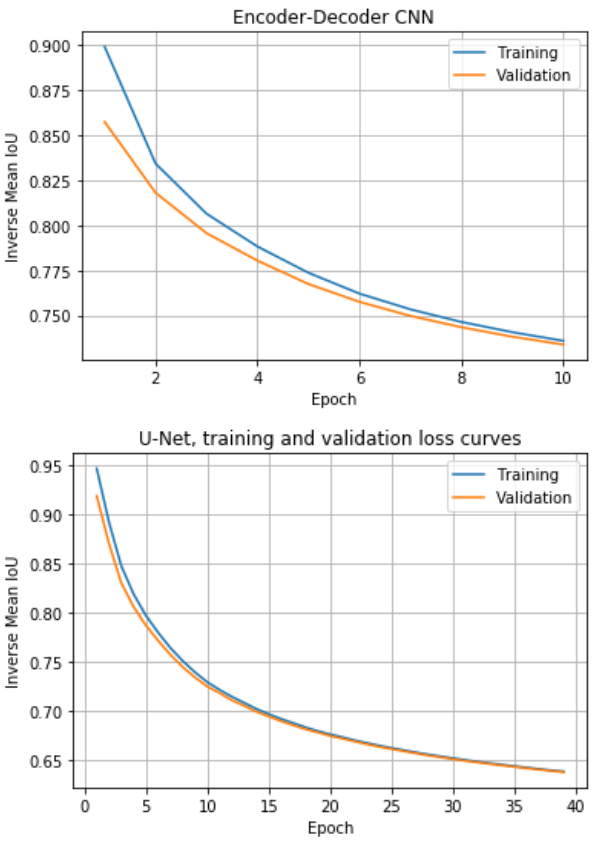


Figure 4: (Top) loss curves for the Encoder-Decoder network both with dropouts and batch normalization. The training and validation loss curve remain close indicating little over-fitting, i.e the dropout layers working. (Bottom) an analogous plot for the U-Net model.

The first network is formed of 4 sections, two encoder layers, a central layer, two decoder layers and a final output layer. The encoder layers consist of two pairs of convolutional layers each followed by a dropout, batch normalisation, Rectified linear unit (ReLu) activation, and ending with a $(2, 2)$ filter max pooling. The central layer is the same, but with no max pooling. The decoder layers again have no max pooling but also use a Deconvolution layer (transposed convolution layer) in place of the first convolutional layer and followers. The final output layer consists of a convolutional layer with one filter and a sigmoid activation to classify as either background or mask. The main difference with these models is the use of convolutional layers, these layers involve applying (usually) (d, d) dimensional *filters* which are convolved (passed over) the input tensor (here an image matrix of dimension (x,y,z)). These filters have weights associated with their d^2 entries, and by matrix multiplication an activation map/s is/are determined using the input tensor and filter/s. This process will infer a dimensionality of the activation map, but by *zero-padding*, that is enclosing the input tensor in n layers of 0s this dimensionality can be manipulated. Filters also have hyper-parameters such as stride, how many positions to move the filter for the next activation. Another important layer is the pooling layer (here max-pool layers are used), the basic concept is to apply a filter, usually of dimension $(2, 2)$ which computes the maximum, average, or other value of a block of 4 inputs and producing an activation with these values as entries, so a $(256, 256, 3)$ image would be transformed to a $(128, 128, 3)$ activation since the filter does not applies max or average over the depth.

Here dropouts were used after convolutional layers to reduce over-fitting by the model, as mentioned in the MLP section. Batch normalisations were also used before each activation, these may be interpreted as a form of preprocessing performed at every layer, the effect is to increase the robustness of a model to 'bad' weight initializations [4]. The top image of figure 4 shows an example of loss curves when the model includes dropouts/batch normalization.

Using raw pixel data the model achieved an IoU of 0.121 as measured on Kaggle. Using the image augmentation techniques and extra data (1034 total images), as discussed in the feature engineering section, an improved score of 0.175 IoU was recorded on Kaggle.

2.2.2 U-Net Architecture and Results

The architecture of the U-Net is similar in spirit to the encoder-decoder, we have a contraction path involving groups of convolutional layers and dropouts ending with a max-pooling layer for each group. As this down sampling progresses the number of activations for each convolutional layer is doubled up to the central layer where it matches the image dimensions (e.g 256). What follows is an up-sampling path using transpose convolutional layers halving the activations each time. Additionally concatenations are performed between up and down sampling groups. A final output layer is a convolutional layer with 1 activation passed through a sigmoid for prediction of the mask.

Using 1000 total images, including all forms of image augmentation and extra data as discussed in the feature engineering section a top score of 0.337 IoU was recorded on submission to Kaggle.

3 Discussion

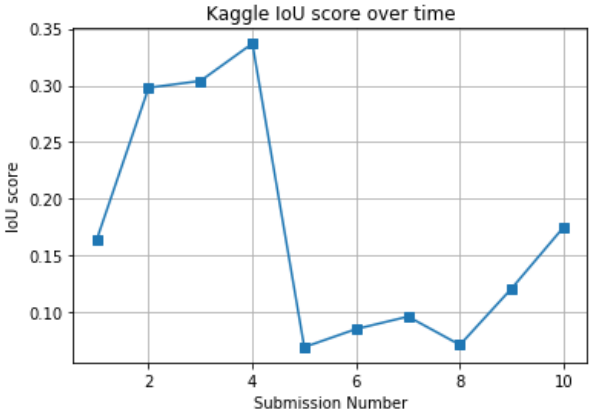


Figure 5: Graph of the progression of Kaggle IoU score (1.0 is a perfect score) during the project, refer below for dates.

1: 26 Feb. Watershed thresholding model.

2: 28 Feb. Began using CNN models starting with the U-Net and raw training data.

3: 28 Feb. The same U-Net model but with added image augmentation (330 new images).

4: 4 Mar. The same U-Net model but with rectangular mask augments, extra data (1000 total images).

5: 9 Mar. MLP model no partitioning and augmented images (1098 total images).

6: 9 Mar. Same MLP but with no image augmentation.

7: 9 Mar. MLP with partitioned images but no image augmentation.

8: 9 Mar. MLP with partitioned images and image augmentation (1130 total images).

9: 12 Mar. Encoder-Decoder model with raw images.

10: 12 Mar. Encoder-Decoder model with image augmentation and extra data (1034 total images).

Many future directions exist for this project. In particular exploring the effect of large amounts of image augmentation perhaps moving into having multiple thousands of augmented images added into the data set. Another direction would be to explore model combination, such as transfer learning from big pre-trained architectures such as VGG16, but also ensemble methods given that models such as the watershed model perform well on particular images in the training set (those mostly black with white nuclei). Finally to continue increasing the score a more detailed approach to feature extraction would be required and perhaps dimensionality reduction.

References

[1] *2018 Data Science Bowl: Find the nuclei in divergent images to advance medical discovery*. URL: https://www.kaggle.com/c/data-science-bowl-2018.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[3] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on.* Vol. 1. IEEE. 2005, pp. 886–893.

[4] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[5] Alexander Jung. *Imgaug*. URL: https://github.com/aleju/imgaug.

[6] Jos BTM Roerdink and Arnold Meijster. "The watershed transform: Definitions, algorithms and parallelization strategies". In: *Fundamenta informaticae* 41.1, 2 (2000), pp. 187–228.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[8] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[9] Sanuj Sharma. *Nuclei-net (Bachelor Thesis)*. URL: https://github.com/sanuj/nuclei-net.