Computing For Medicine: Project 4
Medical Image Analysis

Chris McIntosh*

November 29, 2016

1 Introduction

Cancer treatment has faced a major question over the past 20 years. Why do two patients with seemingly identical cancers receive the same treatment but have drastically different outcomes? Thanks to advances in genomics research and image analysis (often called radiomics in this context) we are now able to observe that even though both patients have tumours in the left lung, the tumours can be very different [1]. Research in genetics and imaging has created the concept of tumor phenotyping, which involves sub categorizing tumors in a particular site, e.g. Lung or Colon Cancer, into different types based on expected patient outcome, response to chemotherapy, or radiation therapy. Within this massive field of research, one particular area is examining the heterogeneity of tumour cells to better understand tumour stage, development, and structure [2].

In this project you will build a system to automatically detect Nuclei centres in histology images using data from [2]. Once the centres are detected, the cell nuclei can be classified into Epithelial, Inflammatory, Fibroblast, and Miscellaneous nuclei, and the heterogeneity of the tumour can be analyzed.

2 Tutorial

Begin by walking through image_processing_tutorial.py, and familiarizing yourself with the data, and some of the image processing techniques that will be used in the project.

2.1 Scoring the detector

In order to finish the tutorial you will need to fill in the score_detector function in project_helpers.py. You will need to write an algorithm to determine if a predicted nuclei centroid is a true positive (TP), or a false negative. A detection is a true positive if it is within a 12 pixel radius of an actual nuclei centre, otherwise it is a false positive. A false negative (FN) occurs when there is no proposed detection within a 12 pixel radius of an actual nuclei centre. The code calculates the number of false positives (FP) and positives (P) for you.

You will also need to keep track of the indexes of `detection_ground_truth` that are false negative (i.e. missed) in a variable `FNList`.

Calculate and return a tuple containing the precision, recall, F1 Measure, and `FNList`. The formulas are as follows: $precision = \frac{TP}{TP+FP}$, $recall = \frac{TP}{P}$, and $F1Measure = \frac{2TP}{2TP+FP+FN}$

Keep track of your results to compare them to the next section.

3 The Project Helpers

Begin working through nuclei_detection_tutorial.py. You will fill in functions in project_helpers.py to get the full tutorial working.

*[1] Department of Medical Imaging & Physics, Princess Margaret Cancer Centre, University Health Network (UHN), Toronto, ON, Canada

3.1 Basic detection

In this part you will compute image features to use for nuclei centroid detection by filling in the function calculate_features in project_helpers.py. The input will be an $M \times N \times 3$ color image.

Using the techniques in the image_processing_tutorial.py as a guide, calculate the following features:

1. The average grayscale intensity of a 3x3 window around each pixel, using the Haematoxylin component of the RGB image converted instead to a Haematoxylin-Eosin-DAB (HED) stain image. The conversion can be done using a built in function from skitit-image. Search the documentation and find the function.
2. The average red value of a 3x3 window around each pixel.
3. The average green value of a 3x3 window around each pixel.
4. The average blue value of a 3x3 window around each pixel.
5. The matched template response for each of the input templates.

Finally the code stacks all of the features into an $M \times N \times K$ array, where K is the number of features, and returns the feature array.

You can get creative and add new features by creating different filter arrays in place of F.

3.2 Adding some more features automatically

In the first part of the tutorial we manually specified a single template. You can experiment by manually specifying different templates and seeing how it impacts your result.

Now implement code to instead get a set of templates, one for each ground truth detection by filling in get_templates in project_helpers.py. Each template is a window of pixels (or an array) centred around a detection's x, y co-ordinate. Just make sure that your window doesn't go outside of the image boundary, since we don't know the intensity at those pixels.

Loop over the detections, and find each template. Store it in the 3rd dimension of a templates array so that we can use it later. Make sure you keep track of any unused slots in the templates array, and remove them so you don't have any blank templates.

Run the rest of the Part 2 and see how your accuracy improves.

3.3 Running a new image

So far we've only tested images that were also used in training, which is very biased and doesn't represent how our model will perform in the real world.

Start by visualizing the sample image. Can you spot the nuclei centres?

Fill in the code in nuclei_detection_tutorial.py to get the features for the new image, perform the prediction, and evaluate the result. Run this process for both the single template model trained in 3.1, and the multiple template model in 3.2. This image looks very different, so it is much harder to match using templates from the first image we looked at.

3.4 More data!

In order to improve results, the next step is to gather features from multiple images, thus training the system on a greater variety of nuclei shapes, sizes, and colors.

You will need to fill in get_features_for_image_set in project_helpers.py. This function will loop over images 1 through `img_count` in the database, extract the features, get the training samples, and stack all of the results together into two larger arrays using Numpy's `vstack` function. Make use of the other helper functions in project_helpers.py.

Get the features, perform the prediction, and evaluate the results for image img84, as you did in Part 3.3. Compare the results to 3.3.

4 Building the Project

In this part you will complete project.py. You now have a framework to train on multiple images, and test on a novel image. Extend your framework to train on the first 20 images, and test on the last 20 (i.e. 80-100). Compare your results to the simple models from Parts 3.1 and 3.2 tested for the same images. Test your code first on a few images, as the final run will take a while.

1. First get all of the templates
2. Get all of the features
3. Train the model
4. Fill in the validate_model to validate a model over a number of test images
5. Run validate_model for the three classifiers, and compare your results

References

[1] H. J. Aerts, E. R. Velazquez, R. T. Leijenaar, C. Parmar, P. Grossmann, S. Carvalho, J. Bussink, R. Monshouwer, B. Haibe-Kains, D. Rietveld, *et al.*, "Decoding tumour phenotype by noninvasive imaging using a quantitative radiomics approach," *Nature communications*, vol. 5, 2014.

[2] K. Sirinukunwattana, S. E. A. Raza, Y.-W. Tsang, D. R. Snead, I. A. Cree, and N. M. Rajpoot, "Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1196–1206, 2016.