Medical Image Analysis

Chris McIntosh*

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 Preparation and Tutorial

Begin by downloading the zip file for this project which includes four Python files and a directory of data files. Ideally, you should not move the files around once you unzip them. This will allow the programs to refer to the relative location of the data files correctly. You will also need to install so additional Python packages as discussed in the lecture. For this project you need `numpy, scipy, scikit-learn, matplotlib, scikit-image`, and `joblib`. If you did project 1, you should already have the first three.

In the lecture we will be walking through the file image_processing_tutorial.py, and discussing the basic data structures and some of the image processing techniques that will be used in the project. You should start this project by re-familiarizing yourself with this. We recommend you run the file line-by-line playing a bit with the various functions in the shell to make sure you understand the commands that you will need in the next steps. There is nothing to change in this code.

3 The Project Helpers

3.0 Scoring the detector

The first task is to fill in the score_detector function in project_helpers.py. You will need to score if the 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 already calculates the number of false positives (FP) and positives (P) for you.

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

Change the code so that it calculates and returns a tuple containing the precision, recall, F1 Measure, and `FNList`. The formulas are as follows:

- $precision = \frac{TP}{TP+FP}$
- $recall = \frac{TP}{P}$
- $F1Measure = \frac{2TP}{2TP+FP+FN}$

3.1 Basic detection

Now you will continue to complete functions in project_helpers.py that are needed by nuclei_detection_tutorial.py and eventually you will run the complete tutorial.

Start by computing image features to use for nuclei centroid detection by filling in the function calculate_features. 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 skikit-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. *Hint, see first example in nuclei_detection_tutorial.py*

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.

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 prediction again with the new model 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.

1. Start by visualizing the sample image. Can you spot the nuclei centres?
2. Fill in the validate_image code in project_helpers.py to get the features for the new image, perform the prediction, and evaluate the result.
3. Fill out the remaining code in nuclei_detection_tutorial.py. 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.

1. 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.

2. Fill out the remaining code in nuclei_detection_tutorial.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 final 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 using the functionality of 3.4 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. Fill in validate_model by looping over the set of testing images.

2. Complete project.py. Use validate_model for the three classifiers (3.1, 3.2, 3.4), and compare your results. Make sure to keep correspondence between your three different template sets, feature sets, classifiers, and results.

   Hint: If you're having trouble waiting around for the process to finish while you work, use fewer training and testing images until you're confident that you've completed the code. Then run the larger set over lunch!

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.