nucliseg - Documentation

Tamás Gyenis

Tamás Gyenis
+36301951330
tamgyen@gmail.com

TABLE OF CONTENTS

1 APPROACH

The task is segmentation and color-based classification of cell nuclei in high resolution images of IHC stained slides. The contours of the nuclei can be searched for using traditional computer vision approaches or modern representation learning techniques. An easy end-to-end supervised representation learning based approach would require a dataset of instance-level segmentation mask labels for this specific application. On the other hand, classical vision approaches tend to include more fine-tuning, filtering and are often more costly consequently. My solution is a hybrid approach including two models:

1. a deep convolutional neural network that predicts keypoints on the image, which represent the center of cell nuclei and
2. a model that searches for the nucleus contour using classical vision techniques.

The intuition is that we can greatly improve the performance of the classical model by providing prior information.

2 FINDING KEYPOINTS

During the literature search for the project, I found a dataset [1], that contains annotated images of IHC stained slides. The images were labelled by expert pathologists and medical students. The dataset has 1780 slides with 100k+ keypoint labels. Despite the lower resolution, the dataset can be used to train a model for keypoint estimation. We only need to resize each image to a uniform 256x256x3 shape along with their keypoints and convert the dataset to coco format.

Instead of the usual Keypoint R-CNN, we detect keypoints using a heatmap-based anchorless method. This technique is also used in multiple newer object detector models like CornerNet and CenterNet. It allows for flexible and lightweight models for a variety of keypoint detection tasks.

Most of the model training was done using the wonderful repository put together by the authors of [2]. It uses Torch as the low level ML framework and Lightning for managing the training loops. Lightning can also automatically find some hyperparameters like batch size and learning rate for the given training run, which is very convenient for fast model prototyping. Lightning also wraps Albumentations, so we have the option for training time on-GPU augmentations. We also get very nice Wandb logging for experiment tracking.

We train a model with U-net type backbone with 3 very standard CNN residual blocks in the encoder with 2 learnable convolutional kernels per block and BatchNorm. The decoder consists of simple deconvolutions. The head predicts an image-sized heatmap and calculates the binary crossentropy with the Gaussian heatmap generated from the keypoint labels.

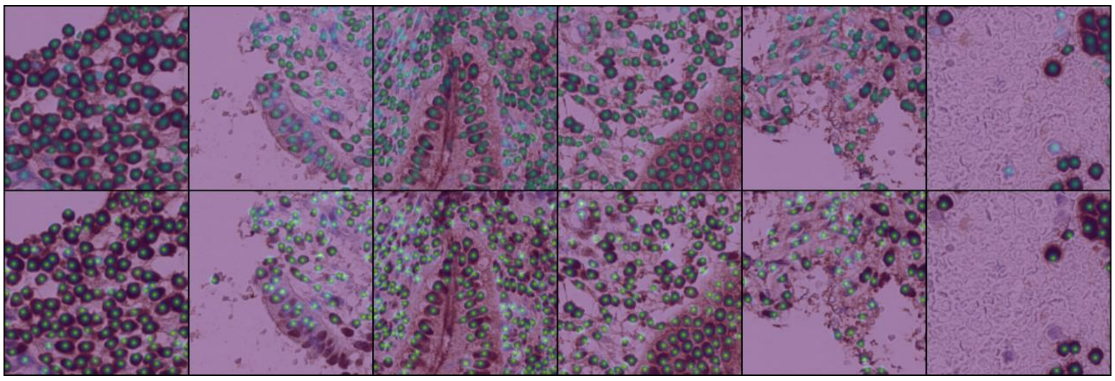

Figure 1 Ground Truth heatmaps (below) and predicted heatmaps (above) for a validation batch for the best model.

I trained 6 models with slightly different hyperpramateres. The training logs can be seen at: https://wandb.ai/tamgyen/nucleus-keypoint?workspace=user-tamgyen

The best model achieved a validation AveragePrecison of 72% with a true-positive pixel threshold of 4, and 90% with a threshold of 8 pixels, meaning it could correctly estimate the keypoint location within 8 pixels for 90% of test samples. Visually checking performance on the src image showed promising performance. The best model is serialized to torchscript and saved for later inference.

In the nucliseg package, the predict_keypoints() function is responsible for inference. For this we first split the large input image into 1024x1024x3 shaped tiles and downsize those to 256x256x3 to match model input dimension. We store the tiles in a list of batch sized batches. Then, we predict the heatmap on each batch and get the keypoints based on local maxima. We finally scale back the predicted points and store them along with their images as KeypointOnImage objects.

3 RESTORING CONTURS

We restore contours separately for each tile. This way we can parallelize the process nicely. We use watershed segmentation which we seed using the keypoints predicted in the previous step. The implementation is done mainly using opencv and numpy for speed. We first color adjust the image to make the object stand out more. I found that boosting the blue channel and the contrast helps. We then extract the background pixels by thresholding the image based on its histogram using Otsu's method. We open the binary mask of possible objects to remove some noise, then dilate it a bit. Here we need to make sure no keypoints land in the background. We can check by plotting some overlays:

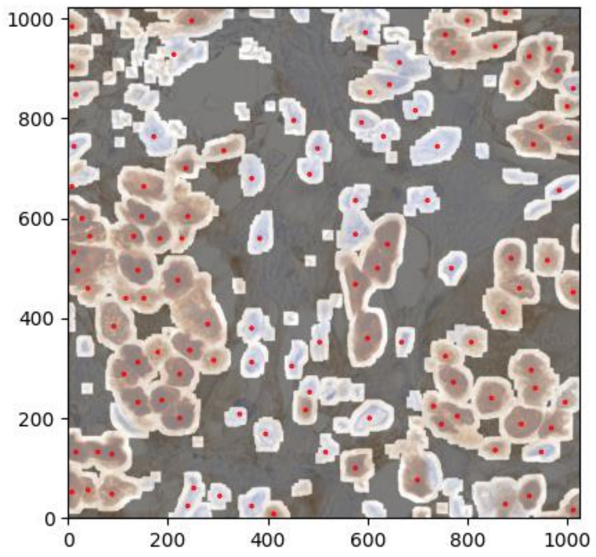

Figure 2: binary mask of background (zeros -bg) overlaid on original image.

We then place the seeds by creating a scalar labelled mask using the keypoints. We label a circular area around each keypoint with a different scalar. We subsctract the foreground mask (where there are scalar labels) from the background mask to get the ambiguous region for the watershed to find the contours in. We execure the watershed algorithm and return the new masks for each tile.

The next step is to classify the masks and plot their contours. For the classification we check the mean HSV color for each image segment that lies under a given mask and calculate the L2 distance of the color vector from reference color vectors. We select the class of the minimum distance. Finally, we calculate the contour of the mask and draw it with the selected color.

4 CONCLUSION AND FURTHER DEVELOPMENT

With this hybrid method we are able to predict nucleus keypoints and then use them to find the actual contours of the objects. Using keypoints as references helps the performance of the segmentation algorithm by excluding false positive regions.

- The tile-based execution allows easy parallelization but introduces artifacts along the tile boundaries. This could be filtered with additional post processing for some extra computational cost.
- Both models have many hyperparameters. Tuning and experimenting, especially for the watershed model could led to considerably better performance. For this, the algorithm is easily configurable using kwargs.

- Regarding the keypoint predictor performance on the src image could be increased by:
  - Pseudo labelling by including some prediction on the src image in the training dataset
  - Copy-paste augmentation by randomly pasting some object to empty regions on the sample image

5 NOTES

- Runtime is ~2 mins end-to-end using a GTX1060 and a core i5-6600k
- Find the repo: https://github.com/tamgyen/nucliseg

6 REFRENCES

[1] https://www.mdpi.com/2306-5729/7/6/75
[2] https://github.com/tlpss/keypoint-detection
[3]