Selected Topics in Visual Recognition using Deep Learning

HW3 - Nuclei segmentation

陳永承 0712534

Project: https://github.com/axde954e6/NYCU_VRDL/tree/main/HW3

Introduction

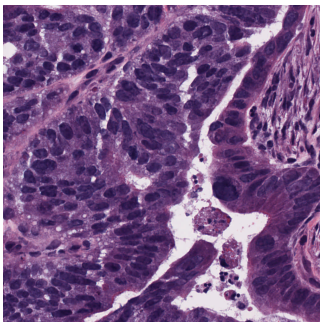
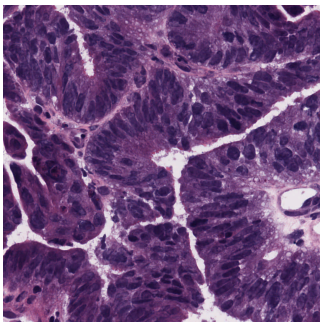
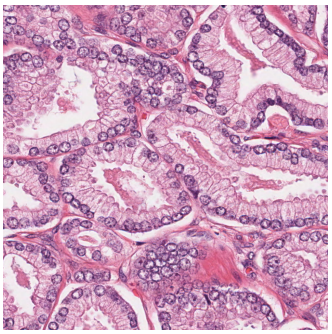
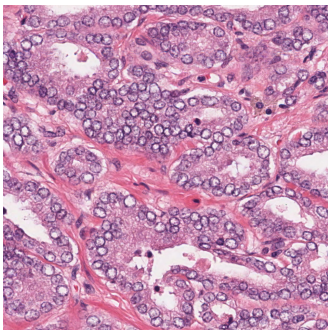
The challenge is Nuclei segmentation. Using data provided by TA, which includes about 24 training images with 14598 nuclear and 6 testing images with 2360 nuclear. In this work, we need to use an instance segmentation model to detect and segment all nuclear in each image.

Methodology

Data Pre-processing

The format of training data, which is provided by TA, can be directly used in Mask RCNN. But there is a lot of training data when you use Colab as an environment, which will cause your model to spend a lot of time loading the dataset. Therefore, I divide those data into many folders. Besides, it will automatically batch normalization, random crops, resize the image, and random flip when you use Mask RCNN to train the model.

I found that some pictures in training data are similar to some testing data, so I will focus on training those data.

training data	testing data
 TCGA-NH-A8F7-01A-01-TS1	 TCGA-AY-A8YK-01A-01-TS1
 TCGA-CH-5767-01Z-00-DX1	 TCGA-G9-6336-01Z-00-DX1

Model Architecture

I use Mask RCNN as our Nuclei segmentation model. To avoid GPU out of memory happening, I decrease `IMAGES_PER_GPU` to 2. And I use Colab to train, which I don't know will cut GPU usage, so I decrease `STEPS_PER_EPOCH` to 5 and `VALIDATION_STEPS` to 1. By doing this, it will save models frequently and also not spend a lot of time on detecting validation data. It also can select the backbone of Mask RCNN. In this work, I use resnet101 as my backbone.
Besides, Mask RCNN will train network heads and all layers separately, so I train network heads first. when the map of testing data is barely improved, training all layers.

Learn Rate Scheduling

Mask RCNN will automatically learn rate schedules. However, it is because I decrease STEP_PER_EPOCH, which will increase the number of epochs but will decrease the number of steps in each epoch, the learning rate will decrease faster and the model will not get enough information. To solve this problem, I reuse the model as pretrain weight and retrain the model again and again.

Model Ensemble

I found that each model has high performance on different test data (I think it is based on training data), so I use 2 models to ensemble. Each model is used to detect different test data. Then merge them together.

Make Submission

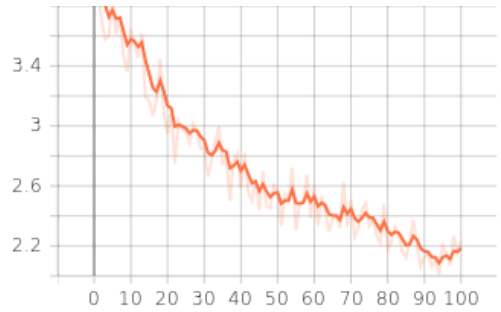
I modify part of the code in nucleus.py, it can dump scores and mask images. Then I use cocoapi to convert mask images to RLE and bbox, and combine score and RLE and bbox to a json file.

Summary

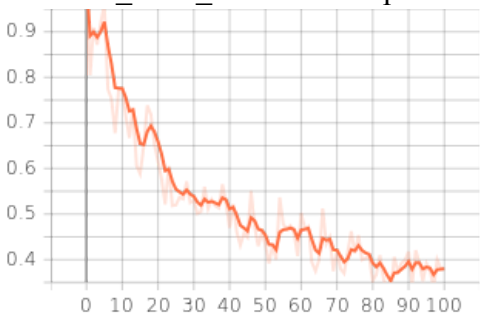
Loss Curve

To evaluate the model, I randomly choose a training image as validation data. Below figures are loss to each epoch (In the first time training model).

loss to each epoch



mrcnn_bbox_loss to each epoch



Some useful methods

For the first time, I trained the network heads using resnet50 as the backbone.

Using training model as pretrained weight

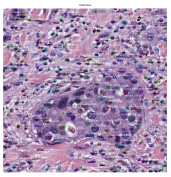
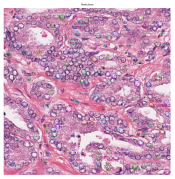
After 100 epochs, the map of testing data is about 0.16. I found that it is barely improving because the learning rate is much slower. So I reuse the model as a pretrained weight and start training the model again and again. By doing this, the map of testing data increases to 0.216.

Change backbone to resnet101

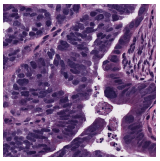
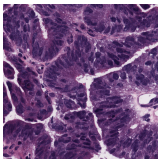
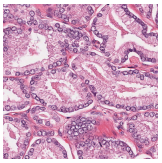
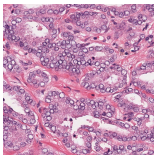
But it seems that is the limit of the original architecture. So I changed the backbone to resnet101, and I think the original model has better performance for network heads. So I use the original model as pretrain weight and only train all layers. After doing this, the map of testing data increases to about 0.22.

Training the image which is similar to testing data

In this case, I found that this model has high performance in some pictures and has bad performance in others. Therefore, I guess it may spend more time on training good performance datasets and spend less time on training bad performance datasets.

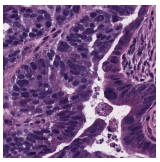
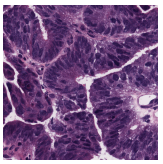
good performance	bad performance
	

So I choose images that are similar to testing data and have bad performance as my training data and retrain the model. By doing this, I got the 0.232map of testing data, which is quite close to the baseline.

training data	testing data
 TCGA-NH-A8F7-01A-01-TS1	 TCGA-AY-A8YK-01A-01-TS1
 TCGA-CH-5767-01Z-00-DX1	 TCGA-G9-6336-01Z-00-DX1

Model Ensemble

I found that the original model has poor performance on TCGA-AY-A8YK-01A-01-TS1. I think it is because the training data is only TCGA-NH-A8F7-01A-01-TS1. Therefore, I use the original model as pretrain weight and only train this data. This model only detects TCGA-AY-A8YK-01A-01-TS1, and the original model detects other images. By this work, the map of testing data increases to 0.234, which is better than the baseline.

training data	testing data
 TCGA-NH-A8F7-01A-01-TS1	 TCGA-AY-A8YK-01A-01-TS1

Reference

https://github.com/matterport/Mask_RCNN
https://github.com/cocodataset/cocoapi