Comp 540 final project final report
Kaggle Data science Bowl 2018

Team name: Comp540_Rice_my26_yh46
Ming-Hsuan Yen, Yuefei Huang

Introduction

The goal of Data science bowl 2018 is to try to build a better model to automate the recognition of nuclei in images. Some nuclei in images are hard even for human to spot. A machine learning model that spot nuclei can help researchers improve their results. With a relativity small training data set available for this competition, overfitting is a problem on training a deep neural network. Various image augmentations are tested and applied to images and mask. We tried two different approaches to this problem, general image segmentation and instance base segmentation. An article from Ronneberger proposed [1] a fully convolutional network architecture that performs well on small data set named U-net. A slightly simplified u-net was built for the segmentation problem. Mask-RCNN is a RCNN extension network architecture proposed by He[3] have outperformed other model on Imagenet data set. A similar network architecture was trained and compare to our U-net model.

Data analysis

In the competition 670 images are provided as training set, each image has several masks for training ground truth mask. Number of masks depends on the objects that appears on the image. The objective of data analysis is to build intuition on the data set. In order to build robust model or get better preprocessing.

A great data exploration is done by Thomas in his post [4]. From the post most of the training data image has the size of 256 by 256. Yet size images in stage one test has more variety. Size of image depends highly on how the image is taken, which means the technique used to get the nuclei images. Numbers of images in different size are shown in figure 1. The distribution is closer to stage 1 train, which makes model selection on stage 2 test more difficult, since selecting model base on stage 1 test might be not reliable.

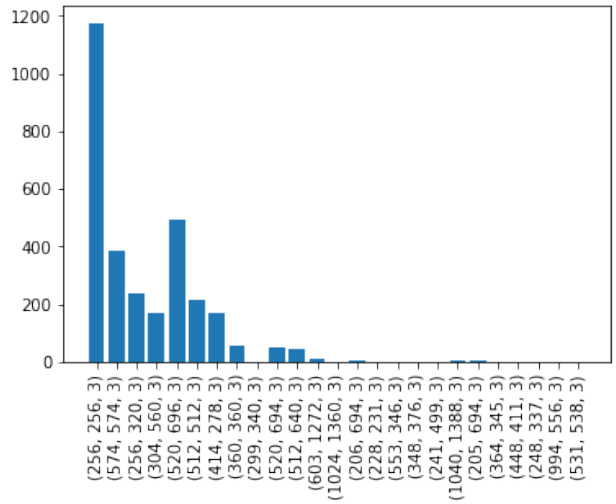


Figure 1. Bar plot of size of stage 2 test images.

Number and size of nucleus are correlated with image size. Base on this observation K-means clustering is applied to split the data set. By manually going through training images, varies kinds of images are found in the training set as shown in Figure 2. K-means clustering on dominate color of images is used to separate images so that we can get good normalization of the data set. Also, to prevent building models only on one type of image We split data into training and validation set with roughly the same percentage in each type of images.

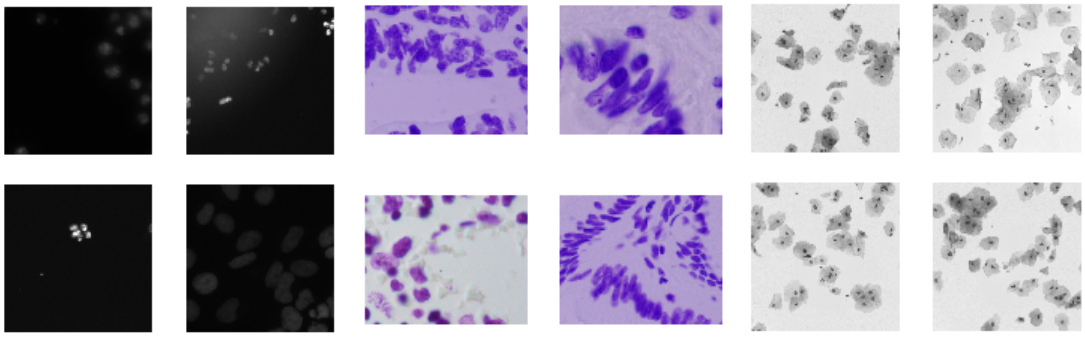


Figure 2. Different types of image separate by K-means clustering.

Table 1. Percentage of different types of images in training data.

Image Type	percentage
Type 1	81.49 %
Type 2	16.11 %
Type 3	2.38 %

After the release of stage 2 test set, we cluster stage 2 data with stage 1 train cluster center. Most of stage 2 data are clustered with Type 1 image. Furthermore, no image in stage 2 is in the same cluster with type 3 images. With K-means cluster result, Type 3 image3 are characterize as outliers.

Yet by observing stage 2 images, it is easy to find some images like those in figure3 which looks different from images in stage 1 train.

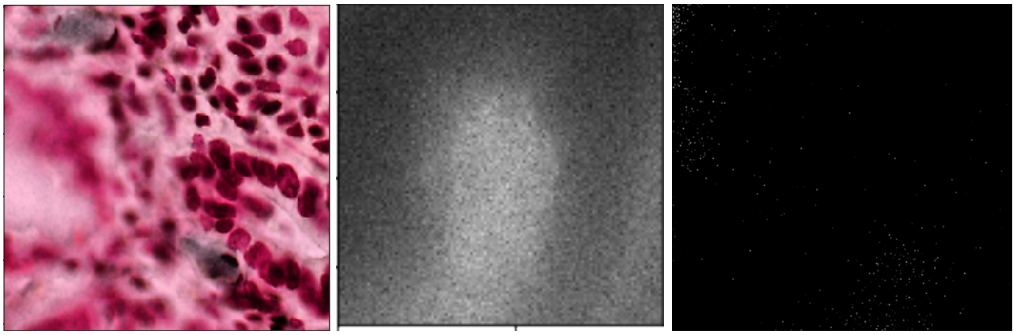


Figure 3. Image Types in stage 2 test set that are not in stage one train.

Image preprocessing

Image Resizing

With observation from data analysis, there are images of varies size in the training data. A fix size convolution network is much easier to build and more can produce more consistence results. Thus, we resize image into two sizes and feed in to U-net. 256 by 256 is due to majority of training images are 256 by 256. With the nature of u-net structure not all image sizes are valid more specifically the 2 by 2 upsampling layers. Only the one that is even, can divide by 6 and residual of dividing 3 is 2. So, slightly lager 368 by 368 input was also tested. Larger input slightly increase result of u-net IoU by 0.01 in validation set, but not any improvement on public leaderboard during stage 1 is achieved.

Histogram Equalization

CLAHE or Contrast-limited adaptive histogram equalization is a common image processing for improving contrast of images. Illustrate in figure 4 some images like the one has black background(a) nuclei is blur and hard to spot. After applying histogram normalizing on V channel in HSV color space than convert to RGB color space. We notice that not only nuclei in black background pops up but in bright back ground as well, so all images are processed with CLAHE before data augmentation and fed into neural network.

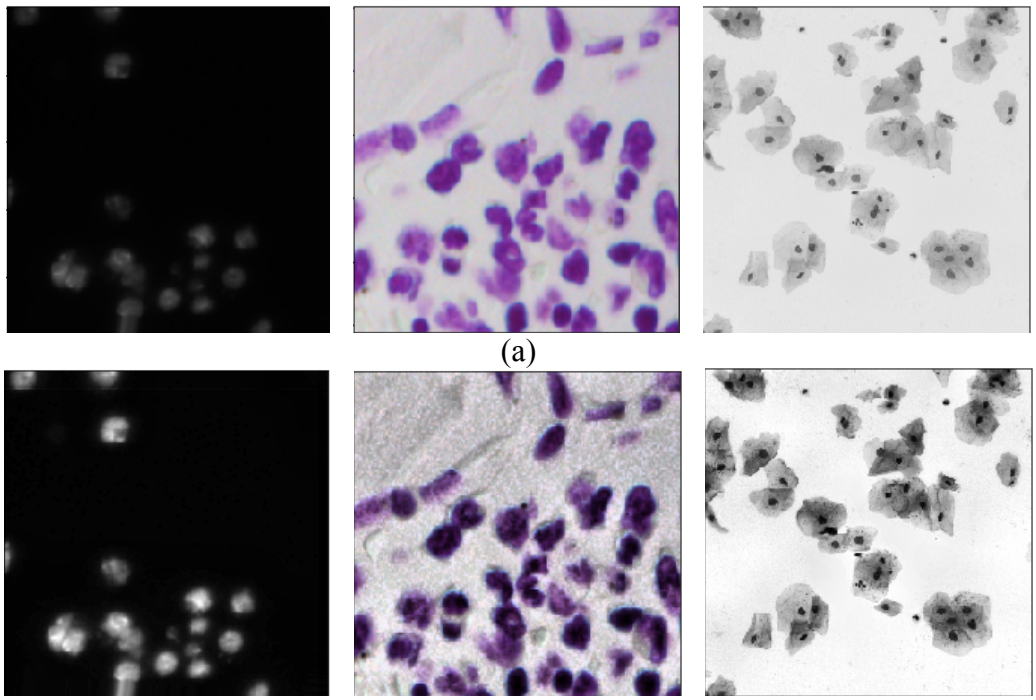
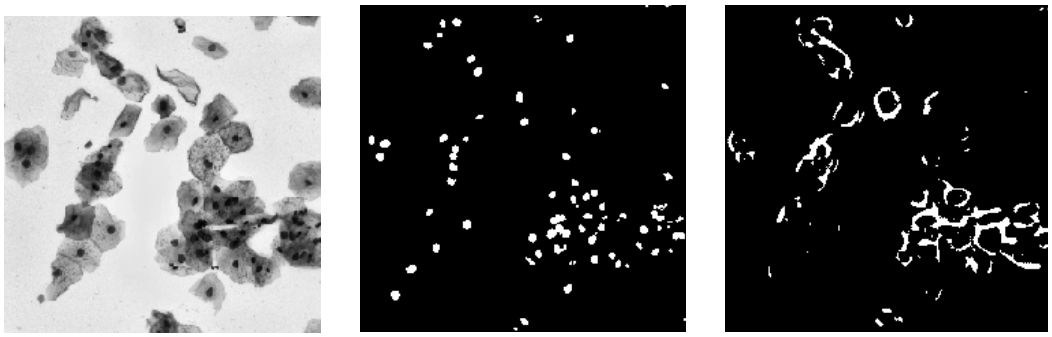


Figure 4. Images before(a) and after CLAHE(b).

Negative Image

As result from K means clustering, some of the images have bright background and some have dark background. From prediction result with raw image shown in Figure 5, we can see that prediction mask of type 3 image is much worse than other types. Models tends to predict the entire cell rather than the nucleus since this type of image has much smaller nucleus. This maybe cause by different technique of getting nuclei images thus getting better contrast than other types of images. Transforming this type of image into negative image seems right, however we later found out that convolutional network will predict similar result. Thus, this transform image to negative image was dropped after we noticed it do not make any difference of our models in terms of stage 1 test public leaderboard score. However, after the release of stage2 test set, no Type 3 image is found in stage 2 test set, so we drop all Type 3 images. The dropped image list is in appendix. With stage 1 test set added to training data I believe this can produce more generalize data without outliers.

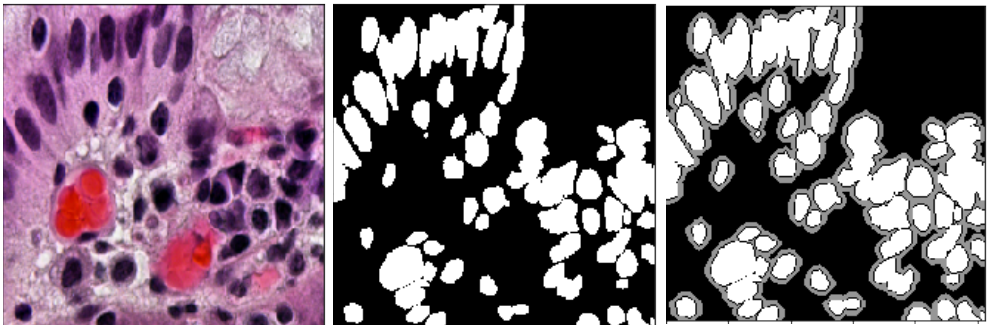


(a) (b) (c)

Figure 5. Type 3 image(a)ground truth mask (b) and prediction mask(c).

Mask watershed

Inspired by one of data science bowl 2017 kernel Improved Lung Segmentation using Watershed [4], we tried to improve prediction result of our model by adding watershed on ground truth masks. Several transformations were tested as illustrate in Figure 6. To improve detection of small gaps between nucleus, we first added watershed to mask and applied to train U-net, however the result did not improve. So, we trained U-net with two channel outputs, one is a ground truth mask and the other is ground truth mask with watershed. The result improved about 0.02 on stage 1 public leaderboard.



(a) (b) (c)

Figure 6. Original images(a), ground truth mask(b) and watershed mask(c).

Image Augmentation

To improve the result of our model several image augmentations are applied to the data set. Two different approach was tried to improve the result. First, we use parameters listed in table 2 to randomly transform images and masks, and use transformed images as input to network. Augmented data did slightly improve result of training and validation IoU, however did not show much on stage 1 public leaderboard. Thus, another image augmentation method was tested.
Other method rather than doing augmentation on one image is a variation of augmentation network purpose by Wang [6]. It is to generate 10 images with parameters listed in table2, which expands 1 image to 11 including original one as in figure 7. While training, we randomly choose and concatenate 2 images from 11 images generated beforehand and pass the 6 channel tensors to a five layers convolutional augmentation network. Then pass the outputs to u-net and update all parameters at the same time in each epoch. As the article uses this augmentation method in image

classification and works fine, but it did not work well on image segmentation problems, a reconstruction of network architecture is needed.

Augmentation network architecture

Conv with 16 channels and 3x3 filters. Relu activations.

Conv with 16 channels and 3x3 filters. Relu activations.

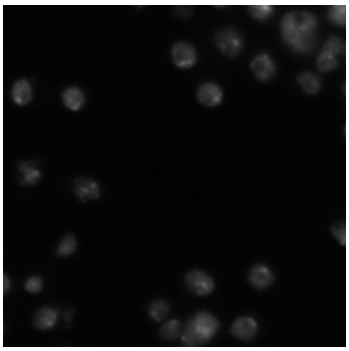
Conv with 16 channels and 3x3 filters. Relu activations.

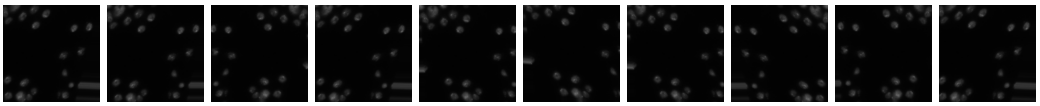
Conv with 16 channels and 3x3 filters. Relu activations

Conv with 3 channels and 3x3 filters.

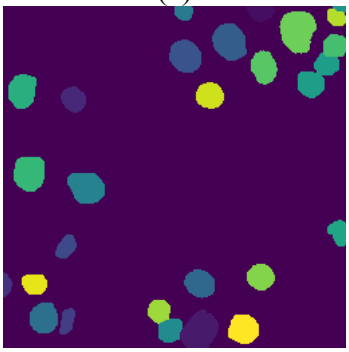
Table 2. Image augmentation parameters

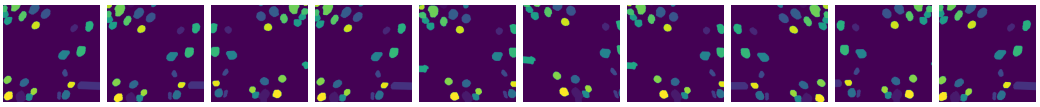
Random rotation	45 degree
Random width shift	20 %
Random height shift	20 %
Random shear	30%
Random zoom	20 %
Random vertical flip	-
Random horizontal flip	-





(a)





(b)

Figure 7. Example of input image(a) and ground truth mask(b) before and after random image augmentation.

Model

Two image segmentation approach was tuned and trained for the competition. U-net propose by Ronneberger is a fully convolution neural network mainly used in image segmentation. It's the base architecture of several Kaggle competition winner models. Mask-RCNN is a new variation of RCNN propose by He. It combined object detection, classification and image segmentation in one single model. Mask-RCNN is an instance base model which means it produce one mask for each object it detects, on the contrary U-net produce a mask that contains all masks for each object we wish to detect.

U-net

Our U-net model is based on the original one illustrated in figure 8. Instead of cropping layers, dropout layers are used to prevent overfitting and keep input and output size consistent. Detailed architecture is in appendix. As mentioned in image preprocessing section, different input and output are used to train U-net. The best stage 1 single model gets 0.365 on public leaderboard with raw image input. The best stage 2 U-net model uses augmented input and reaches 0.28 on private leaderboard. Unfortunately, we choose the best performed model as our final score which is 0.130 on private leaderboard this indicates the model is not a generalize model.

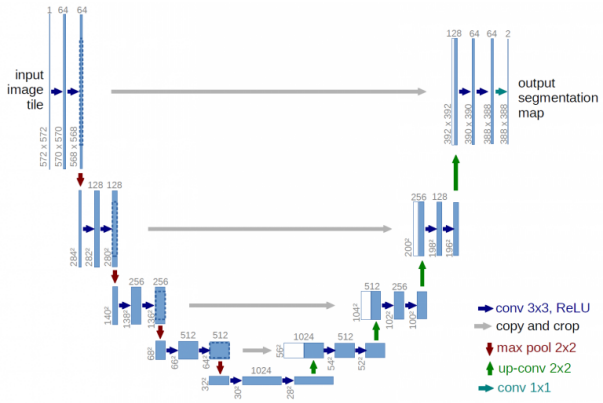


Figure 8. U-net architecture purpose by Ronneberger.

Sørensen-Dice coefficient was used to evaluate model during training, because mean IoU takes a lot longer to compute thus it makes training process longer even on GPU. We choose sum of binary cross entropy loss and dice coefficient loss to be the objective loss function to minimize. Our intuition is to get both number of ones and the position of pixels that are ones as close to the ground truth as possible.

Sørensen-Dice coefficient:

$$d = 1 - \frac{2|X \cap Y|}{|X| + |Y|}$$

Loss function:

Binary cross entropy loss + Dice coefficient loss

Binary cross entropy loss:

$$\sum_i y_{true} \log\left(\frac{1}{y_{pred}}\right)$$

Dice coefficient loss:

1- Dice coefficient

Mask-RCNN

Mask-RCNN is a variation of Fast-RCNN published earlier this year. It outperformed older methods on imagent. A GitHub project implemented a keras version with tensorflow backend [7] makes training on this dataset with mask-rcnn far easier. We feed in stage 1 train and test data and train with COCO pretrain weights on Resnet50. After training on single GPU for around 8 hours, a very good mask can produce very good prediction on black background images, but other types of images have very bad prediction as shown in Figure 9. With this result and running out of time we decide not to upload the result of Mask-RCNN. More training time is required even with COCO pre-train weight.

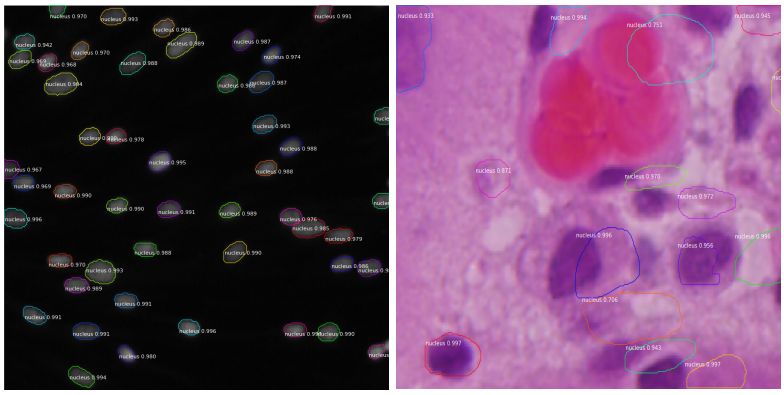
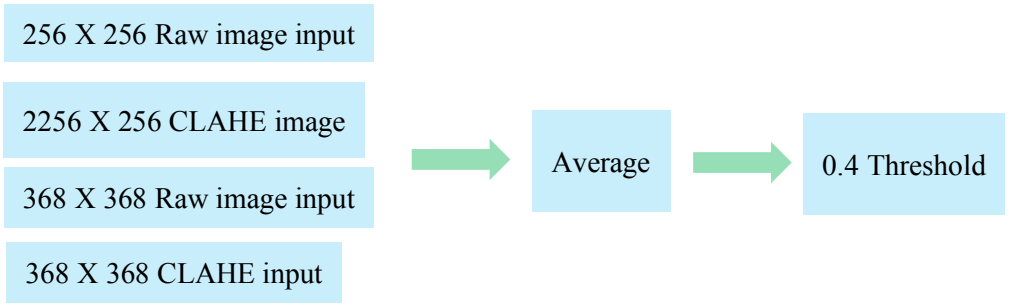


Figure 9. Prediction of Mask-RCNN after 8 hours.

Ensemble

We started building ensemble model after stage 2 test release. So, we did not have a chance to evaluate our ensemble model using query set on Kaggle. To further reduce variance of the output, several models' outputs were produced. Averaging over of them gives the final prediction. Yet the ensemble model did not work well on private leaderboard. The reason might be threshold of the first layer and the final production is hard to select with same evaluating data (validation set). The score of this ensemble method is below 0.1 on private leaderboard. With better threshold selection method and first layer model selection we believe this method can have more fruitful result.



Training flow

1. Resize training images and mask to 256 by 256 or 368 by 268 3 channel images. All grey scale images are convert to RGB images. CLAHE is apply after resizing.
2. Generate mask waterhed this (not in stage 2 test).
3. Do image augmentation (in some model)
4. Initialize U-net model.
5. Train until 1500 epoch or loss do not reduce for 10 epochs learning rate is reduced by ½ if loss do not reduce for 3 epochs but not less than 0.000001.
6. Resize test images as the same to training images. CLAHE is also apply after resizing.
7. Predict submission result.

Adam is applied as update method with 0.0001 as learning rate. Weights are initialized by kernel initializer 'he_normal' in Keras Tensorflow backend whose weights are initialized by normal distribution with zero mean and stander deviation equal to $\sqrt{2/n}$, where n is number of inputs in the layer. Sørensen-Dice coefficient is used to monitor during training time, while sum of Binary cross entropy loss and Dice coefficient loss is the loss function. Finally, mean IoU is applied to evaluate models on validation set in order to tune parameters.

All of the models are trained by GPU on Amazon web service deep learning AMI (Amazon Linux) version 6.0 with 'p2.xlarge' a 12 GB Tesla K80 GPU with 4 core CPU and 60 GB RAM. Models are built with python 3 using Keras Tensorflow backend with support of linear algebra package like numpy scipy and so on. On average U-net model can be trained under an hour with 80 ~ 100 epochs. For Mask-RCNN with ResNet 50 train with freeze backbone for 20 epochs and keep training for around 8 hours. Each epoch takes more than 10 minutes that will result in around 50 epochs in total.

Model Results

Our top five scored result are shown in figure 10. Models are sorted by upload time. Model 1 is the earliest and model 5 is the latest one. Mean IoU are listed in table 3. Model selection for stage 2 test is based on stage 1 public leader board score.Yet it is not a good evaluating method but still better than using validation score due to overfitting of our model. The model is trained with same method but with additional stage 1 test data as external data. Details of these model are in appendix.

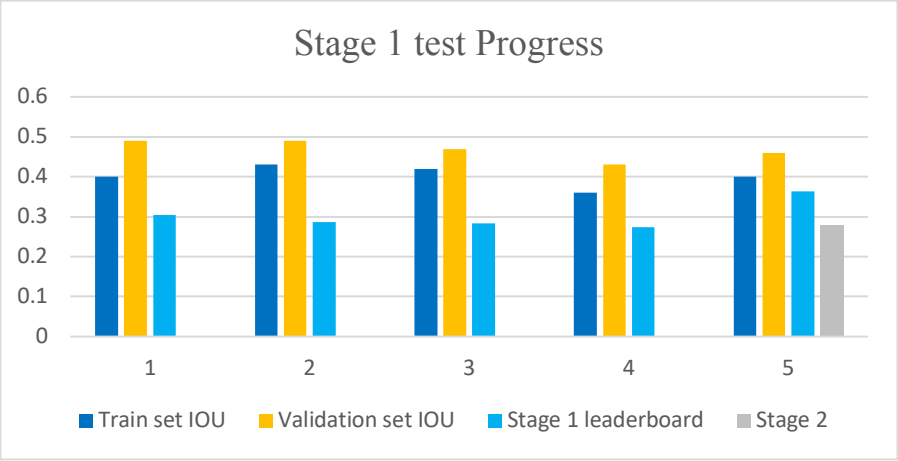


Figure 10 Training set, validation set IoU and stage 1, stage 2 scores on leaderboard.

Table 3. progression of model mean IoU over time

	Train set	Validation set	Public leader board	Private leader board
Model 1	0.40	0.49	0.305	-
Model 2	0.43	0.49	0.286	-
Model 3	0.42	0.47	0.283	-
Model 4	0.36	0.43	0.273	-
Model 5	0.40	0.46	0.363	0.280
Mask-RCNN	0.256	0.270	-	0.033

Note that due to lack of time MASK_RCNN did not train long enough to get good prediction result. The private leaderboard score is from late submission and is just to see how it perform compare to our U-net models.

Summary

Overall, our model performed good prediction on type 1 images. However, it performs poorly on predicting outliers type 3 image masks. Several preprocessing methods were tested during stage 1. Yet we focuse on improving prediction result of type 3 image that do not exist in stage2 test set this is the main failure of our approach. A better model can be built if we focus on building a more generalize model instead of trying to make the model fits on a minority group of the data.

Appendix

Type3 cluster image IDs:

'c395870ad9f5a3ae651b50efab9b20c3e6b9aea15d4c731eb34c0cf9e3800a72',
'5e263abff938acba1c0cff698261c7c00c23d7376e3ceacc3d5d4a655216b16d',
'547936244l3c7d0e048173f7aeee85de3277f7e8d47c82e0a854fe43e879cd12',
'2a1a294e21d76efd0399e4eb321b45f44f7510911acd92c988480195c5b4c812',
'4e07a653352b30bb95b60ebc6c57afbc7215716224af731c51ff8d430788cd40',
'091944f1d2611c916b98c020bd066667e33f4639159b2a92407fe5a40788856d',
'5d58600efa0c2667ec85595bf456a54e2bd6e6e9a5c0dff42d807bc9fe2b822e',
'8d05fb18ee0cda107d56735cafa6197a31884e0a5092dc6d41760fb92ae23ab4',
‘3594684b9ea0e16196f498815508f8d364d55fea2933a2e782122b6f00375d04',
‘8f94a80b95a881d0efdec36affc915dca9609f4cba8134c4a91b219d418778aa',
'4217e25defac94ff465157d53f5a24b8a14045b763d8606ec4a97d71d99ee381',
'7f38885521586fc6011bef1314a9fb2aa1e4935bd581b2991e1d963395eab770',
'08275a5b1c2dfcd739e8c4888a5ee2d29f83eccfa75185404ced1dc0866ea992',
'1a11552569160f0b1ea10bedbd628ce6c14f29edec5092034c2309c556df833e',
'1b44d22643830cd4f23c9deadb0bd499fb392fb2cd9526d81547d93077d983df',
‘76a372bfd3fad3ea30cb163b560e52607a8281f5b042484c3a0fc6d0aa5a7450'

U-net model summary:

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 368, 368, 3)	0	
conv2d_41 (Conv2D)	(None, 368, 368, 32)	896	input_3[0][0]
dropout_19 (Dropout)	(None, 368, 368, 32)	0	conv2d_41[0][0]
conv2d_42 (Conv2D)	(None, 368, 368, 32)	9248	dropout_19[0][0]
max_pooling2d_9 (MaxPooling2D)	(None, 184, 184, 32)	0	conv2d_42[0][0]
conv2d_43 (Conv2D)	(None, 184, 184, 64)	18496	max_pooling2d_9[0][0]
dropout_20 (Dropout)	(None, 184, 184, 64)	0	conv2d_43[0][0]
conv2d_44 (Conv2D)	(None, 184, 184, 64)	36928	dropout_20[0][0]
max_pooling2d_10 (MaxPooling2D)	(None, 92, 92, 64)	0	conv2d_44[0][0]
conv2d_45 (Conv2D)	(None, 92, 92, 128)	73856	max_pooling2d_10[0][0]
dropout_21 (Dropout)	(None, 92, 92, 128)	0	conv2d_45[0][0]
conv2d_46 (Conv2D)	(None, 92, 92, 128)	147584	dropout_21[0][0]
max_pooling2d_11 (MaxPooling2D)	(None, 46, 46, 128)	0	conv2d_46[0][0]
conv2d_47 (Conv2D)	(None, 46, 46, 256)	295168	max_pooling2d_11[0][0]
dropout_22 (Dropout)	(None, 46, 46, 256)	0	conv2d_47[0][0]
conv2d_48 (Conv2D)	(None, 46, 46, 256)	590080	dropout_22[0][0]
max_pooling2d_12 (MaxPooling2D)	(None, 23, 23, 256)	0	conv2d_48[0][0]
conv2d_49 (Conv2D)	(None, 23, 23, 512)	1180160	max_pooling2d_12[0][0]
dropout_23 (Dropout)	(None, 23, 23, 512)	0	conv2d_49[0][0]
conv2d_50 (Conv2D)	(None, 23, 23, 512)	2359808	dropout_23[0][0]
conv2d_transpose_9 (Conv2DTrans	(None, 46, 46, 256)	524544	conv2d_50[0][0]
concatenate_9 (Concatenate)	(None, 46, 46, 512)	0	conv2d_transpose_9[0][0]

			conv2d_48[0][0]
conv2d_51 (Conv2D)	(None, 46, 46, 256)	1179904	concatenate_9[0][0]
dropout_24 (Dropout)	(None, 46, 46, 256)	0	conv2d_51[0][0]
conv2d_52 (Conv2D)	(None, 46, 46, 256)	590080	dropout_24[0][0]
conv2d_transpose_10 (Conv2DTran	(None, 92, 92, 128)	131200	conv2d_52[0][0]
concatenate_10 (Concatenate)	(None, 92, 92, 256)	0	conv2d_transpose_10[0][0] conv2d_46[0][0]
conv2d_53 (Conv2D)	(None, 92, 92, 128)	295040	concatenate_10[0][0]
dropout_25 (Dropout)	(None, 92, 92, 128)	0	conv2d_53[0][0]
conv2d_54 (Conv2D)	(None, 92, 92, 128)	147584	dropout_25[0][0]
conv2d_transpose_11 (Conv2DTran	(None, 184, 184, 64)	32832	conv2d_54[0][0]
concatenate_11 (Concatenate)	(None, 184, 184, 128	0	conv2d_transpose_11[0][0] conv2d_44[0][0]
conv2d_55 (Conv2D)	(None, 184, 184, 64)	73792	concatenate_11[0][0]
dropout_26 (Dropout)	(None, 184, 184, 64)	0	conv2d_55[0][0]
conv2d_56 (Conv2D)	(None, 184, 184, 64)	36928	dropout_26[0][0]
conv2d_transpose_12 (Conv2DTran	(None, 368, 368, 32)	8224	conv2d_56[0][0]
concatenate_12 (Concatenate)	(None, 368, 368, 64)	0	conv2d_transpose_12[0][0] conv2d_42[0][0]
conv2d_57 (Conv2D)	(None, 368, 368, 32)	18464	concatenate_12[0][0]
dropout_27 (Dropout)	(None, 368, 368, 32)	0	conv2d_57[0][0]
conv2d_58 (Conv2D)	(None, 368, 368, 32)	9248	dropout_27[0][0]
conv2d_59 (Conv2D)	(None, 368, 368, 2)	578	conv2d_58[0][0]
conv2d_60 (Conv2D)	(None, 368, 368, 1)	3	conv2d_59[0][0]

Total params: 7,760,645
Trainable params: 7,760,645
Non-trainable params: 0

Model details:
Model 1: U-net model with 6 pooling and upsampling layers. Raw 256X256 image input.
Model 2: U-net model with 5 pooling and upsampling layers. Before every pooling and upsampling layers a batch-norm layer is added. Input image is resized into 256X256 and convert to HSV color space and apply CLAHE including test set.
Model 3: Similar to model 2 but input is in RGB color space.
Model 4: U-net model with 5 pooling and upsampling layers. Random rotation, horizontal flip, vertical flip and random zoom are applied in the training phase. No color space convert or histogram equalization is applied.
Model 5: Base line model with 256X256 image input. A fine tune U-net model with 5 pooling and upsampling layers without any data augmentation.

Reference

[1] Olaf Ronneberger, Philipp Fischer and Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015
[2] Ross Girshick, Fast R-CNN, ICCV 2015
[3] Kaiming He, Georgia Gkioxari, Piotr Dolla´r and Ross Girshick, Mask R-CNN
[4] Jerry Thomas, Exploratory Analysis https://www.kaggle.com/jerrythomas/exploratory-analysis
[5] Data seience bowl 2017, Improved Lung Segmentation using Watershed https://www.kaggle.com/ankasor/improved-lung-segmentation-using-watershed/data
[6] Jason Wang, Luis Perez The Effectiveness of Data Augmentation in Image Classification using Deep Learning
[7] Mask-RCNN keras : https://github.com/matterport/Mask_RCNN

Kaggle post:

U-net base line model : https://www.kaggle.com/keegil/keras-u-net-starter-lb-0277?scriptVersionId=2164855

Python library:

Keras Tensorflow API: https://keras.io
Imgaug image augmentation package: http://imgaug.readthedocs.io/en/ltest/
Skimage for image I/O and processing: http://scikit-image.org
Opencv image I/O and processing: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html
Numpy for linear algebra: http://www.numpy.org
Scipy for linear algebra: http://www.scipy-lectures.org
Scikit-learn useful machine learning tools: http://scikit-learn.org/stable/