School of Electronic
Engineering and
Computer Science

MSc Big Data Science
Project Report 2018

Histological Image Synthesis Using Deep Learning

Watchara Sirinaovakul

August 2018

Disclaimer

This report, with any accompanying documentation and/or implementation, is submitted as part requirement for the degree of MSc in Big Data Science at the University of London. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Abstract

Deep learning has a lot of potentials and has been applied to many tasks in medical imaging. However, training a deep learning model often requires a large training dataset, which is very expensive. This project will introduce an Optional Pixel to Pixel model which will synthesize training dataset based on options using a small training dataset and the synthesized training dataset can be used for further object segmentation tasks. The Optional Pixel to Pixel model is an improved version of the famous Pixel to Pixel model but unlike the plain one, Optional Pixel to Pixel is capable of handling variations in the dataset using an option layer. The Optional Pixel to Pixel takes synthesized masks as input and produces synthesized images. This project also shows how to synthesize masks using random sampling nucleus from Disentangled Variational Autoencoder model. The result of this project looks very realistic and the synthesized images look reasonably similar to the original images. This project also shows that the Optional Pixel to Pixel model works better than the plain Pixel to Pixel model since the former can synthesize images based on options. We hope this model will be useful for further nuclei segmentation tasks.

Acknowledgements

I would like to say special thanks to my supervisor Dr.Qianni Zhang for giving me a great opportunity to work in this project and supporting me everything I need. Without her, I would not have done my project at this quality. I can wholeheartedly say that doing dissertation is one of the best parts of my masters degree.

Contents

1 Introduction

1.1 Overview

In medical examinations, medical images is one of the crucial tools to diagnose patients' diseases. The most used types of medical images is histological images which is produced by taking patients' tissues and then digitizing them by microscope. These images are in turn analyzed by medical experts and the result is given back to the patient. The process of analyzing the medical images is very tedious and expensive as the medical experts have to manually examine the images one by one.

Deep learning techniques is more and more widely used for biomedical imaging since it gives better results than traditional approaches. In the deep learning algorithms such as nuclei segmentation, the model requires images and ground-truths or masks and has to be large enough to train a model. However, obtaining biomedical training datasets is very expensive and difficult as expertises are required. Therefore, this project will tackle this unavoidable problem of deep learning which is lacking of training dataset. More specifically, this project will build a model that is used to synthesize fake training dataset using a deep learning techniques called Generative Adversarial Networks which is very successful in the recent years especially for generating realistic looking images.

1.2 Motivation

When examining histological images, fig.1, a medical expert need to analyze the large amount of images manually, which is time consuming and has a high potential to be substituted by deep learning robots. However, supervised deep learning techniques take large training datasets, which is not easy to be obtained.

The influential research for this project is "Image-to-Image Translation with Conditional Adversarial Networks (Pixel to Pixel) [7]" which generates images from their corresponding images, the example can be found in fig.2. As shown in the examples, this research is able to map 2 types of images such as from a city image to a map or a drawing to an image. The research seems adjustable to our task because deep learning in biomedical image usually uses 2 types of images in their training datasets which are images and ground-truth masks. So, this project will apply Pixel

to Pixel algorithm to generate histological images from random masks.

If this project is successful, the problem of lacking training dataset in biomedical research will be solved and will have a high impact to the community.

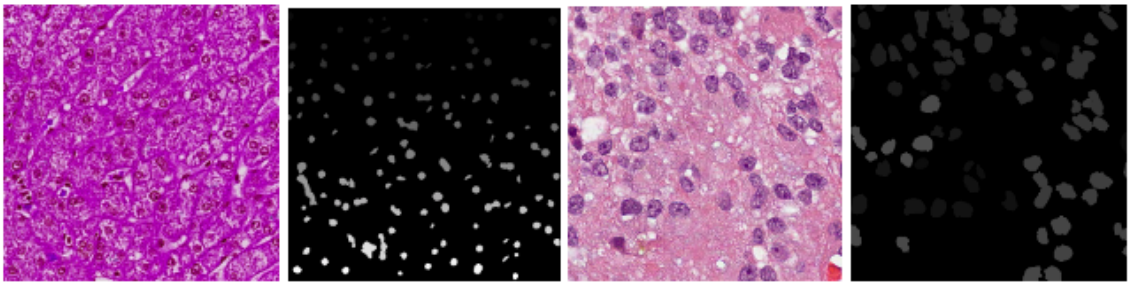


Figure 1: shows example images and masks in our dataset.

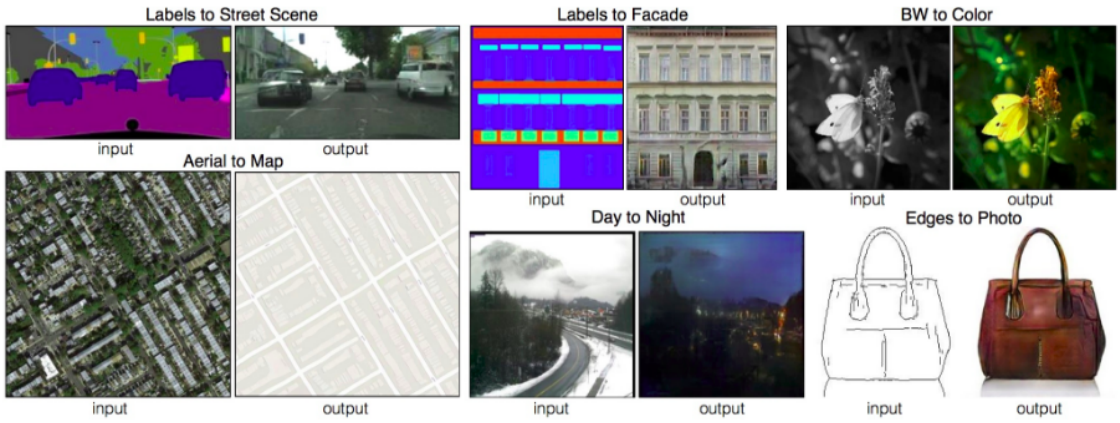


Figure 2: shows examples of Pixel-to-Pixel network [7].

1.3 Objective

This project is a subproject of the larger one which tries to minimize costs and resources spending for biomedical imaging examinations by finding where the nucleus are, classify whether they are cancerous or not, and if so how severe they are. However, this project will cover only data preparation or more specifically synthesizing fake training dataset. The reason why we have to synthesize training images is that getting the ground-truth or the locations of nucleus in the images is very tedious and expensive. Also, deep learning requires large amount of training images which is unlikely in biomedical data.

There are two necessary sets of data. The first data is the histological / nucleus images as shown in fig.1. The second data is the corresponding masks of the nucleus

images which indicate where the nucleus are in the images. Instead of getting a mask of an image by manually labeling like traditional approach, this project will try to synthesize the nucleus images from random masks. The reason that we do not do the opposite is that generating a mask is much easier than generating an image from scratch. Therefore, we will build a model, Optional Pixel to Pixel, that maps from the masks to the images. Then, we will synthesize fake masks. Lastly, we will use the synthesized masks to synthesize nucleus images using the Optional Pixel to Pixel model. The overview of this project is shown in fig.3.

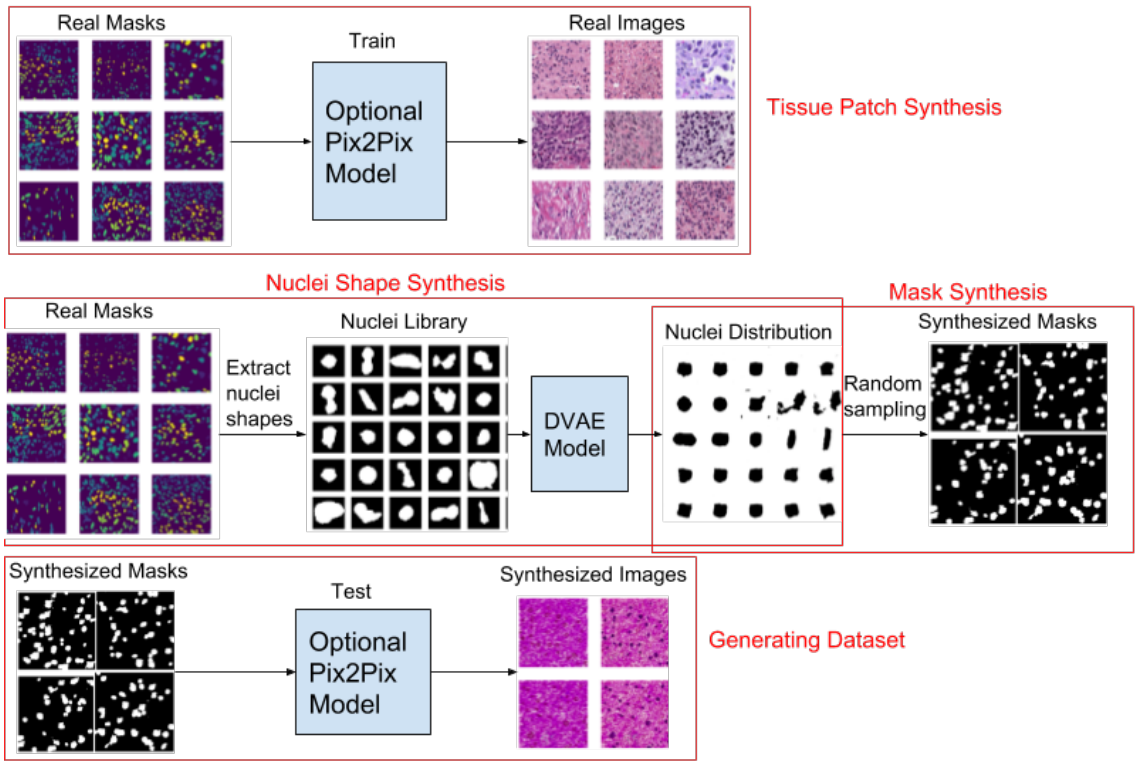


Figure 3: shows an overview of this project.

1.4 Contribution

The contributions of this project will be as follow:

1. Optional Pixel-to-Pixel Network which is an extension of Pixel-to-Pixel Network

2. Concept of option layer (in Optional Pixel-to-Pixel) which can be applied in many GAN networks

3. Model to synthesize nucleus images with masks

4. Cleaned annotated datasets for further uses

1.5 Structure

The report will start with foundations in section 2, which includes 2.1 Concepts required to understand the rest of the report, 2.2 Related Works, 2.3 Technologies and tools used in this project. Then, the datasets used in this project will be described in section 3. In section 4, we will introduce Option Pixel to Pixel model. Next, section 5 will show how to synthesize nuclei shapes, which are parts of masks. Then, in section 6, we will put the synthesized nuclei shapes together to create masks and show the results of using synthesized masks on Optional Pixel to Pixel. Lastly, the resources (source codes and datasets) information can be found in Appendix A.

2 Foundations

2.1 Concepts

2.1.1 Machine Learning

Machine learning is algorithms that let computer figure out how to solve problems itself without any explicit coding instruction. It is divided into 2 categories, supervised learning and unsupervised learning. Supervised learning is built (trained) by telling the machine what are the expected outcomes (labels) of the given inputs. This is divided further to classification, having discrete outcomes, and regression, having continuous outcomes. Whereas unsupervised learning is a technique of letting the machine find the underlying structure of the data without given any specific output/label [13].

Machine learning algorithm tries to find the best parameters/weights of a model through optimization algorithm (usually gradient descent). The loss function of a model is the function that determines how good the model is, e.g. mean squared error, and the loss of the model is back propagated throughout the model to update the weights.

2.1.2 Neural Network and Deep Learning

Neural Network is a type of machine learning which is designed to work as similar to human brain as possible. Neural network contains nodes which represent numbers and edges which represent linear combinations between the nodes. The output of each node will be applied nonlinear function, such as sigmoid or tanh, to make the network arbitrarily complex. Hidden layers are the layers between the input and the output, where the number of hidden layers is an open option to choose.

Deep Learning is a type of neural network where the number of hidden layers is large. Deep learning nowadays gets huge attention from researchers around the world since it outperforms all the machine learning algorithms in all areas as long as there are enough dataset. Deep learning algorithms can be separated into several types which are designed for each specific task such as convolutional neural network and recurrent neural network.

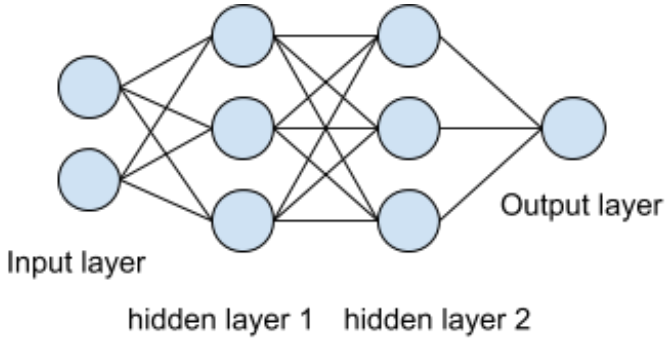


Figure 4: shows example of neural network with 2 hidden layers.

Activation Function is a nonlinear function in neural network whose role is to make to model more complex otherwise the whole model will only be simple linear model. Examples of activation functions are rectified linear unit (Relu), leaky rectified linear unit (leaky Relu), tanh.

Batch Normalization is a technique to rescale values in particular layers to their normal form. This usually come along with Relu activation function in order to keep half of the values above 0 and the other half below 0.

2.1.3 Convolutional Neural Network

Convolutional layer Convolutional layer is one type of neural network layers which is designed to handle spatial data such as image. It got the concept from the convolution filter in traditional computer vision where the filter tries to find interesting context in image e.g. edges. Convolutional layer is represented by filter kernels (size 1x1, 2x2, 3x3, ...). Each pixel of the filter is a learnable weight and the filter is applied to the input image using sliding window approach. The output of convolutional layer can be recognised as another image.

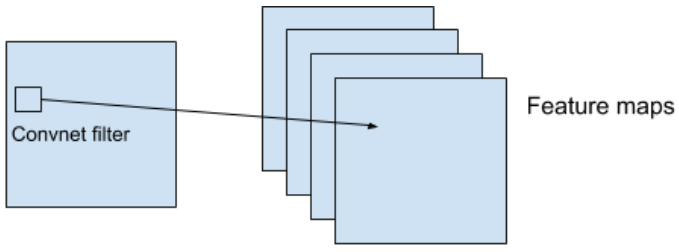


Figure 5: shows how convolutional layer performs. In this example, the convolution kernel produces 4 new channels.

Deconvo lutional layer Outputs from convolutional layers have equal or lower resolution than the inputs, which is applicable for classification task. However, in many cases, an output of the network has the same size as the input (segmentation) or larger size (super resolution). This is why deconvolutional layer is useful. Deconvolutional layer is a converse of convolutional layer which tries to upsampling an input image to higher resolution with filter kernels.

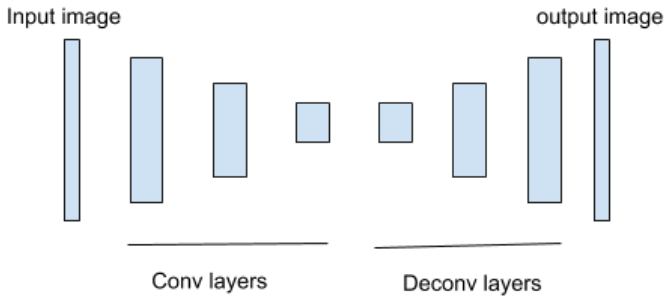


Figure 6: shows an example of network with 3 convolutional layers and 3 deconvolutional layers. The output of convolutional layers is smaller than the input. Conversely, the output of deconvolutional layers is larger than the input.

Convolutional Neural Network is one type of Neural Network which is designed specifically for spatial data such as images. This network consists of convolutional layers and fully connected layers.

2.1.4 Object Segmentation

Object segmentation is a problem in computer vision that tries to solve each pixel in an image belongs to which objects. For example, in our work, we want to find where the nucleus are in the image so output will be a mask of which the nucleus pixels are represented by 1 and the others by 0.

2.2 Related Works

2.2.1 Autoencoder [2]

Autoencoder is a simple type of neural network that takes any input, e.g. image, text, and tries to regenerate the input from its structure. As shown in fig.7, the structure consists of an encoder and a decoder. The encoder compress the input

and remove irrelevant information to a low dimension vector or a code. Then the decoder tries to regenerate the input from the code. For the encoder and decoder, the architectures are an open options and should be appropriately selected based on input data. This network uses mean squared error (MSE) between input and output as a loss function.

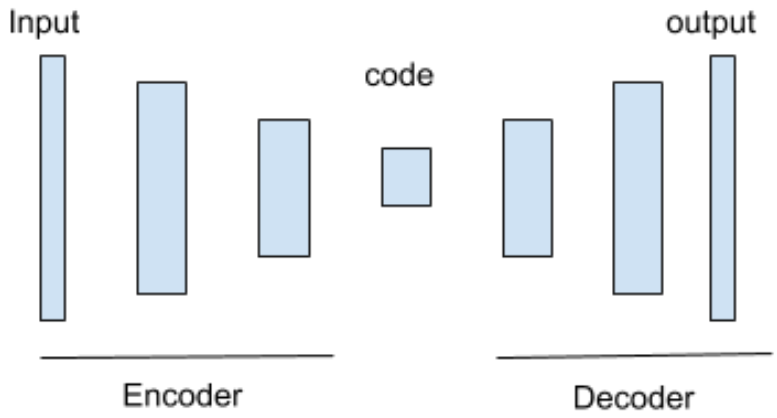


Figure 7: shows the structure of an Autoencoder Network.

2.2.2 Variational Autoencoder (VAE) [3]

VAE is an improved version of Autoencoder, shown in fig.8. The weakness of autoencoder is that values of the vector z are discontinuous so the value of the vector z cannot be randomly assigned, leading to Variational Autoencoder. In the encoder process, instead of generating only a vector z, it generates mean and variance vectors. Then, the network samples a vector z from those generated mean and variance vectors. This technique produces different vector z from the same input image. Consequently, the decoder network has to be robust enough to regenerate the same image from different z vectors. The loss function of this network is the sum of reconstruction error (MSE between input and output) and divergence between the latent vector distribution and the input distribution (KL divergence). The formula of the loss function is $L(\theta, \phi; x, z) = E_{q_\phi(z|x)}[logp_\theta(x|z)] - D_{KL}(q_\theta(z|x)||p(z))$.

2.2.3 Disentangled Variational Autoencoder (DVAE or βVAE) [9]

This is an improved version of Variational Autoencoder. The reason behind the improvement is that we want each dimension of the vector z (compressed input) to

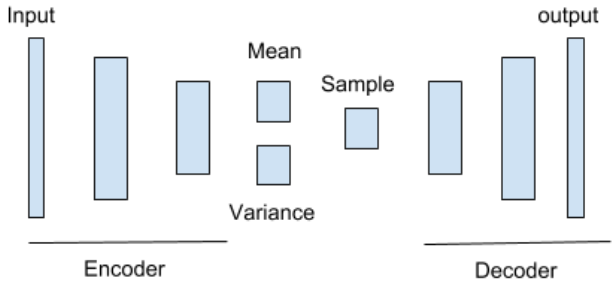


Figure 8: shows the structure of a Variational autoencoder Network.

represent 1 semantic appearance of the output image. In other words, a dimension may represent the size of the output, another dimension may represent the rotation of the output etc. We can accomplish this by multiplying KL loss term by Beta values (Beta > 1). The formula for the loss function is $L(\theta, \phi; x, z) = E_{q_\phi(z|x)}[logp_\theta(x|z)] - \beta D_{KL}(q_\theta(z|x)||p(z))$.

Disentangled Variational Autoencoder will be used in this project for encoding the nuclei shapes to latent vector and decoding the nuclei shapes back to their shapes.

2.2.4 U-Net: Convolutional Networks for Biomedical Image Segmentation [14]

U-Net is one type of the convolutional network architectures. The main characteristic of U-Net is having contracting path from encoder layer to corresponding decoder layer, which allows the network to capture context and precise localisation. This network outperforms the prior best method (a sliding window convolutional network) with significantly smaller training dataset.

We will use U-Net as a main architecture in most of our models in this project because as mentioned our models should be able to handle a small training dataset.

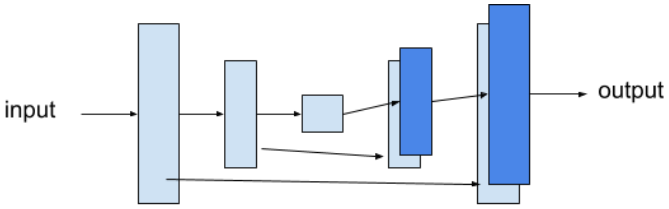


Figure 9: shows the structure of U Net.

2.2.5 Residual Network (ResNet): Deep Residual Learning for Image Recognition [5]

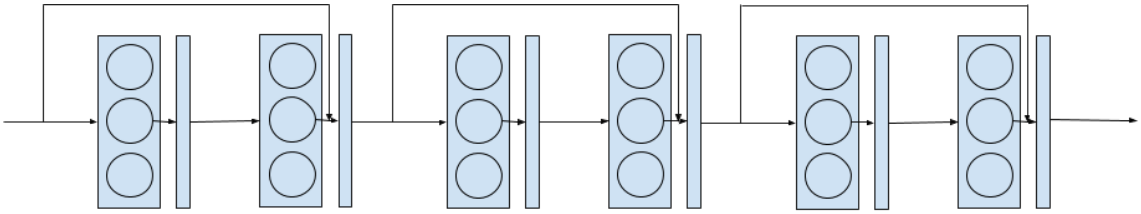


Figure 10: shows the structure of a residual network consisting of 3 residual blocks.

Prior to the invention of Resnets, deep neural networks have a problem of vanishing gradient (loss cannot be propagated to layers closed to input) and the deeper networks do not necessarily result in better performance. A residual network (fig.10) is a stack of residual blocks (fig.11) where every residual block contain a shortcut / a skip connection. The shortcut helps passing loss gradient to the early layers of the network and reducing the vanishing gradient problem.

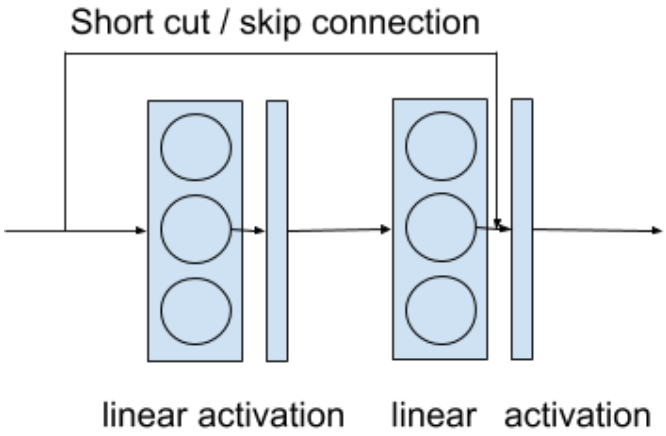


Figure 11: shows the structure of a residual block.

2.2.6 Generative Adversarial Network (GAN) [4]

One kind of unsupervised models where you have 2 models, generative model and discriminative model, playing game against each other (a minimax two-player game). The generative model tries to find a distribution underlying the input data (image, text etc.) from random noise and tries to generate the fake data that is as close to the real data as possible. The discriminative model tries to accurately classify

whether the input data is real or fake (generated from generative model). The two models are trained simultaneously. The final result of GAN is the generative model which is able to realistically generate the data.

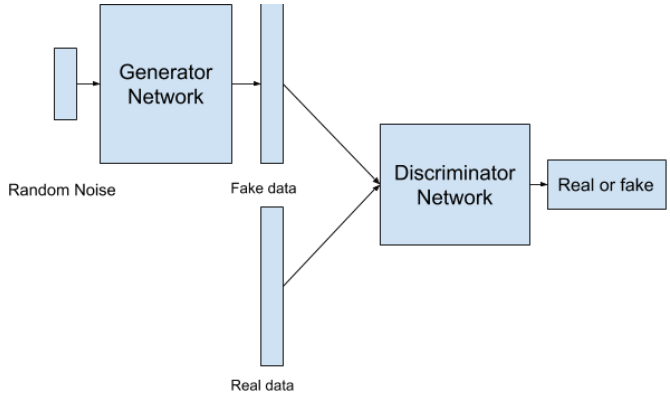


Figure 12: shows the structure of Generative Adversarial Network (GAN).

2.2.7 Conditional Generative Adversarial Network (CGAN) [12]

CGAN is a supervised model built on top of GAN where the label is concatenated to the noise and input and pass through a generative and a discriminative models. The model subsequently learn from the label and generate the data based on the input label.

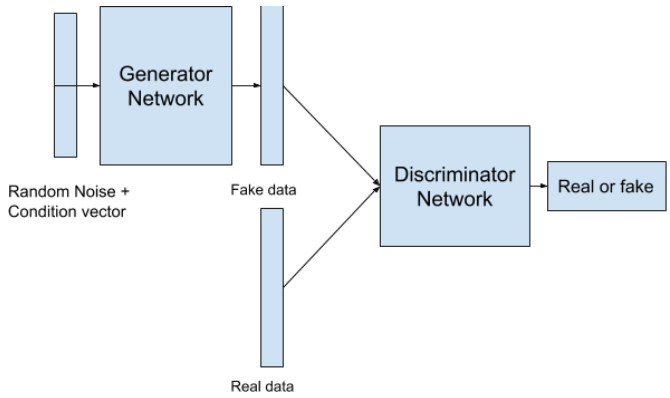


Figure 13: shows the structure of Conditional Generative Adversarial Network (CGAN).

2.2.8 Image-to-Image Translation with Conditional Adversarial Network (pixel2pixel) [7]

Pixel2Pixel adapts the concept of conditional adversarial network to image-to-image translation. In Pixel2Pixel, the input image is feeded to the generator to get a generated/fake image. Then, either generated image or input image with it label are feeded to discriminator. Please note that there is no noise in this network, so there is no variation in generated images. The generator network uses either Unet or ResNet architecture and the discriminator uses basic convolutional neural network.

2.2.9 Conditional CycleGAN for Attribute Guided Face Image Generation [11]

This paper uses CycleGAN, a version of Pixel to Pixel network, to translate from low resolution images to high resolution images. The output of the network does not only depend on the low resolution images but also a set of attributes related to facial appearances For example, if we input low resolution image is female with male attribute, the network will try to generate high resolution male images based on the input.

Our project is similar to this Conditional CycleGAN research where our project also has conditional on the network called Option Layer. However, our work is based on Pixel to Pixel network, not CycleGAN.

2.2.10 Unsupervised Histopathology Image Synthesis [6]

This paper uses unsupervised learning technique to synthesize histopathology images. It uses traditional computer vision techniques to extract patch from a real histopathology image, remove nucleus pixels (black dots in the image), then randomly put black dots as a nucleus to the zero-nucleus image. The locations of the new nucleus are kept as a mask. Then, the edited image are refined by GAN network. This network works well enough to synthesize fake training dataset for histopathology tasks, which improves the performance of the model compared to a solely real dataset.

We will use the idea of synthesizing nucleus onto an image from this paper for our project. However, using only traditional computer vision techniques is not efficient

and can be improved by a neural network.

2.3 Technologies and Tools

2.3.1 Python

Python is the most popular programming language for data science. It is high-level, general-purpose, interpreted, dynamic programming language. It allows programmers to focus more on concepts than syntax.[8]

2.3.2 Pytorch

Pytorch is an open source deep learning framework library in Python created by Facebook. This library has become more and more popular among deep learning researchers due to it simplicity and dynamic graph.

2.3.3 Jupyter lab / Jupyter notebook

Jupyter lab and Jupyter notebook are Python interactive Integrated development environment (IDE) where developers can code along with keep results shown on the notebook, just like a real notebook.

2.3.4 Google Colaboratory (Colab)

Colab is an online Jupyter Notebook on google drive which also provides Graphics processing unit (GPU) computation (K80) for free. This is very convenient platform for developers and researchers to try and build deep learning models.

3 Datasets Collection and Pre-processing

In this project, we used 3 main sources of datasets, which are 1.Kaggle's Data Science Bowl 2018 2.MICCAI 3.CRLM. The datasets consist of nucleus images and their corresponding masks indicating locations of nucleus. The primary dataset in this project is CRLM which is annotated by our lab and the others are supplementary. The details of each dataset are as follow:

3.1 Kaggle's Data Science Bowl 2018 [10]

This dataset is a part of Kaggle's 2018 Data Science Bowl competition. This competition is to find the nuclei in divergent images to advance medical discovery. The competitors were building image instance (nuclei) segmentation models from the provided datasets for $100,000 prize. This dataset contains nucleus images from different sources having diverse styles as shown in fig.14. (more info about the competition).

We will only use stage1 train.zip data, which consists of 670 nucleus images and their corresponding masks. The locations of nucleus or masks for each nucleus image are stored in separated file, one file per one nuclei, as in fig.15. In the masks, the location of the nuclei is represented by completely black values (intensity of 255).

Image Size	Count	Mean	SD	Mean per Pixel	SD per Pixel
256x256	334	28.56	21.30	0.00044	0.00032
603x1272	6	229.50	94.22	0.00030	0.00012
256x320	112	41.80	30.57	0.00051	0.00037
520x696	92	103.83	74.61	0.00029	0.00021
360x360	91	23.45	5.29	0.00018	0.00004
1024x1024	16	84.06	27.31	0.00008	0.00003
260x347	5	81.60	47.03	0.00090	0.00052
512x640	13	31.62	11.75	0.00010	0.00004
1040x1388	1	14.00	0.00	0.00001	0.00000
Total	670	43.97	47.93	0.00003	0.00003

Table 1: Statistics of the nucleus in the Kaggle dataset.

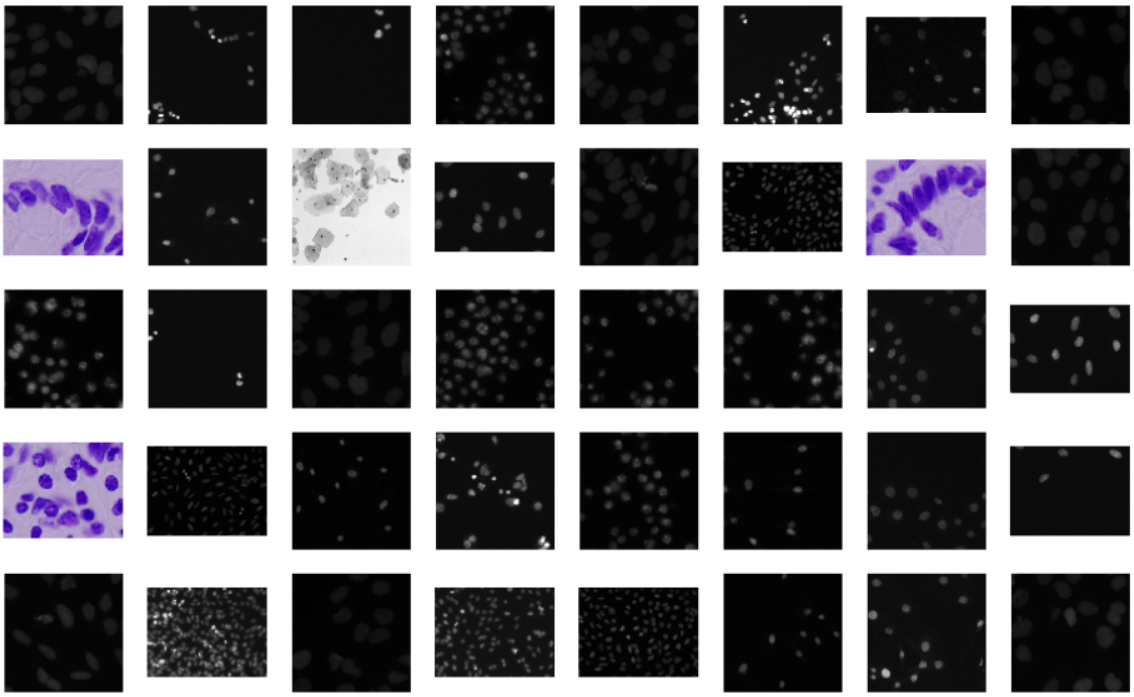


Figure 14: shows 40 sample images of Kaggle's Data Science Bowl 2018 dataset.

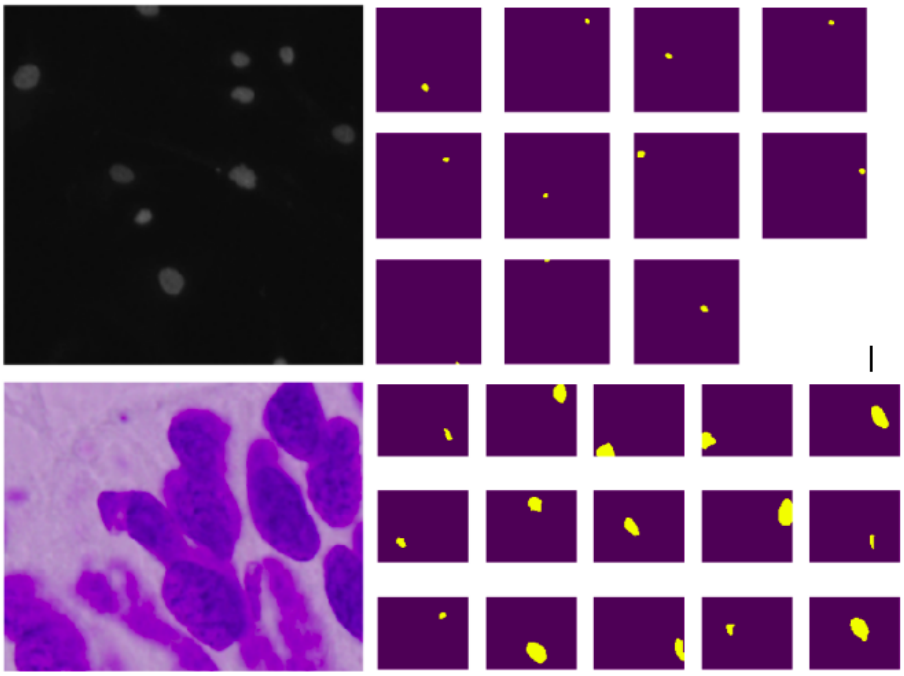


Figure 15: shows 2 sample image and their masks of Kaggle's Data Science Bowl 2018 dataset.

As shown in table 1, there are 9 different sizes of the images in this dataset: 334 images with 256x256 pixels, 112 images with 256x320 pixels, 5 images with 260x347 pixels, 91 images with 360x360 pixels, 13 images with 512x640 pixels, 92 images with 520x696 pixels, 6 images with 603x1272 pixels, 16 images with 1024x1024 pixels, 1 images with 1040x1388 pixels. All of them have 4 channels and are in PNG format.

The average of nucleus per image in this dataset is 43.97 with 47.93 standard deviation and the average nucleus per pixel is 0.00003 with 0.00003 standard deviation. More details about the number of nucleus in this dataset is in table.1.

3.2 MICCAI 2018: Computational Precision Medicine [1]

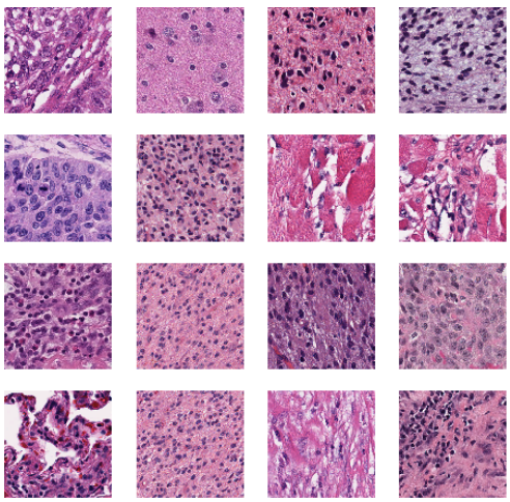
Medical Image Computing and Computer Assisted Intervention Society (MICCAI) is a non-profit corporation, whose goal is to build a society for experts in multi disciplinary areas. This year (2018), MICCAI held a "Computational Precision Medicine" competition where researchers / competitors with the best medical image segmentation model take the prize.

This dataset consists of 32 images with pathological images and their corresponding masks as shown in fig.16. The masks tell where the nucleus are on the images by using integer numbers (1,2,3,..) as ids for each nuclei on each image. Consequently, the masks have different color representing nucleus locations due to different ids (fig.16b).

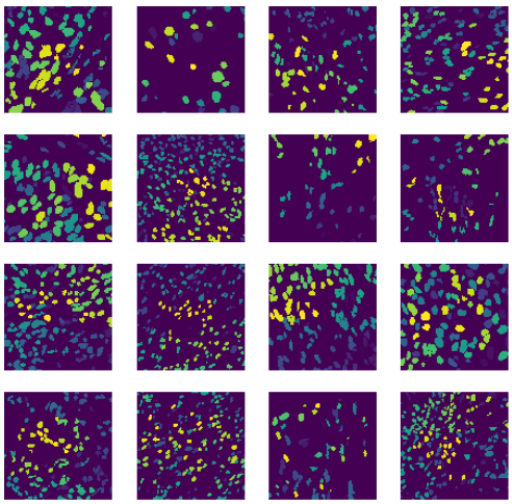
There are 3 different sizes of the images in this dataset. The first size is 500x500 pixels with 4 channels, accounting for 19 images. The second is 600x600 pixels with 3 channels, accounting for 2 images. The last is 600x600 pixels with 4 channels, accounting for 11 images.

The average number of nucleus of 500x500 pixels is 102.37 (or 0.0004 per pixel) with 46 (or 0.0002 per pixel) standard deviation. The average number of nucleus of 600x600 pixels is 143 (or 0.0004 per pixel) with 47.78 (or 0.0001 per pixel) standard deviation. The original format of this dataset is PNG.

The average of nucleus per image in this dataset is 118.88 with 50.82 standard deviation and the average nucleus per pixel is 0.00033 with 0.00014 standard deviation. More details about the number of nucleus in this dataset is in table.2.



(a) Samples of MICCAI CPM's images



(b) Samples of MICCAI CPM's masks

Figure 16: shows samples of MICCAI's dataset. Please note that the intensities of the mask are not the same due to the different number in each pixel.

Image Size	Count	Mean	SD	Mean per Pixel	SD per Pixel
500x500	19	102.37	46.00	0.00041	0.00018
600x600	13	143.00	47.78	0.00040	0.00013
Total	32	118.88	50.82	0.00033	0.00014

Table 2: Statistics of the nucleus in the MICCAI dataset.

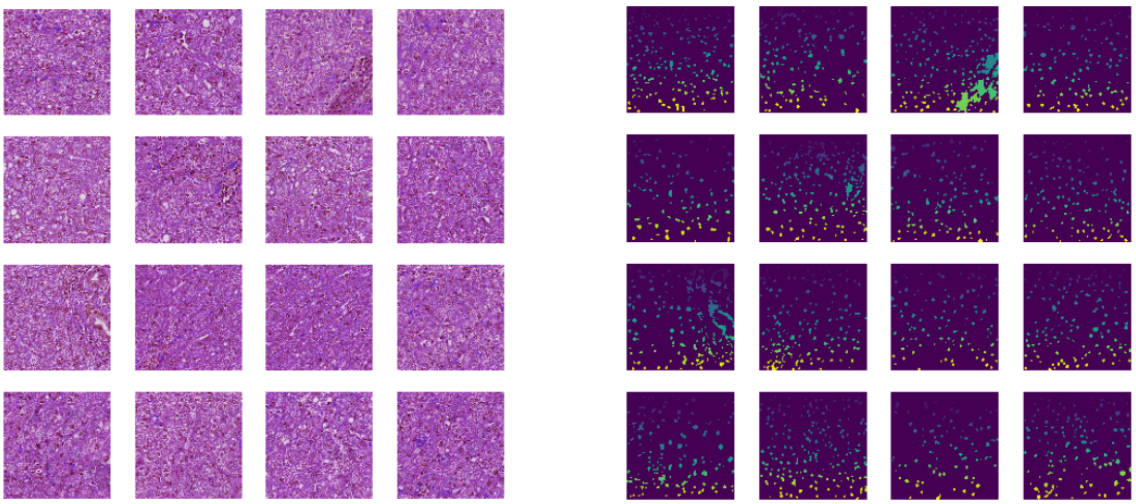
3.3 Colorectal Cancer Liver (CRLM)

Colorectal Cancer Liver (CRLM) is a normal liver cell dataset. The original dataset is in whole slide image and is cropped and annotated by our lab. The original annotations are in JSON format indicating coordinates (X-Y) of boundaries of nuclei locations.

We also built an algorithm to convert the X-Y boundary coordinates to mask images. The algorithm firstly fills boundaries' gaps (the X-Y coordinates are not connected and have gaps between two consecutive points). Then, it draws lines between left-right, top-bottom boundaries, resulting in masks on nuclei locations.

This dataset consists of 114 nucleus images and masks are as shown in 17 and 18. The masks are in the same format as in MICCAI CPMN's dataset where each pixel in the masks represents an id of the nucleus by integer number. So, the mask images have different dot intensities. This dataset have a consistent format with 1000x1000 pixels and 3 channels for all of them, and the format is PNG.

The average of nucleus per image in this dataset is 129.2 with 21.34 standard deviation and the average nucleus per pixel is 0.00013 with 0.00002 standard deviation. More details about the number of nucleus in this dataset is in table.3.



(a) Samples of CRLM's images

(b) Samples of CRLM's masks

Figure 17: shows samples of CRLM's dataset.

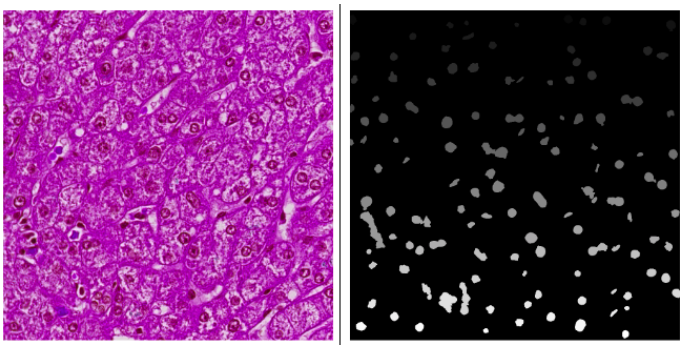


Figure 18: shows one sample of CRLM's dataset. Please note that the intensities of the mask are not the same due to the different number in each pixel.

Image Size	Count	Mean	SD	Mean per Pixel	SD per Pixel
1000x1000	114	129.20	21.34	0.00013	0.00002
Total	114	129.20	21.34	0.00013	0.00002

Table 3: Statistics of the nucleus in the CRLM dataset.

4 Tissue Patch Synthesis

In this section, we will build a model that synthesizes tissue patches (nucleus images) from masks. The model will be trained by inputing masks and having nucleus images as groundtruths. We will also introduce a new network called "Optional Pixel to Pixel" which is an extension of the plain Pixel to Pixel network.

4.1 Generating Nuclei Image with Pixel to Pixel

Pixel 2 Pixel or "Image-to-Image Translation with Conditional Adversarial Networks" is a state of the art algorithm for image to image translation tasks. It gives promising results for many problems such as converting day to night image, converting sketch to image, as shown in fig.2. In our problem, we want to convert mask images to nucleus images which sound reasonable to be done by this network.

As in fig.19, Pixel to Pixel contains 2 main parts, a generator network and a discriminator network. The generator network takes mask image as an input and produces generated fake nucleus image that looks as similar to the real nucleus image as possible. Then, the discriminator network takes either generated fake nucleus image or real nucleus image attached by mask and classifies whether the

input is real or fake. The loss of the discriminator network is then propagated back to update all parameters.

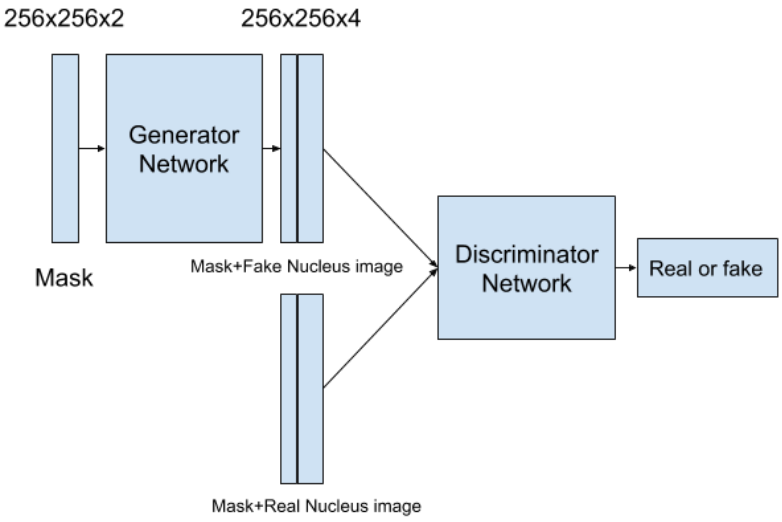


Figure 19: shows the structure of Pixel to Pixel Architecture.

4.1.1 Architecture

We used U-Net for the generator network and CNN for the discriminator network. The detail of the network is the same as that of Optional Pixel to Pixel network and will be discussed in section 4.2.1

4.1.2 Results and Performance

As you can see in the fig.20, the results from the pixel to pixel network looks quite well. However, our dataset has various nucleus image types (e.g. different background colors) but the network converges and produces only 1 type (dark pink background).

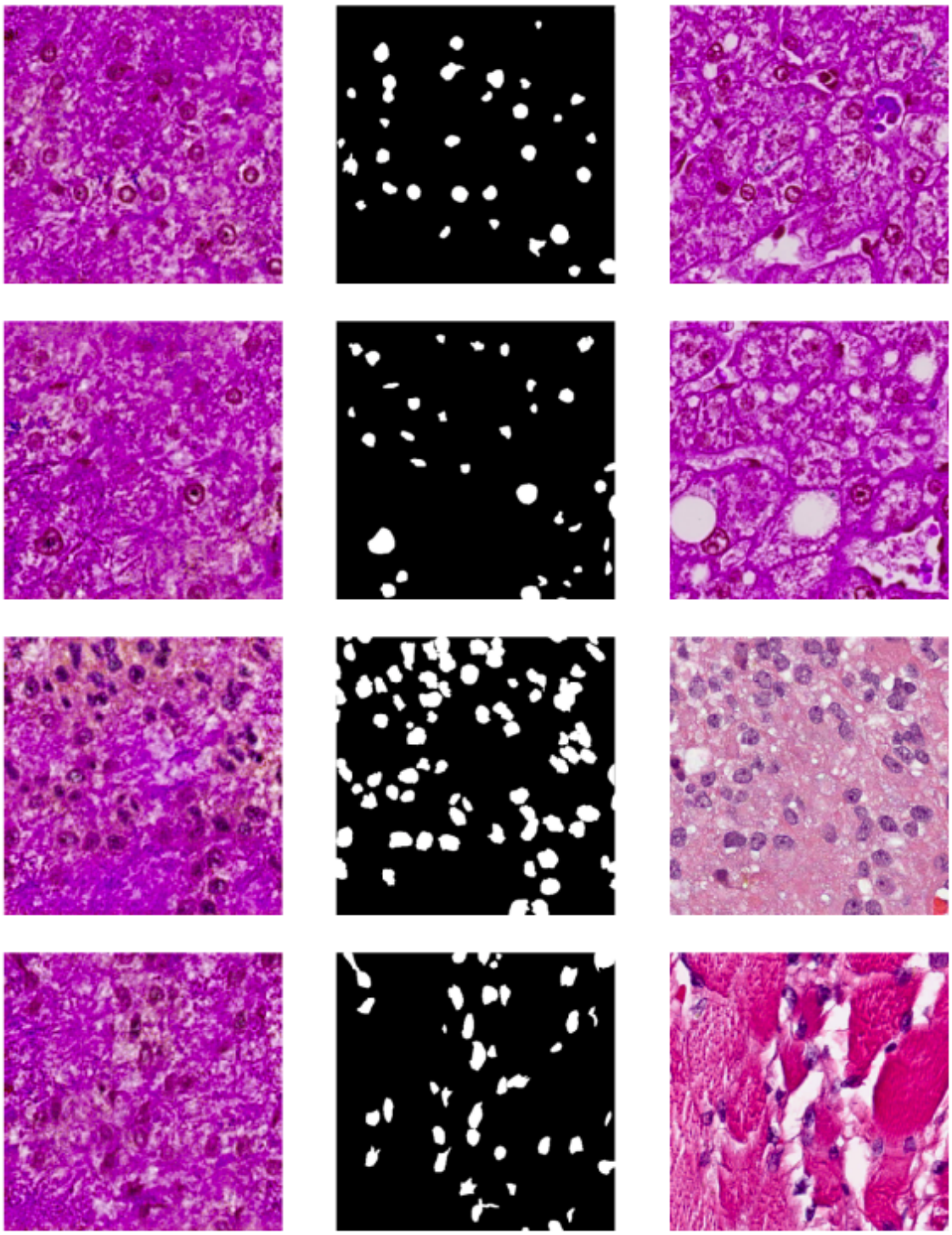


Figure 20: shows results from Pixel to Pixel network on training data. Left is generated image, middle is the input mask, right is the real image.

4.2 Optional Pixel to Pixel

The plain Pixel to Pixel method works quite well in generating nucleus images. However, it cannot handle the variations in our datasets since it does not contain any random noise or extra variable. On the other hand, the Optional Pixel to Pixel network, as shown in fig.22, introduces an option layer which specify which type of images is being generated. This new layer is added to both generator and discriminator by adding extra channel to the mask with all pixels having the same identical value e.g. all 1 for one dataset and all 2 for another dataset, as shown in fig.21. This option layer will let the generator knows which type of image is desired and generate the right kind. Also, the option layer in discriminator will let the discriminator knows which type of image is being considered, classify more accurate and converge faster.

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

2	2	2	2
2	2	2	2
2	2	2	2
2	2	2	2

Figure 21: shows option layers of size 4x4 with option 1 and option 2.

4.2.1 Architecture

We selected U-Net for the generator of our Optional Pixel to Pixel network as our dataset is not large enough for more complicated network like ResNet, which will cause overfitting with our dataset. Moreover, U-Net works quite well with the small dataset due to skip connections between encoder and decoder layers. The U-Net consists of 8 layers of encoder and 8 layers of decoder with Relu and Batch normalization. The detail of the generator can be found in appendix D.1.

As for discriminator, we used 5 layers of convolutional layers with leaky Relu and batch normalization. The output is a 2D image of 30x30 pixels where each pixel has probability of whether the input image is real or fake. The final result of the network is an average of all the pixels in the output. The detail of the discriminator

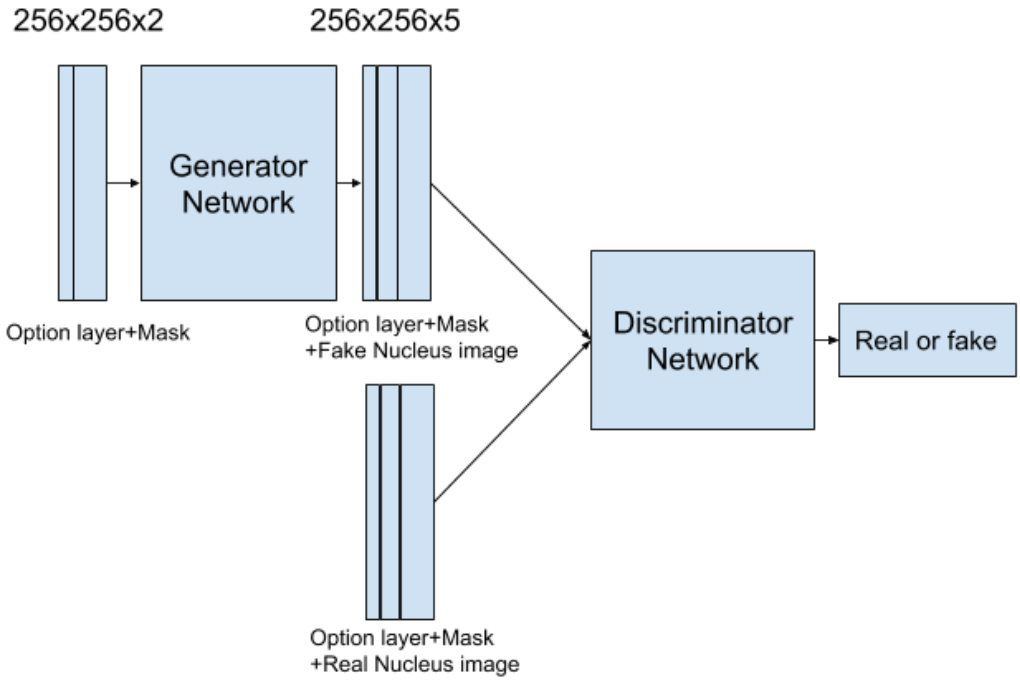


Figure 22: shows the structure of Optional Pixel to Pixel network.

network can be found in appendix D.2.

4.2.2 Result and Performance

As you can see in the fig.23, the optional pixel to pixel network works quite well and can also handle the variations within the dataset. The first column is the synthesized images with option 1 and the second is those with option 2. The most right column is the original images. The first two rows of the original images are option 1 and last two rows are option 2. Obviously, the first column tends to be more similar to the option 1 images and the second column tends to be more similar to the option 2 images.

4.3 Conclusion

The optional pixel to pixel network does well for histology image synthesis in our project. It handles variations better than the plain pixel to pixel network version, which produces only 1 type of images even if input is changed. Hoover, this op-

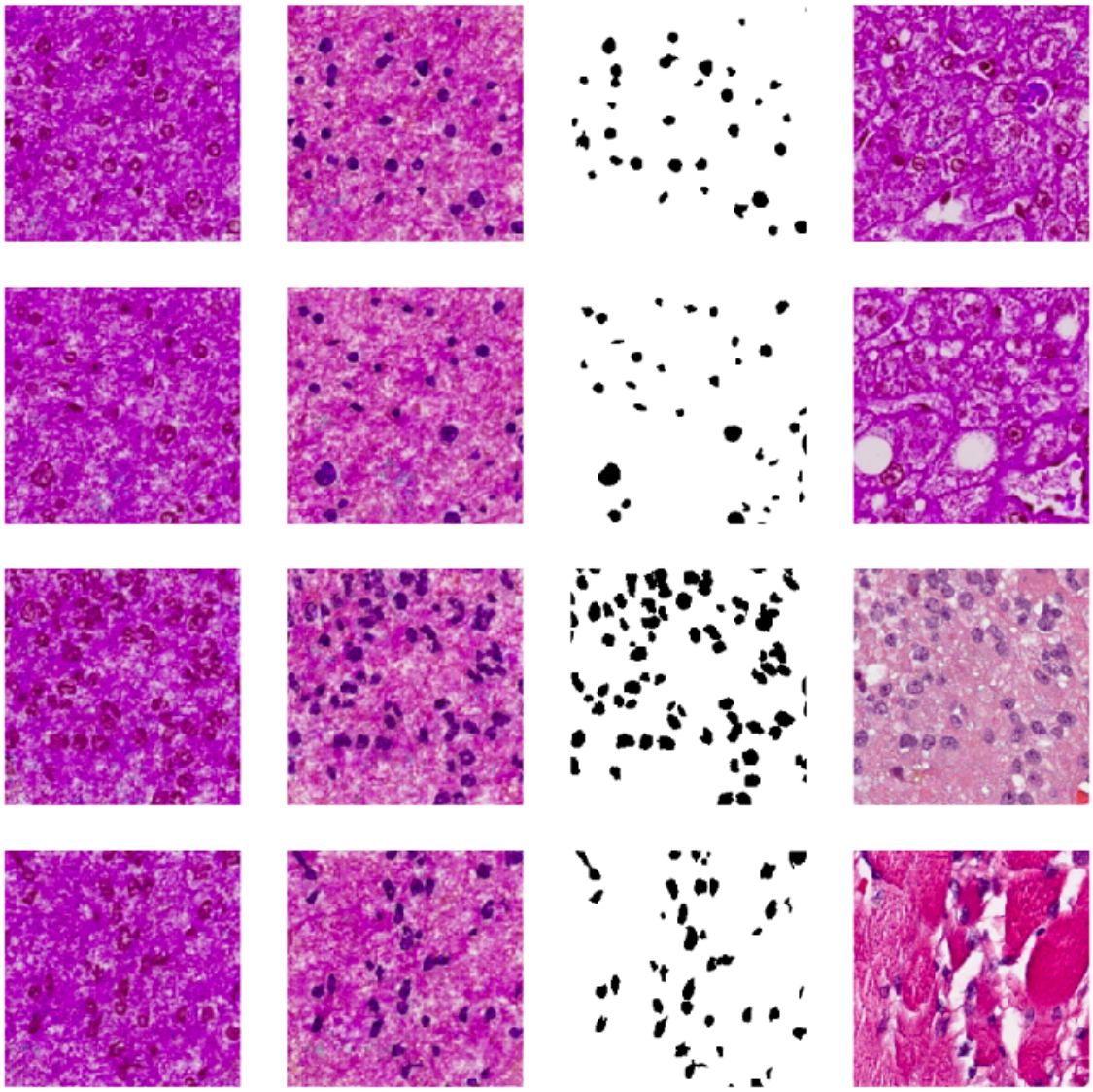


Figure 23: shows qualitative performance on training data, First two columns are generated images (option 1 and option 2), Third column is the input/mask, Last column is the real nucleus image.

tional pixel to pixel network requires manual separation of image types which might be tedious if the number of types is huge or might be impossible if the types are unknown. The possible improvement for this network might be a network that can automatically handle the types of images or integration of 2 models (Mask Synthesis section 6 and Nucleus image Synthesis section 4). We hope this model is useful in handling lacking training dataset problem in biomedical imaging tasks.

5 Nuclei Shape Synthesis

This section will discuss how to build a model to synthesize nuclei shapes. We will first introduce a nuclei library. Then, we will describe Disentangled Variational Autoencoder (DVAE) which take the nuclei library as an input to build a model to synthesize nuclei shapes. Lastly, we will show how we select latent vector size for encoding the nuclei shapes.

5.1 Nuclei Library

This is a library of nucleus, fig.24, which contains all nucleus extracted from all the datasets we have. We will later use this library to generate mask images.The size of the images in the library is 64x64 pixels. If the original size is larger than 64x64 pixels, it is resized. If it is smaller, it is padded. This library also filter out too small nucleus (the multiplication of row and column pixels $< 800\ pixels^2$) and those with incomplete shape (nucleus on the boundary of the images).

5.2 Disentangled Variational Autoencoder (DVAE)

"Disentangled Variational Autoencoder (DVAE)" is selected for nuclei shape synthesis. DVAE will find the distribution of all the shapes in the nuclei library using a deep learning network. The first part of DVAE is an encoder network which will firstly encode all the nuclei shape (64x64 pixels) into 2 small vectors (10 dimensions), mean and variance. It will, then, sample another small vector called "latent variable" based on given mean and variance vectors. The latent vector represents most of the variations of all the nucleus, so theoretically, all of the nuclei shapes can be reconstructed from the latent vector. Another part of DVAE is decoder network,

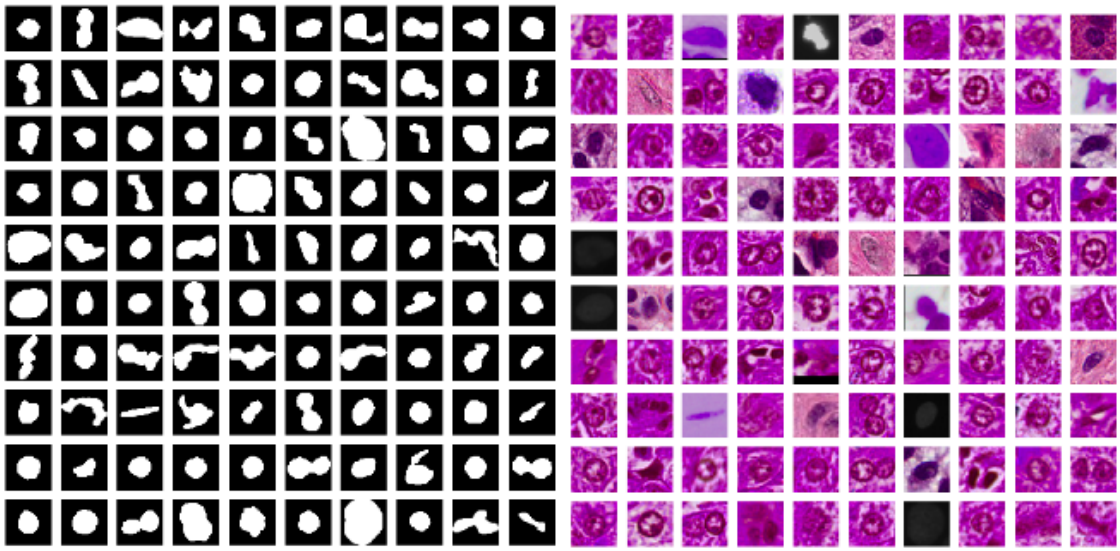


Figure 24: Left shows Sample nuclei shapes and their corresponding images extracted from the whole images.

which will regenerate the nuclei shape from the latent vector. At the end, we will get the latent vector which will represent all of the higher dimensions of nuclei shapes and the decoder network which can decode the latent vector to the corresponding nuclei shape (this may loss some information since it does dimensionality reduction).

One thing to mention is that each dimension in the latent vector carries distinguished semantic meaning, where different dimensions will change different properties of the shapes. For example, if you want a bigger shape, you just increase a value in a specific dimension of the input.

5.3 Architecture

The Variational Autoencoder consists of encoder network that encodes the input to mean and variance vectors, which can be sampled as a latent vector, and decoder network that decodes the latent vector back to the image. In our project, we will use convolutional layers and deconvolutional layers for encoder and decoder, respectively, as they are better for spatial data than normal fully connected layers. The details for encoder and decoder networks are as follow.

Encoder The encoder part takes 64x64 nuclei shape image and it, then, goes through 4 convolutional layers with batch normalization and Relu. The output of 4th

convolutional layer is converted to 1D vector and is fed to 2 fully connected network of size 10. The 2 outputs of fully connected layers act as mean and variance of the latent vector, which is sampled from these 2 vectors using normal distribution. The fig.25 visualize the encoder network and more details about the network architecture is in appendix C.1.

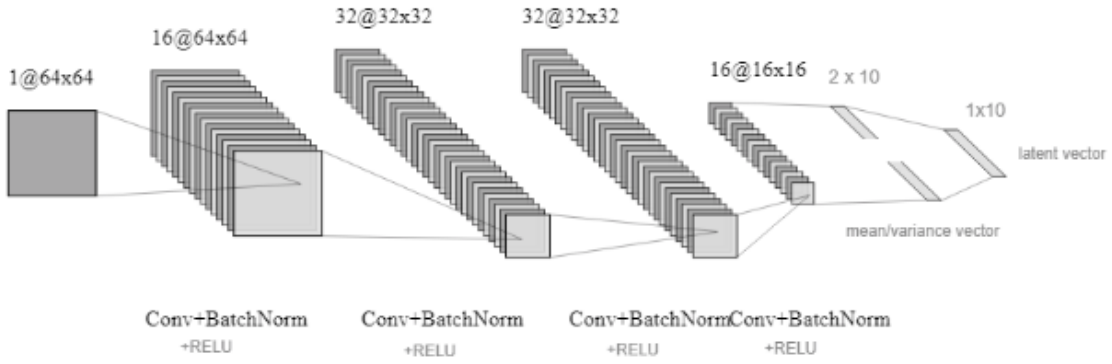


Figure 25: Left shows encoder network of the DVAE.

Decoder The decoder network decodes from latent vector to shape image. Started with latent vector of size 10, it is then feeded to 2 fully connected layers with Relu of size 100 and 1096 in order. The vector of size 1096 from the output of 2nd fully connected layer is converted to 16 channels of 16x16 image and feeded to 4 deconvolutional layer with batch normalization and Relu. The fig.26 visualize the decoder network and more details about the network architecture is in appendix C.2.

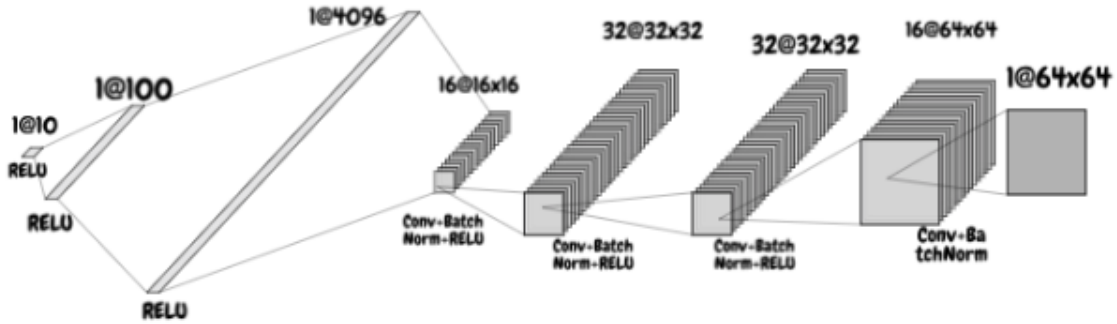


Figure 26: Left shows decoder network of the DVAE.

The output of the last convolutional layer is an image of size 64x64 with single channel or the nuclei shape image we want. The latent vector size is a hyperparam-

eter which is adjustable. The next section 5.4 will show why we selected the latent vector size of 10 dimensions.

5.4 Finding the best latent vector size

We want a latent vector size that best represents the data while keeping the size as small as possible. The larger the size, the more data is captured. However, smaller dimensions requires less storage as well as it is more simple and easier to interpret.

Mean squared error (MSE) between regenerated nuclei shapes and original nuclei shapes and the loss function of the network (MSE + KL divergence) are selected as metrics to measure how well each latent vector size can regenerate original images. We started with sizes [10, 14, 18, 22, 26, 30] dimensions with 200 epochs each to find the tendency of the performance.

Latent vector size	Loss (mse+KL)	MSE
10	142.888	35.758
14	177.121	36.166
18	180.111	37.039
22	163.292	36.651
26	153.452	38.680
30	164.498	60.691

Table 4: shows loss (mse+KL) and MSE for each specific latent vector size.

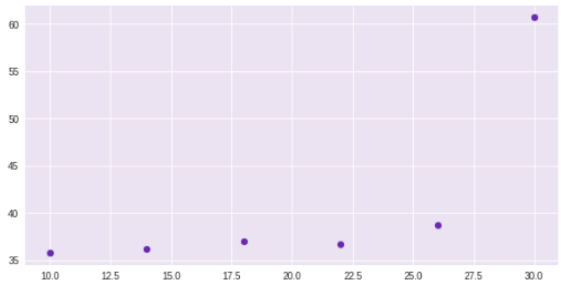


Figure 27: y-axis is the final mse, x-axis is the latent vector sizes.

According to the final error in the fig.27, the first 5 sizes have no difference and significantly better than the largest size. In other words, size of 10 gives roughly the same performance as other bigger sizes, This may be due to the fact that it is

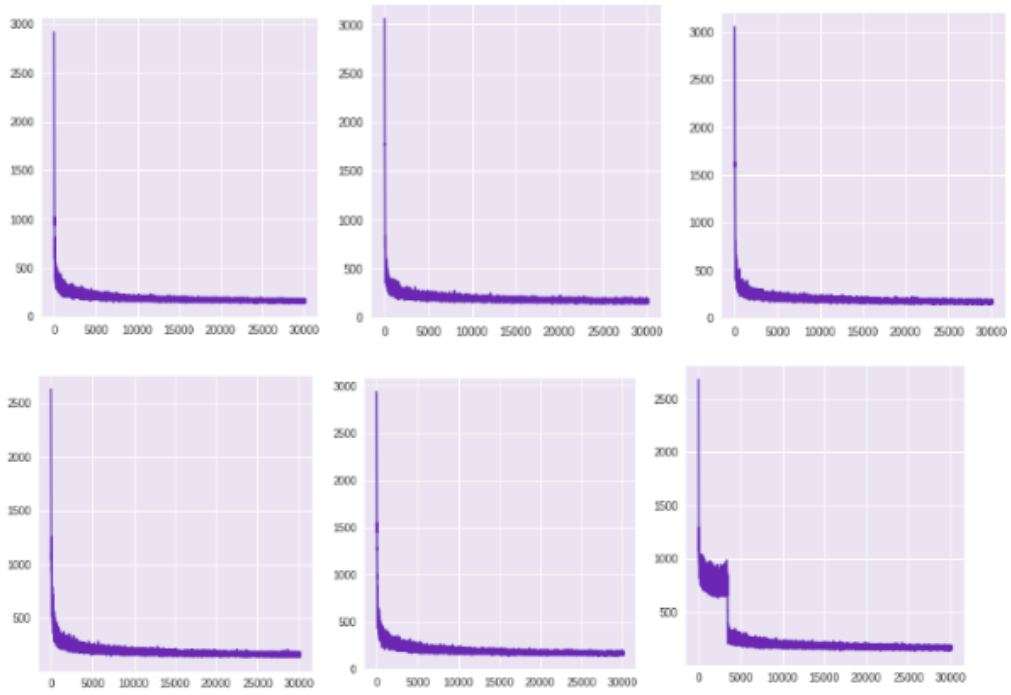


Figure 28: y-axis is the loss, x-axis is the epochs for sizes of 10, 14, 18, 22, 26, 30 (top to bottom) respectively.

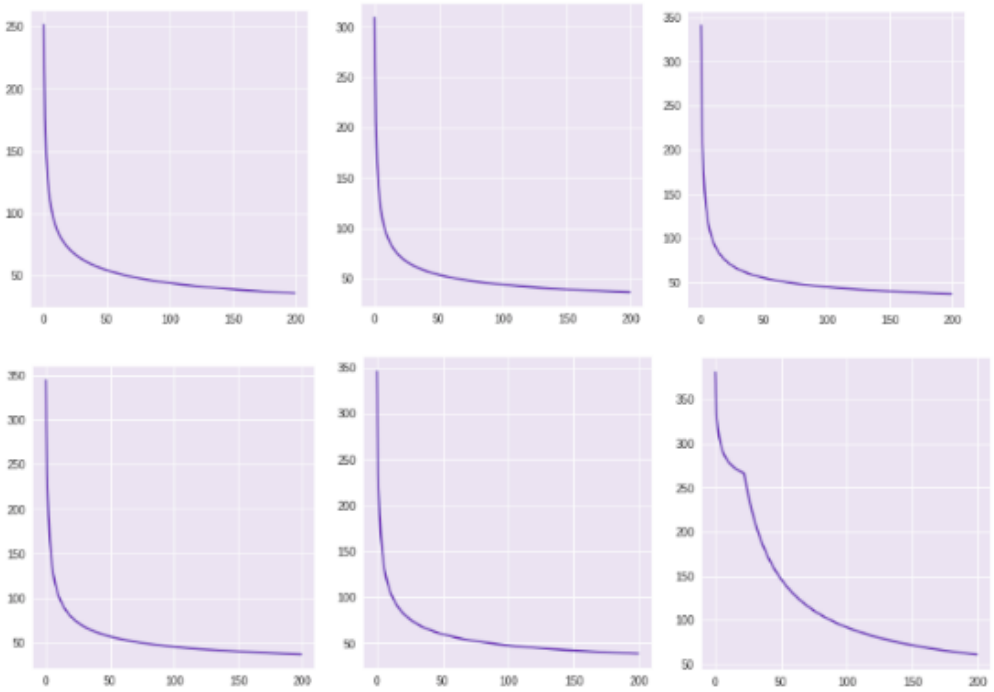


Figure 29: y-axis is the mse, x-axis is the epochs for sizes of 10, 14, 18, 22, 26, 30 (top to bottom) respectively.

easier to converge. To make sure the result are reliable, fig.28 and fig.29 show the graphs of loss and mse per epoch. These graphs verify that the models converge enough in order not to give bias decision. Based on mentioned reasons, size of 10 is the most preferable among the experimented sizes since it is smallest but still give comparable performance.

To confirm that latent vector size of 10 is the best of choices, we will rerun the experiment with 300 epochs (bigger epochs) on the sizes of 8, 10, and 15. The final results are shown in table.5, fig.30, fig.31 and fig.32.

Latent vector size	MSE
8	34.690
10	32.737
15	33.083

Table 5: shows mean squared error of the size 8, 10, 15.

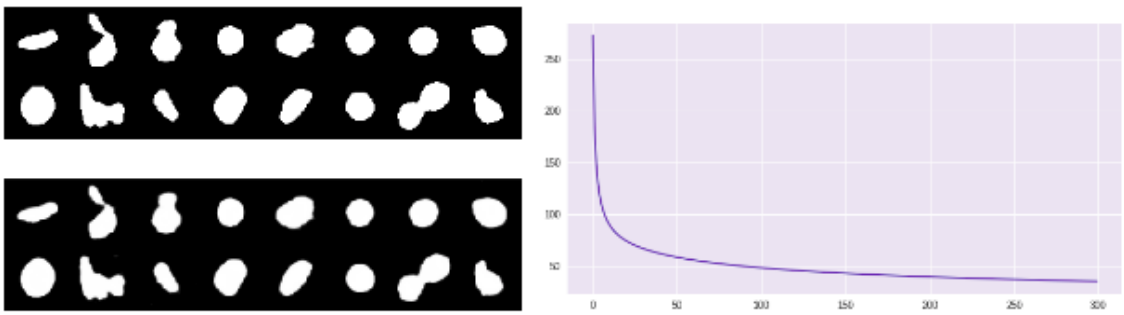


Figure 30: the left shows the original shapes (top) and the regenerated shapes (bottom) and the right shows the mse trend by epochs for the size of 8.

According to the table 5, the size of 10 still gives the lowest loss. Also, the regenerated shapes of the size 10, shown in fig.31, are reasonable good and have no significant difference from the size 15, shown in fig.32. Therefore, we will continue using the latent vector of size 10 for the later experiments.

5.5 Results and Performance

We now use 300 epochs to train the DVAE model with a latent vector size of 10 to make the final model. The graph in fig.33 shows the convergence of the model. This

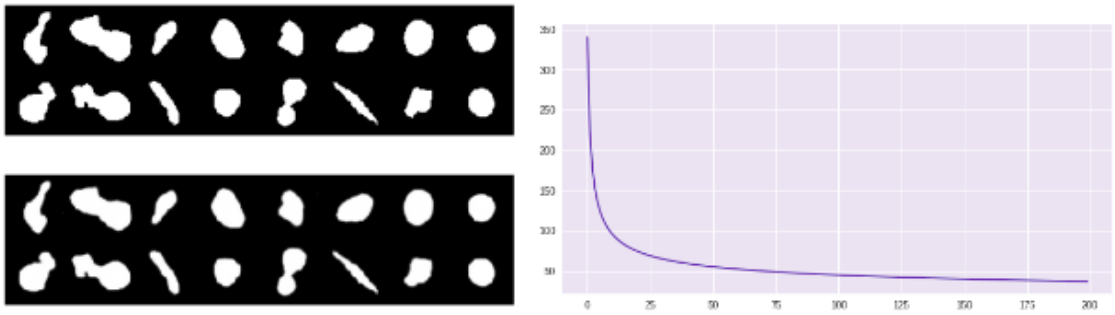


Figure 31: the left shows the original shapes (top) and the regenerated shapes (bottom) and the right shows the mse trend by epochs for the size of 10.

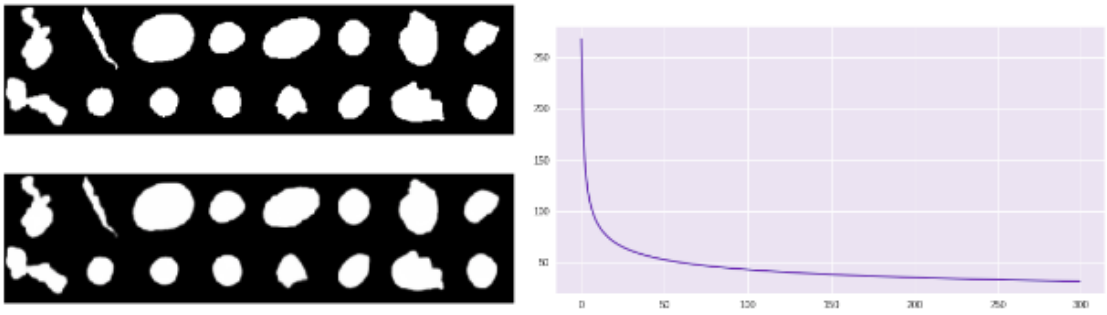


Figure 32: the left shows the original shapes (top) and the regenerated shapes (bottom) and the right shows the mse trend by epochs for the size of 15.

make sure that the model is trained enough and already converged. The final mean squared error of the training data of size 64x64 is 33.65 or 0.82%, meaning in the image of size 64x64, there will be around 33.65 pixels that is incorrect (0 instead of 1 and 1 instead of 0). fig.34 also shows the comparison between the original shapes and the regenerated shapes from the model.

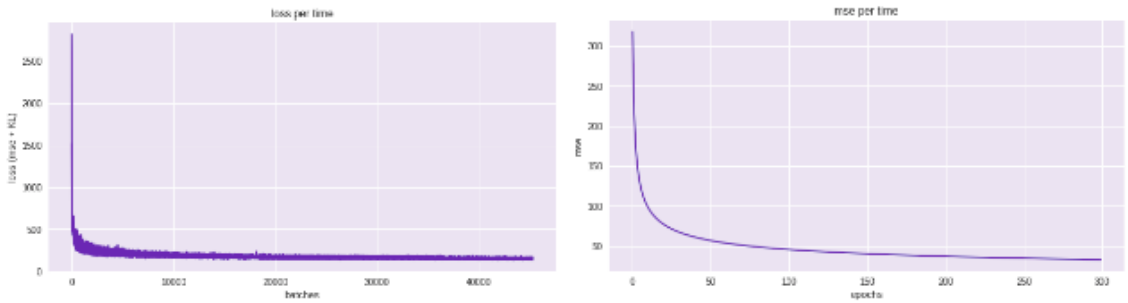


Figure 33: left: loss per batches, right: mean squared error per epochs.

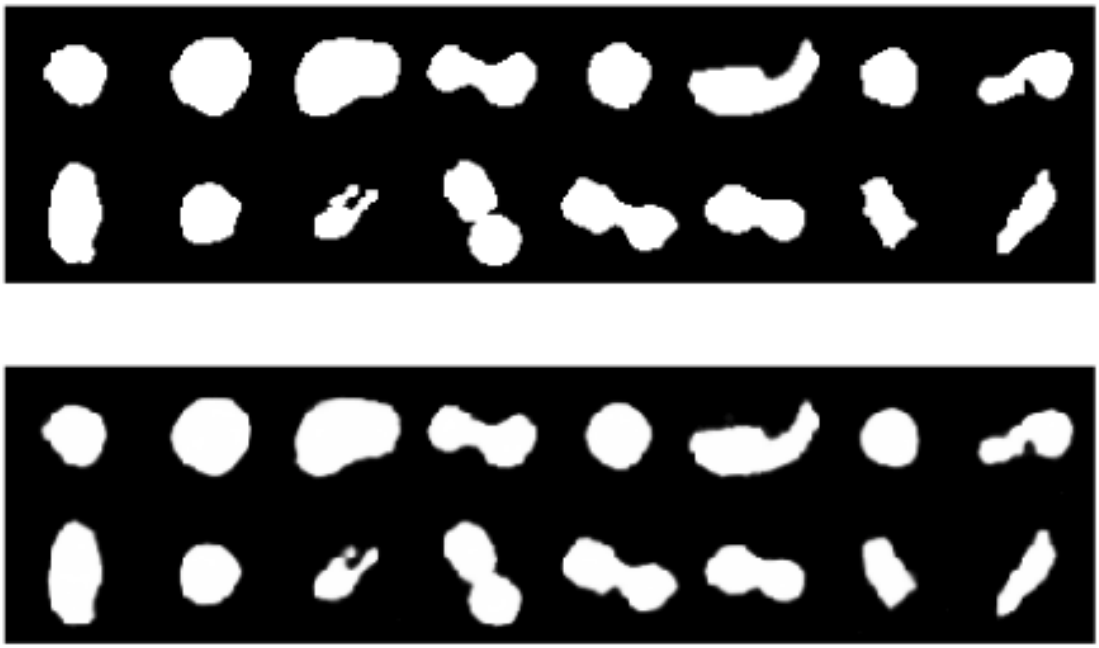


Figure 34: left: top is the original shape, bottom is the regenerated shape.

The fig.35 shows how nuclei shape changes when adjusting the value in each dimension. For example, dimension 0 changes the size of nuclei and dimension 9 rotates the nuclei.

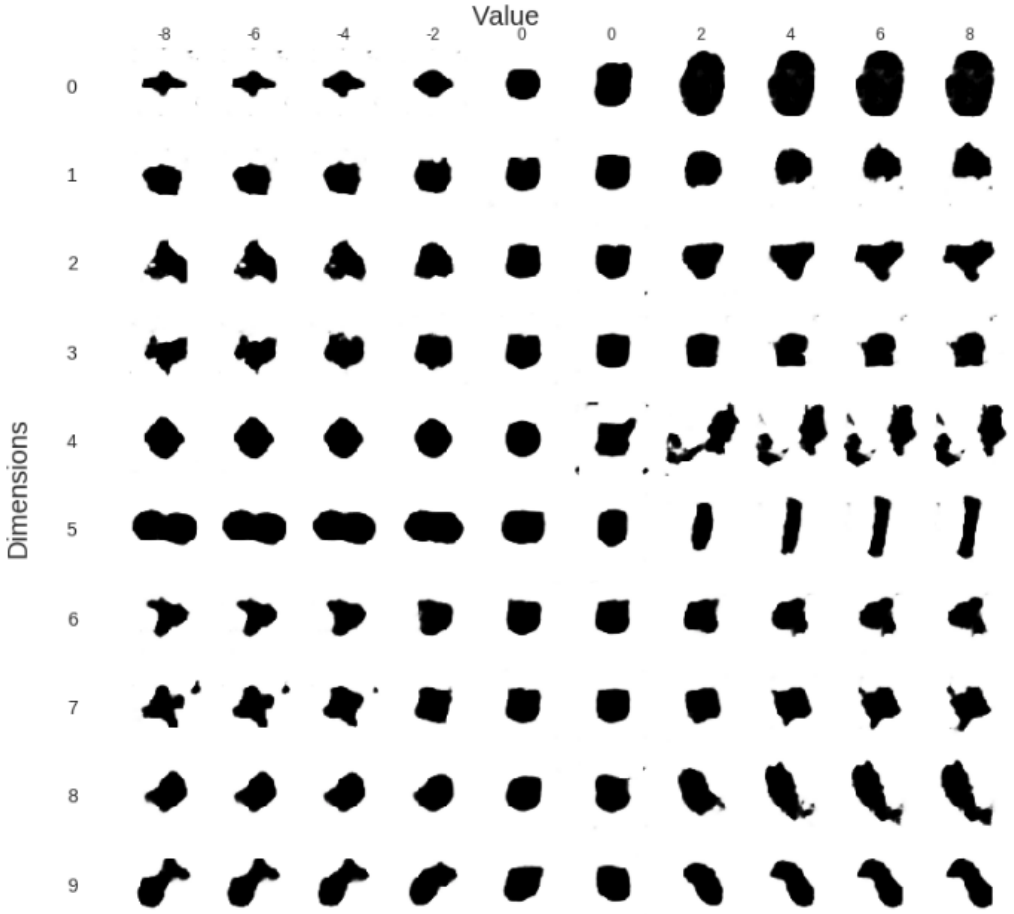


Figure 35: shows how nuclei changes when adjusting value in each dimension and fix the others as 0.

5.6 Conclusion

In this section, we have implemented nuclei library and used the library to train DVAE model. We got great results from the DVAE model as the regenerated images and the original images are almost the same. So, instead of storing all the images, we can just use the model and a random latent vector to generate an image. DVAE has another great property which is semantic meanings on each dimension of its latent vector. We also showed why 10 dimensions of latent vector is the best for our task. The DVAE model from this section will be used in the section 6 for generating masks.

6 Mask Synthesis

This section will show how to synthesize nucleus masks that will later be used as inputs for the Optional Pixel to Pixel model, in section 4, to synthesize nucleus images.We will first analyze all the masks in CRLM dataset using statistical analysis. Then, we will show how to synthesize a mask from synthesized nucleus from the DVAE model. Lastly, we will show the results of using synthesized masks for the Optional Pixel to Pixel model.

6.1 Analysis on mask images

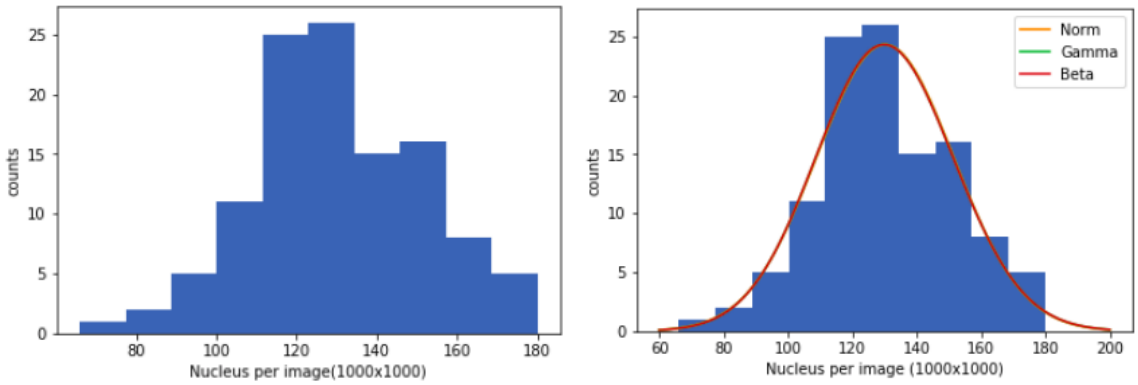
In this section, we will analyze the nucleus images and their masks in CRLM dataset in order to know how to generate nucleus masks realistically. Statistical analysis will be used to find underlying structure of the mask images such as number of nucleus per image or sizes of each nuclei. We will also find the distributions that best describe the structures of the mask images.

6.1.1 Nucleus per image

As you can see in the fig.36a, the distribution of nucleus per image look similar to normal distribution. We confirm this by plotting normal, beta, and gamma distributions on to the histogram as in fig.36b. The parameters for each distribution are calculated by maximum likelihood estimates (MLE), the results of which are as follow:

- Normal distribution: mean = 130.20, standard deviation = 21.34
- Gamma distribution: Shape (K) = 4848.05, location = -1355.85, scale = 0.31
- Beta distribution: Alpha = 3475.29, Bata = 53981757, location = -1128.22, scale = 19548241

Gamma and Beta lines align perfectly on the normal line. Therefore, we can approximately conclude that the distribution of nucleus per image is normal with mean equals to 130.20 and standard deviation equals to 21.34 per 1,000x1,000 pixels image.



(a) histogram of nucleus per image

(b) histogram with distributions

Figure 36: shows histogram of nucleus per image (on the left) and the histogram with normal, Gamma and Beta distribution (on the right).

6.1.2 Sizes of each nuclei

According to table.6 and fig.37, the average nucleus per image is 14,843. The widths of the nucleus are spread between 1 to 213 pixels and the heights of the nucleus are spread between 1 to 253 pixels. As you can see, most nucleus are captured by a size of 64x64 pixels (around 98%). This number, 64x64 pixels, will be used for the nuclei library in the section 5.1.

Size	Miss count	Miss percent
64	303	2.0414 %
128	25	0.1684 %
150	15	0.1011 %
200	5	0.0337 %

average nucleus per image: 14,843

Max width: 213.0 Min width: 1.0

Max height: 253.0 Min height: 1.0

Table 6: statistics of nucleus sizes.

6.2 Building Fake Masks

We will now generate fake masks which will later be used as inputs for the Optional Pixel to Pixel model to synthesize nucleus images, which have nuclei locations cor-

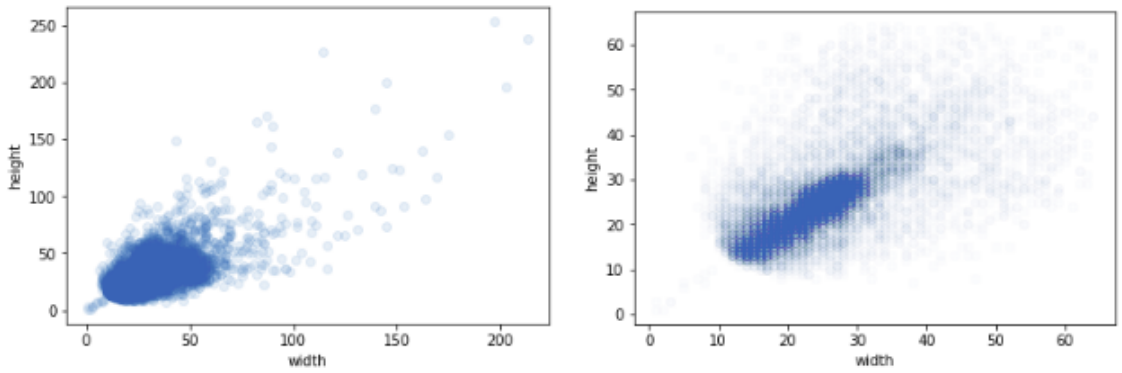


Figure 37: Left shows a scatter plot of weights and heights of each nuclei, Right shows a scatter plot of weights and heights of each nuclei but only those with width and height are smaller than 65.

respond to the masks. As discussed in the analysis on mask images 6.1, the average nucleus per 1,000x1,000 image is 130.20 and the standard deviation is 21.34. In this part, we want to generate 350x350 mask image so the average and standard deviation of the nucleus can be calculated as follow:

New average = 130.20 x 350 / 1,000 = 45.57

New Standard deviation = 21.34 x 350 / 1,000 = 7.47

The normal distribution with mean equals to 45.57 and standard deviation equals to 7.47 will be used to randomly draw the number of nucleus (N) in a synthesized mask. Then, we will use the trained Disentangled Variational Autoencoder (DVAE) model from the section 5 to generate N random nuclei shapes. The value of a latent variable, which is an input of DVAE, is sampled from normal distribution with 0 mean and 0.5 variance. Lastly, we randomly put the synthesized nucleus to a blank image, resulting in a synthesized mask.

6.3 Result and Performance

The results of the synthesized masks are in fig.38. As you can see, the results look quite similar to the real masks. We then synthesize nucleus images with these synthesized masks by inputing the masks to Pixel to Pixel model, fig. 39, and Optional Pixel to Pixel model, fig. 40. The results look really realistic compared to the original images. However, the results from Optional Pixel to Pixel have 2 types of images with different backgrounds. This shows that Optional Pixel to Pixel model

still works better than the plain Pixel to Pixel model when using the synthesized masks.

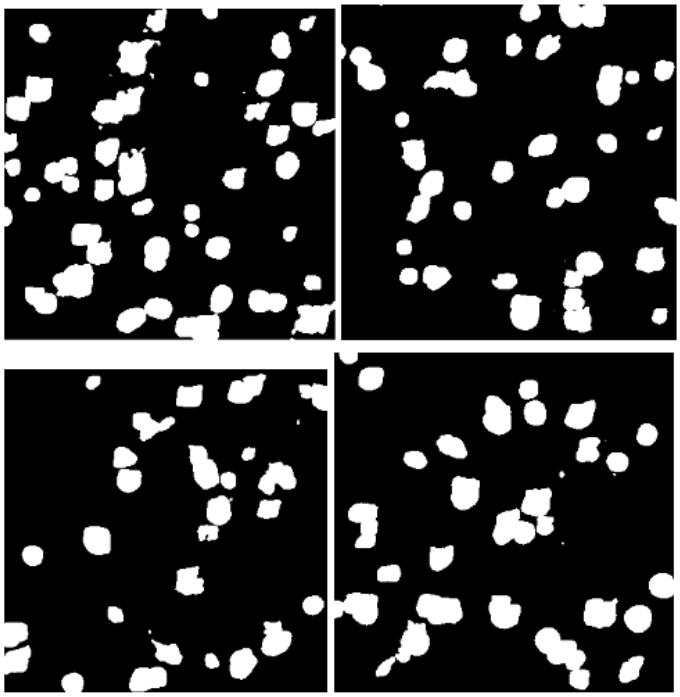


Figure 38: shows 4 samples of generated masks with latent vectors drawn from normal distribution of mean=0, std=0.5.

6.4 Conclusion

In this section, we do statistical analysis on all the masks from CRLM dataset. Then, we showed how to synthesize masks from Disentangled Variational Autoencoder model. We also showed and compared the results of synthesized images from synthesized masks of the plain Pixel to Pixel model and the Optional Pixel to Pixel model. This results confirm that Optional Pixel to Pixel works better in handing variations in the dataset.

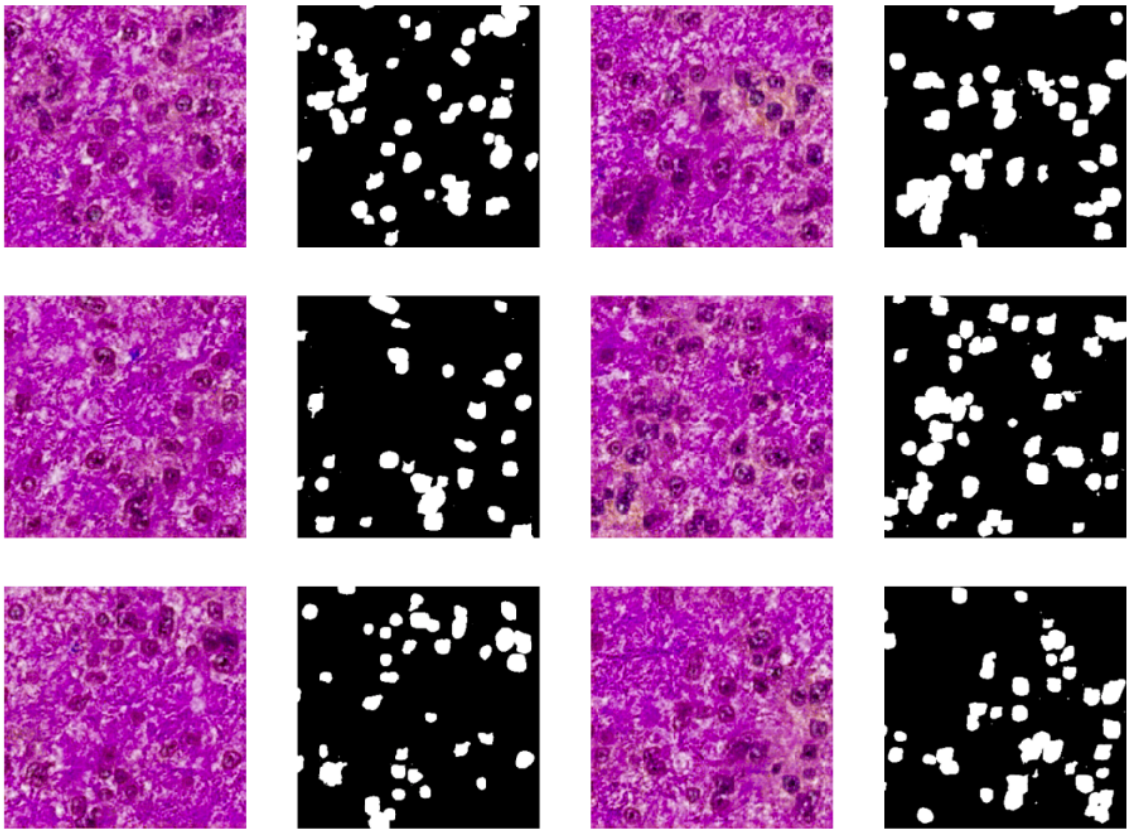


Figure 39: shows generated images on test data. Synthesized masks on 2nd and 4th columns provide synthesized nucleus images on 1st and 3rd columns, respectively.

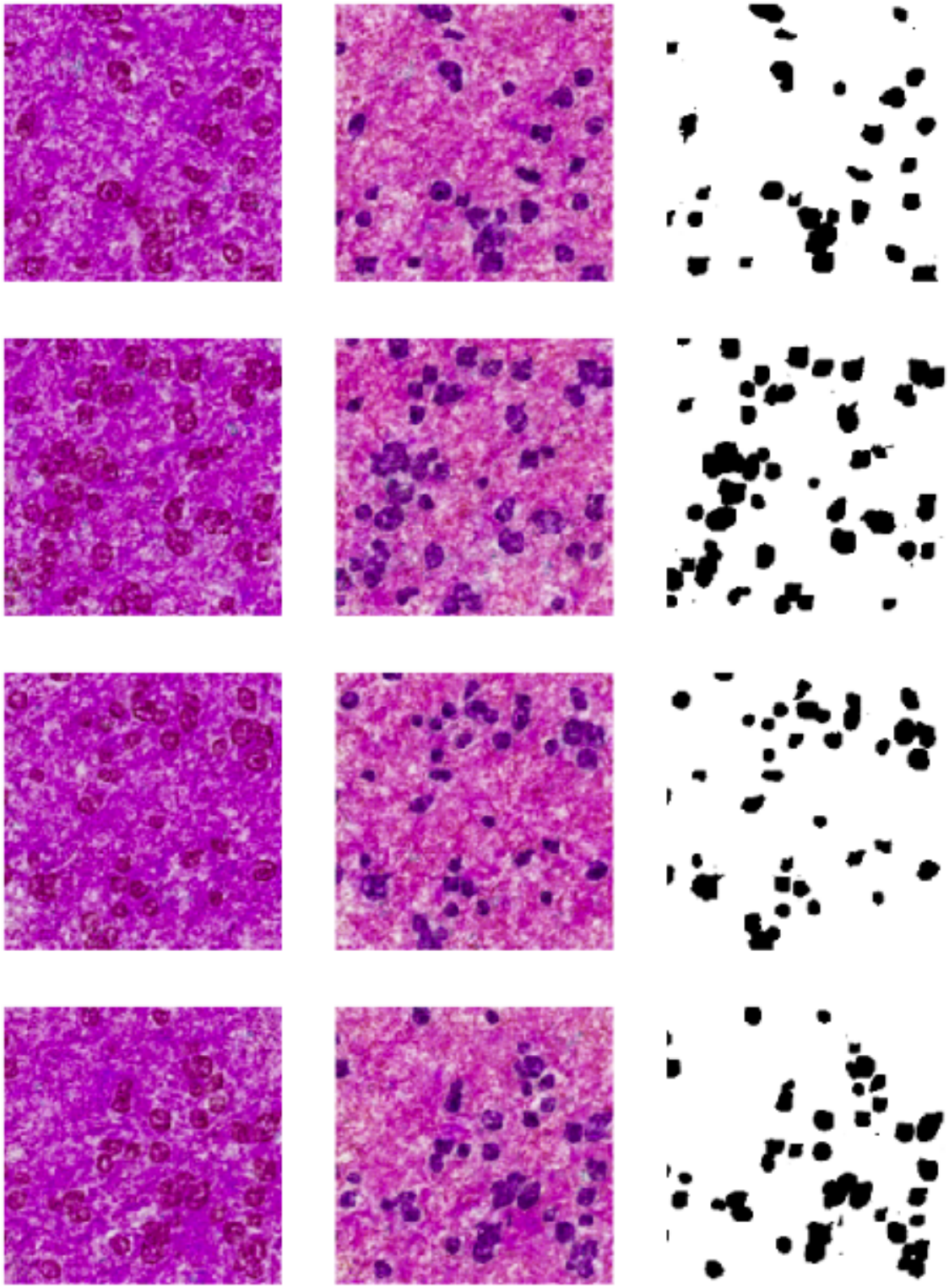


Figure 40: shows qualitative performance on test data, First two columns are generated images (option 1 and option 2), Third column is the generated mask from the previous part.

7 Conclusion and Future Works

The results from this work shows that deep learning algorithms, such as Disentangled Variational Autoencoder (DVAE) and the newly proposed Optional Pixel to Pixel, have high potential in biomedical image synthesis. DVAE works well in capturing nuclei shape distribution by encoding high dimensional nuclei shape images to 10 dimensional vectors and decoding the 10 dimensional vectors back to approximate nuclei shape images. After having a model for nuclei distribution, we can generate masks of nucleus images by randomly putting together synthesized nucleus on the same image and this masks will be used as inputs for the Optional Pixel to Pixel model. Furthermore, Optional Pixel to Pixel model provides multiple types of realistic synthesized nucleus images. These synthesized images will have the corresponding masks (the input of the model) which are quite accurate and these generated images can be used as a training dataset for further works such as nuclei segmentation.

Further improvement for this project might be first combining mask and nucleus image synthesis into a single model to improve the efficiency. Second, using GAN typed model to generate masks as in our approach does not account for correlations or distances between nucleus. Third, the Optional Pixel to Pixel model can be improved by being able to handle variations automatically.

We hope this project can solve the problem of not having enough training dataset for the future works and leads to advancement in biomedical imaging area.

References

[1] Miccai challenge 2018: Digital pathology: Segmentation of nuclei in images.

[2] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[3] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Le Hou, Ayush Agarwal, Dimitris Samaras, Tahsin M Kurc, Rajarsi R Gupta, and Joel H Saltz. Unsupervised histopathology image synthesis. *arXiv preprint arXiv:1712.05021*, 2017.

[7] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[8] Przemek Lewandowski. How to pick the right programming language (and why it should be python), Nov 2017.

[9] Yang Li, Quan Pan, Suhang Wang, Haiyun Peng, Tao Yang, and Erik Cambria. Disentangled variational auto-encoder for semi-supervised learning. *arXiv preprint arXiv:1709.05047*, 2017.

[10] Vebjorn Ljosa, Katherine L. Sokolnicki, and Anne E. Carpenter. Annotated high-throughput microscopy image sets for validation. *Nature Methods*, 9:637 EP –, Jun 2012. Correspondence.

[11] Yongyi Lu, Yu-Wing Tai, and Chi-Keung Tang. Conditional cyclegan for attribute guided face image generation. *arXiv preprint arXiv:1705.09966*, 2017.

[12] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[13] Andrew Ng. Machine learning.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Appendices

A Resources

Github repository for this project is

https://github.com/aiywatch/Histological-Image-Synthesis-Using-Deep-Learning

File list:

1. format_MICCAI_to_DSBowl.py

   Restructure MICCAI folders to Data Science Bowl format

2. datasets analysis.ipynb

   Analysis of all datasets

3. John dataset.ipynb

   Cleaning CRLM dataset, transform JSON to Masks, crop image

4. nuclei mask analysis.ipynb

   Analysis on mask, finding statistics of nucleus

5. nuclei library.ipynb

   Building Nuclei Library

6. Disentangled VAE.ipynb

   https://colab.research.google.com/drive/1CmaCl5PknXWVwJ3Gp9ZTVx-Stow3reM9

   Disentangled VAE model for nuclei distribution

7. Disentangled VAE - finding latent size.ipynb

   https://colab.research.google.com/drive/1P6Crtw4bxXHbe5nP0-EmTVKe5jEBh7JC

   Hyper-parameter tuning on latent vector size

8. mask_creation.ipynb

   https://colab.research.google.com/drive/1TA2mQop8d6z8r2lQlRID9fawMpnqBCaj

   Synthesizing masks, require trained DVAE models

9. optional pix2pix.ipynb

   https://colab.research.google.com/drive/1zYvpUM80K_x-Y2QltqQWP1sWGJ9qo8bE

   Optional Pixel to Pixel model for image synthesis

10. optional pix2pix (ResNet, unused).ipynb

    https://colab.research.google.com/drive/1YVnxoc0Q9TupVj0b3UHCvcgmaQ4kKzWq

    Optional Pixel to Pixel model for image synthesis - ResNet option but overfitting

11. pix2pix.ipynb

    https://colab.research.google.com/drive/1_IyugDochJx7g7yfMmJH4om7Ey-I6tZu

    Pixel to Pixel model for image synthesis

12. nuclei_shape_library.zip

    Nuclei shape dataset

13. web-demo

    Demo website for Optional Pixel to Pixel model. This also includes a trained model of Optional Pixel to Pixel.

B Demo Website

To make this project more practical, we have built a demo website. This website will synthesize a nucleus image from a mask using Optional Pixel to Pixel model which we trained. The home page of the website is shown in fig.41. You can upload a mask image to the website and click submit button. The nucleus image corresponding to the input mask will be shown, as in fig.42.

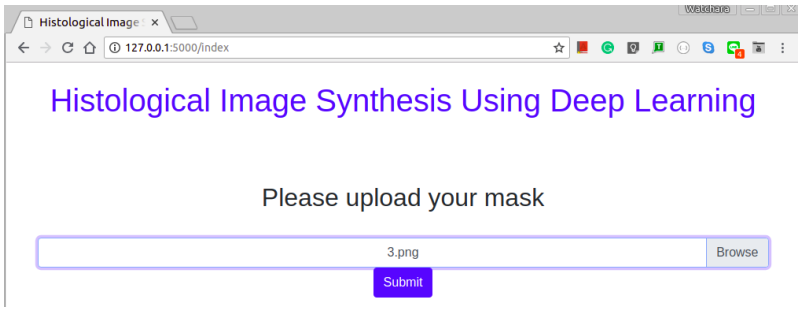


Figure 41: shows the home page of the demo website.

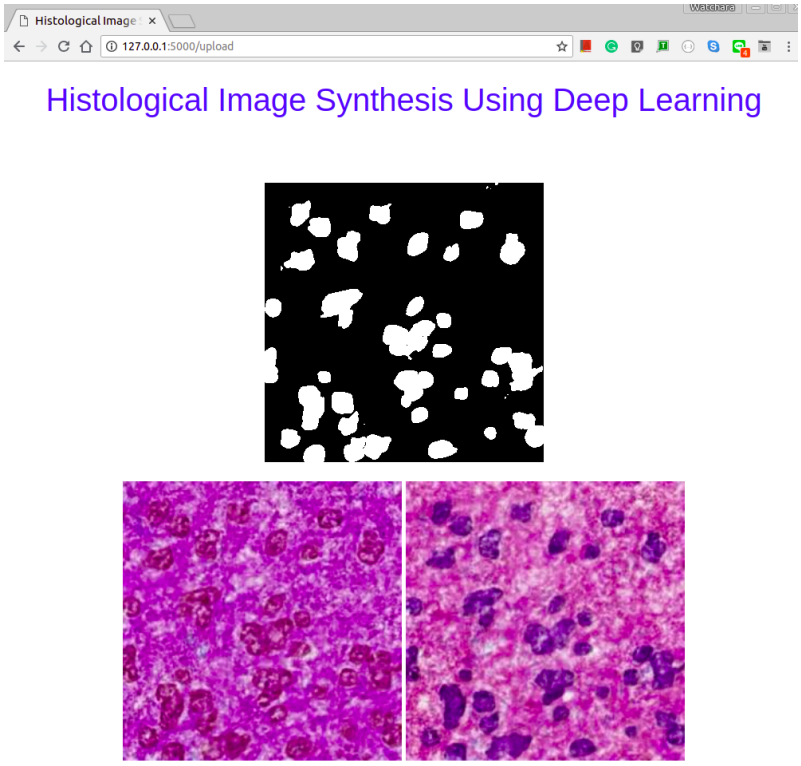


Figure 42: shows a result of synthesized nucleus images from an input mask.

C Disentangled Variational Autoencoder: Network Architecture Details

C.1 Encoder

1. Convolution: input channel=1, output channel=16, kernel size=3, stride=1, padding=1 -> Batch Normalization -> Relu
2. Convolution: input channel=16, output channel=32, kernel size=3, stride=1, padding=1 -> Batch Normalization -> Relu
3. Convolution: input channel=32, output channel=32, kernel size=3, stride=1, padding=1 -> Batch Normalization -> Relu
4. Convolution: input channel=32, output channel=16, kernel size=3, stride=1, padding=1 -> Batch Normalization -> Relu
5. 2 Fully connected: input size=1096, output size=10 -> Relu
6. Normal distribution from mean and variance vector -> 10 dimension vector

C.2 Decoder

1. Fully connected: input size=10, output size=100 -> Relu
2. Fully connected: input size=100, output size=1096 -> Relu
3. Deconvolution: input channel=16, output channel=32, kernel size=3, stride=2, padding=1 -> Batch Normalization -> Relu
4. Deconvolution: input channel=32, output channel=32, kernel size=3, stride=1, padding=1 -> Batch Normalization -> Relu
5. Deconvolution: input channel=32, output channel=16, kernel size=3, stride=2, padding=1 -> Batch Normalization -> Relu
6. Deconvolution: input channel=16, output channel=1, kernel size=3, stride=1, padding=1 -> Batch Normalization -> Relu

D Optional Pixel to Pixel: Network Architecture Details

D.1 Generator: U-Net

Input: Mask (256x256x1) + Option layer (256x256x1) = 256x256x2

Encoder Layer 1: Convolutional layer(input channel=2, output channel=64, kernel size=4, stride=2, padding=1), output=128x128x64 -> Leaky Relu(slope=0.2)

Encoder Layer 2: Convolutional layer(input channel=64, output channel=128, kernel size=4, stride=2, padding=1), output=64x64x128 -> Leaky Relu(slope=0.2) -> Batch Normalization

Encoder Layer 3: Convolutional layer(input channel=128, output channel=256, kernel size=4, stride=2, padding=1), output=32x32x256 -> Leaky Relu(slope=0.2) -> Batch Normalization

Encoder Layer 4: Convolutional layer(input channel=256, output channel=512, kernel size=4, stride=2, padding=1), output=16x16x512 -> Leaky Relu(slope=0.2) -> Batch Normalization

Encoder Layer 5: Convolutional layer(input channel=512, output channel=512, kernel size=4, stride=2, padding=1), output=8x8x512 -> Leaky Relu(slope=0.2) -> Batch Normalization

Encoder Layer 6: Convolutional layer(input channel=512, output channel=512, kernel size=4, stride=2, padding=1), output=4x4x512 -> Leaky Relu(slope=0.2) -> Batch Normalization

Encoder Layer 7: Convolutional layer(input channel=512, output channel=512, kernel size=4, stride=2, padding=1), output=2x2x512 -> Leaky Relu(slope=0.2) -> Batch Normalization

Encoder Layer 8: Convolutional layer(input channel=512, output channel=512, kernel size=4, stride=2, padding=1), output=1x1x512 -> Relu

Decoder Layer 1: Deconvolutional layer(input channel=512, output channel=512, kernel size=4, stride=2, padding=1), output=2x2x512 -> Batch Normalization -> Drop out(0.5)

Decoder Layer 2: Input=Relu(Decoder 1 +Encoder 7) -> Deconvolutional layer(input channel=1024, output channel=512, kernel size=4, stride=2, padding=1), output=4x4x512 -> Batch Normalization -> Drop out(0.5)

Decoder Layer 3: Input=Relu(Decoder 2 +Encoder 6) -> Deconvolutional layer(input channel=1024, output channel=512, kernel size=4, stride=2, padding=1), output=8x8x512 -> Batch Normalization -> Drop out(0.5)

Decoder Layer 4: Input=Relu(Decoder 3 +Encoder 5) -> Deconvolutional layer(input channel=1024, output channel=512, kernel size=4, stride=2, padding=1), output=16x16x512 -> Batch Normalization

Decoder Layer 5: Input=Relu(Decoder 4 +Encoder 4) -> Deconvolutional layer(input channel=1024, output channel=256, kernel size=4, stride=2, padding=1), output=32x32x256 -> Batch Normalization

Decoder Layer 6: Input=Relu(Decoder 5 +Encoder 3) -> Deconvolutional layer(input channel=512, output channel=128, kernel size=4, stride=2, padding=1), output=64x64x128 -> Batch Normalization

Decoder Layer 7: Input=Relu(Decoder 6 +Encoder 2) -> Deconvolutional layer(input channel=256, output channel=64, kernel size=4, stride=2, padding=1), output=128x128x64 -> Batch Normalization

Decoder Layer 8: Input=Relu(Decoder 7 +Encoder 1) -> Deconvolutional layer(input channel=128, output channel=3, kernel size=4, stride=2, padding=1), output=256x256x3 -> Tanh

Output: Nucleus Image (256x256x3)

D.2 Discriminator

Input: Nucleus image (256x256x3) + Mask (256x256x1) + Option layer (256x256x1) = 256x256x5

Layer 1: Convolutional layer(input channel=5, output channel=64, kernel size=4, stride=2, padding=1), output=128x128x64 -> Leaky Relu(slope=0.2)

Layer 2: Convolutional layer(input channel=64, output channel=128, kernel size=4, stride=2, padding=1), output=64x64x128 -> Batch Normalization -> Leaky Relu(slope=0.2)

Layer 3: Convolutional layer(input channel=128, output channel=256, kernel size=4, stride=2, padding=1), output=32x32x256 -> Batch Normalization -> Leaky Relu(slope=0.2)

Layer 4: Convolutional layer(input channel=256, output channel=512, kernel size=4, stride=1, padding=1), output=31x31x512 -> Batch Normalization -> Leaky Relu(slope=0.2)

Layer 5: Convolutional layer(input channel=512, output channel=1, kernel size=4, stride=1, padding=1), output=30x30x1 -> Sigmoid

Output: 30x30x1 -> Classify whether the input is real or fake