DATS6203 Machine Learning II
Final Project Report

Predicting Invasive Ductal Carcinoma in Tissue Slices

Yue Li

Advisor: Amir Jafari

Fall 2022

Dec 13, 2022

Introduction

Invasive ductal carcinoma (IDC) is one of the most common types of breast cancer. It's malicious and able to form metastases which makes it especially dangerous. Often a biopsy is done to remove small tissue samples. Then a pathologist must decide whether a patient has IDC, another type of breast cancer or is healthy. In addition, sick cells need to be located to find out how advanced the disease is, and which grade should be assigned. This has to be done manually and is a time-consuming process. Furthermore, the decision depends on the expertise of the pathologist and his or her equipment. Therefore, deep learning could be of great help to automatically detect and locate tumor tissue cells and to speed up the process. In order to exploit the full potential, one could build a pipeline using massive amounts of tissue image data of various hospitals that were evaluated by different experts. This way one would be able to overcome the dependence on the pathologist which would be especially useful in regions where no experts are available.

In the project, we used two pre-trained convolutional neural networks to detect IDC in the tissue slice images. In order to avoid the overfitting issue and improve the model performance, we implemented two methods. One is data augmentation method and the other is learning rate search method. In the data augmentation, methods in the transformer were used to increase the diversity of the images and a general adversarial network was implemented to standardize the image. In the training process, the cyclical learning rate (CLR) search method was used when training the Resnet18 and the VGG16 framework. The best performance is the accuracy 0.85 on the test set on Resnet18 with CLR search.

Individual Work

In this project, my work is data processing, model selection, model training, cGAN model training and fake images generation.

Transfer Learning

Except for using a self-training CNN based framework and training from scratch, we selected a pre-trained CNN model. Pretrian model refers to a model or a saved network created by someone else and trained on a large dataset, using the resources that are not available to everyone. Take ImageNet for example. It contains over 14 million images with 1.2 million of them assigned to one of a 1000 categories. Hence it would be really beneficial for us to use these models. The weights and bias were extracted from the trained model and could be used in the future. It saves huge efforts required to reinvent the wheel. The idea of using the trained weights and bias on a different but related problem is transfer learning. With the concept of transfer learning, the ‘knowledge’ of pre-trained models could be transferred.

In computer vision, for example, neural networks usually try to detect edges in the earlier layers, shapes in the middle layer and some task-specific features in the later layers. In transfer learning, the early and middle layers are used and we only retrain the latter layers. It helps leverage the labeled data of the task it was initially trained on. According to DeepMind CEO Demis Hassabis,

transfer learning is also one of the most promising techniques that could lead to artificial general intelligence (AGI) someday.

Pre-trained Model - VGG

ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) is an annual event to showcase and challenge computer vision models. In the 2014 ImageNet challenge, Karen Simonyan & Andrew Zisserman from Visual Geometry Group, Department of Engineering Science, University of Oxford showcased their model in the paper titled "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION," which won the 1st and 2nd place in object detection and classification. The 16 in VGG16 refers to 16 layers that have weights. In Figure 3 and Figure 4 , the architecture of VGG-16 is shown. The input of VGG16 is 224*224 with three channels. The structure of VGG16 is two or three convolutional layers and one pooling layer. This structure is called a block. In VGG16, it contains 5 blocks, 13 Conv layers, 5 pooling layers and 3 dense layers.

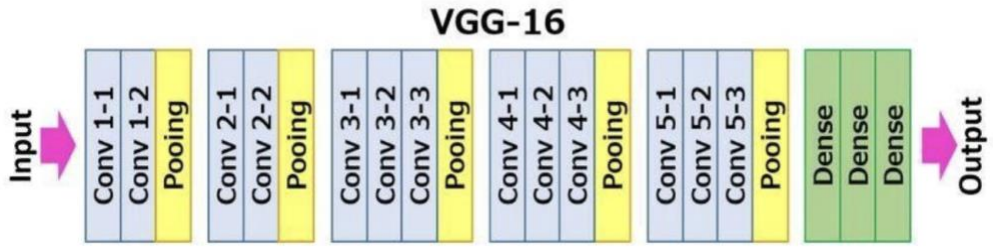


Figure 3. The architecture of VGG16

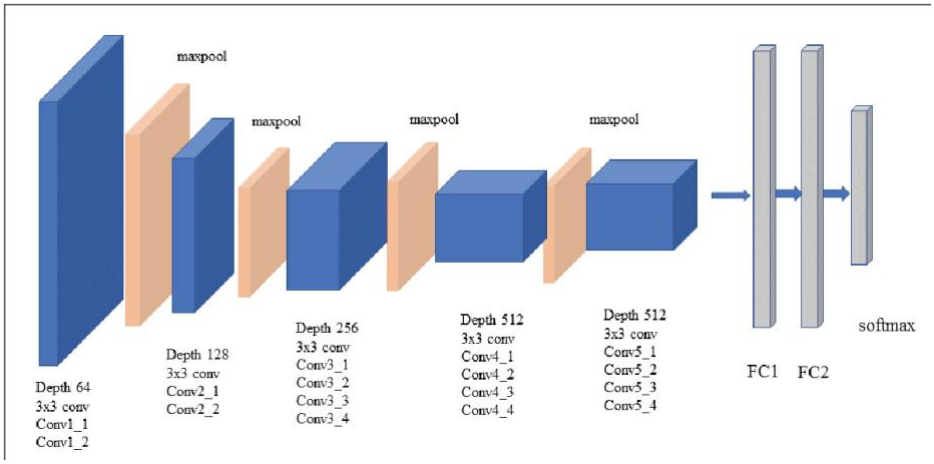


Figure 4. The architecture of VGG16

In Figure 5, the basic structure of VGG16 was built in python. It contained 134,268,738 parameters and the parameter size was 537MB. In this project, we used a pre-trained VGG16 instead of training this architecture from scratch. Rebuilding this basic structure helped us have a better understanding of the architecture of the VGG16.

```
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
VGG16                                    [10, 2]                   --
├─Conv2d: 1-1                            [10, 64, 224, 224]        1,792
├─Conv2d: 1-2                            [10, 64, 224, 224]        36,928
├─MaxPool2d: 1-3                         [10, 64, 112, 112]        --
├─Conv2d: 1-4                            [10, 128, 112, 112]       73,856
├─Conv2d: 1-5                            [10, 128, 112, 112]       147,584
├─MaxPool2d: 1-6                         [10, 128, 56, 56]         --
├─Conv2d: 1-7                            [10, 256, 56, 56]         295,168
├─Conv2d: 1-8                            [10, 256, 56, 56]         590,080
├─Conv2d: 1-9                            [10, 256, 56, 56]         590,080
├─MaxPool2d: 1-10                        [10, 256, 28, 28]         --
├─Conv2d: 1-11                           [10, 512, 28, 28]         1,180,160
├─Conv2d: 1-12                           [10, 512, 28, 28]         2,359,808
├─Conv2d: 1-13                           [10, 512, 28, 28]         2,359,808
├─MaxPool2d: 1-14                        [10, 512, 14, 14]         --
├─Conv2d: 1-15                           [10, 512, 14, 14]         2,359,808
├─Conv2d: 1-16                           [10, 512, 14, 14]         2,359,808
├─Conv2d: 1-17                           [10, 512, 14, 14]         2,359,808
├─MaxPool2d: 1-18                        [10, 512, 7, 7]           --
├─Linear: 1-19                           [10, 4096]                102,764,544
├─Linear: 1-20                           [10, 4096]                16,781,312
├─Linear: 1-21                           [10, 2]                   8,194
==========================================================================================
Total params: 134,268,738
Trainable params: 134,268,738
Non-trainable params: 0
Total mult-adds (G): 154.80
==========================================================================================
Input size (MB): 6.02
Forward/backward pass size (MB): 1084.46
Params size (MB): 537.07
```

Figure 5. The summary of architecture for VGG16 in python

Pre-trained Model - ResNet

In the ILSVRC after the first CNN based architecture (AlexNet) that won the ImageNet 2012 competition, every subsequent winning architecture uses more layers in a deep neural network to reduce the error rate. Experts attempted to build a more and more deep neural network expecting better performance. This works for fewer layers, but when we increase the number of layers, there is a common problem in deep learning associated with that called the Vanishing/Exploding gradient. This causes the gradient to become 0 or too large. Thus when we increase the number of layers, the training and test error rate also increases. In Figure 6, we can observe that a 56-layer CNN gives more error rate on both training and testing set than a 20-layter CNN. After analyzing more on error rate the authors were able to reach the conclusion that it is caused by vanishing or exploding gradient.

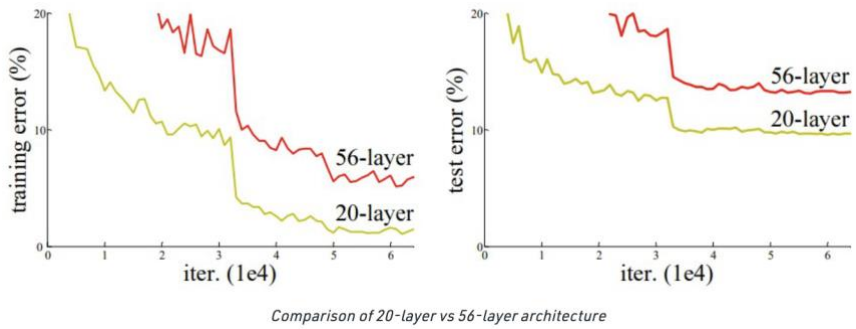


Figure 6. Loss on 20-layer network and 56-layer network

ResNet, which was proposed in 2015 by researchers at Microsoft Research, introduced a new architecture called Residual Network. In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Blocks. In this network, we use a technique called skip connections. The skip connection connects activations of a layer to further layers by skipping some layers in between. This forms a residual block, shown in Figure 7. Resnets are made by stacking these residual blocks together. For a deep neural network, it is hard to fit the relationship between input and output. In a residual unit, the input is x, output is $H(x), H(x) = F(x) + x$. $F(x)$ is the residual. The advantage of adding this type of skip connection is that if any layer hurts the performance of architecture then it will be skipped by regularization. Therefore, in the residual unit, network fit the $F(x) = 0$ mapping. Figure X shows the different structure of ResNet. They all have four residual units. For ResNet18 and ResNet34, the residual unit contains two 3*3 convolutional layers, which is different from the bottleneck structure for ResNet50, ResNet101, ResNet152.

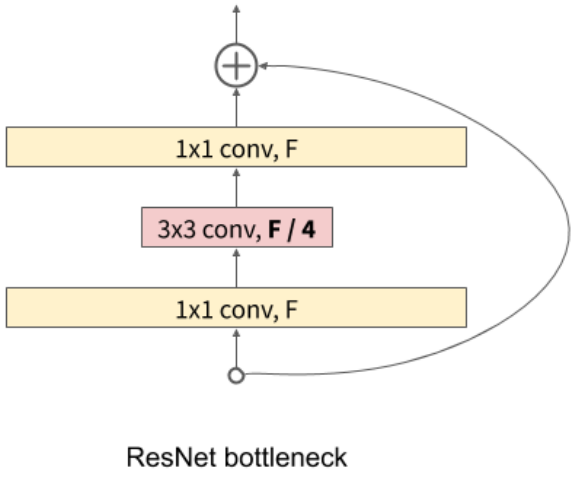


Figure 7. The structure of ResNet bottleneck

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 8. ResNet architectures

Cyclical Learning Rates (CLR)

Cyclical Learning Rates is a method to search the best learning range for a training process. The learning rate is a hyper-parameter that determines the pace at which an algorithm updates or learns the values of a parameter estimate.

Instead of monotonically decreasing the learning rate, this method lets the learning rate cyclically vary between reasonable boundary values. Training with cyclical learning rates instead of fixed values achieves improved classification accuracy without a need to tune and often in fewer iterations.

The Cyclical Learning rates method was implemented to train a neural network with a LR that changes in a cyclical way for each batch, instead of a non-cyclic LR that is either constant or changes on every epoch. The learning rate schedule varies between two bounds.

It is one of the most important hyper-parameters for training neural networks and is the key to effective and faster training of the network. Learning rate decides how much of the loss gradient is to be applied to our current weights to move them in the direction of lower loss.

In CLR, we vary the LR between a lower and higher threshold. The logic is that periodic higher learning rates within each epoch helps to come out of any saddle points or local minima if it encounters into one. If the saddle point happens to be an elaborated plateau, lower learning rates will probably never generate enough gradient to come out of it, resulting in difficulty in minimizing the loss.

General Adversarial Network (GAN)

The fundamental structure of GAN is a discriminator and a generator. Generator generates fake images. The input of the generator is noise source, the output is a fake image. In discriminator, it

is a convolution based network. The aim of the discriminator is to identify the real image and fake image. These two networks are adversarial. The loss of GAN is shown below in equation 1. m is the number of samples. x_i is the real image. z_i is the fake image. $D(G(z_i))$ and $D(x_i)$ is the result of the fake image and real image, which the probability of label equals to 1. For a discriminator, the aim is to identify every fake image and make $D(G(z_i))$ equals to 0 and $D(x_i)$ equals to 1. Therefore the loss function is maximum. For a generator, the aim is to make $D(G(z_i))$ equals to 1. The loss function is minimum. The aim of training GAN is to minimize the loss on generator and maximum the loss on discriminator, like equation 2. At the beginning training process, $D(G(z_i))$ equals to 0 and $D(x_i)$ equal to 1. The expecting result of the GAN is $D(G(z_i))$ and $D(x_i)$ equal to 0.5.

$$V(D,G) = \frac{1}{m}\sum_{i=1}^{m}[logD(x_i) + log(1 - D(G(z_i)))] \tag{1}$$

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}\ (x)}\left[logD(x)\right] + \mathbb{E}_{z\sim p_z\ (z)}\left[log(1 - D(G(z)))\right] \tag{2}$$

After introducing the loss function of GAN, we will explain the architecture of GAN. In Figure X, it shows the structure of the discriminator and generator DGAN. First, discriminator is a CNN based network. Generator uses transposed convolutional layers A transposed convolutional layer is usually carried out for upsampling i.e. to generate an output feature map that has a spatial dimension greater than that of the input feature map, just as shown in the Figure 9, generator is able to upsample the feature image. In this project, we implemented a cGAN to do data augmentation.

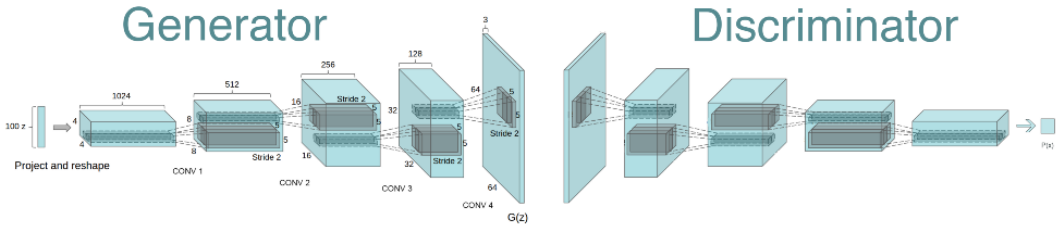


Figure 9. GAN architectures

Data augmentation

Image diversity increase

In this project, we did two parts of methods for data augmentation on the train set to avoid overfitting and improve the performance. Our dataset is H&E stained Tissue slice dataset. First

method is increasing image diversity. Second is the standardizing of the color of images, called color normalization. In this project, we implemented 'imguag' and 'skimage'. Figure X, is the original image. We used several methods to transform images. The examples are shown below.

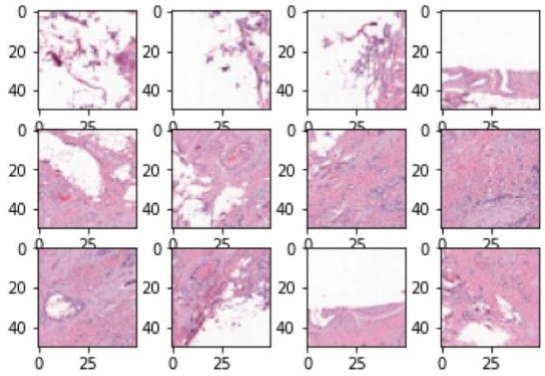


Figure 11. Images applied *'iaa.arthmetic.Affine'*

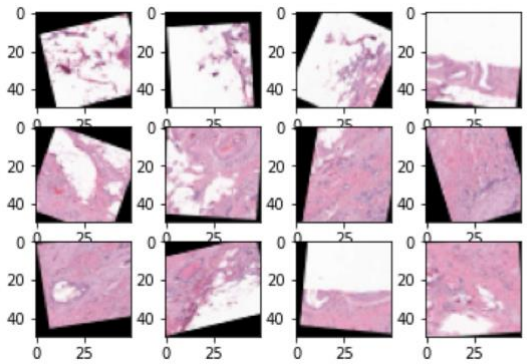


Figure 12. Images applied '*iaa.arthmetic.AdditiveGaussianNoise'*

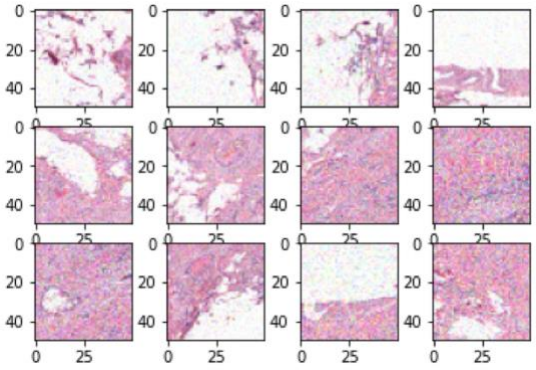


Figure 13. Images applied 'iaa.color.MultiplyAndAddToBrightness'

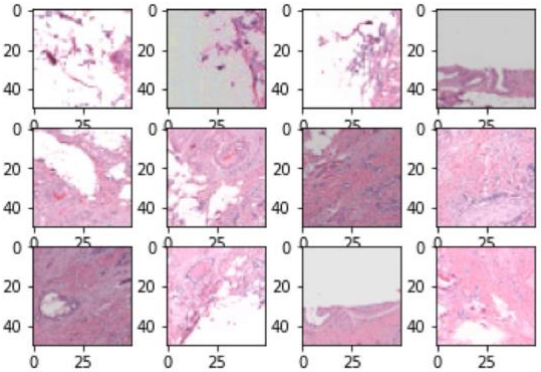


Figure 14. Images applied '*iaa.color.MultiplyAndAddToBrightness'*

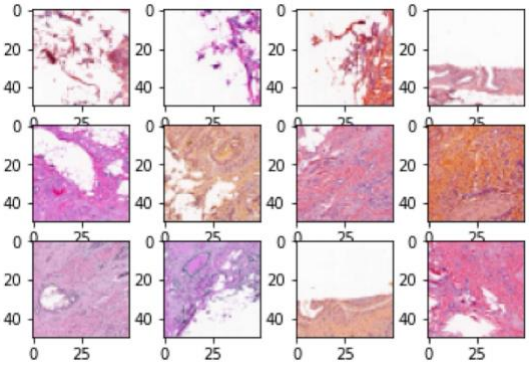


Figure 15. Images applied '*iaa.color.MultiplyHueAndSaturation*'

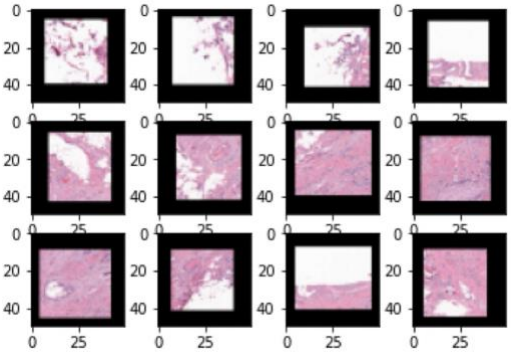


Figure 16. Images applied ' *iaa.size,CropAndPad'*

Image Color normalization

In the color normalization, we usePix2Pix (cGAN) for stain transfer. Pix2Pix is a conditional GAN (cGAN) set up as a pairwise image translation algorithm. The pair consists of a target image and an input condition/label image which is passed to the generator. It should be noted that unlike the cGAN the input is an image, not a noise+label vector. The components of Pix2Pix architecture are a follows:

Generator: A U-Net, which has skip connections, is used so that low level information can be passed from input image to output image. Noise in the form of dropout is applied to several layers of the generator instead of an input noise used in cGAN models.

Discriminator: The discriminator is a PatchGAN network. It classifies smaller patches of the input image (either real or fake) instead of discriminating the entire image at once.

Here the RGB color and corresponding grayscale transform of an image tile serve as target/condition image pairs. Grayscale images are normalized across channels and serve as a neutral template for stain-style transfer. As training progresses, the stain style generalizes capturing the statistics over the entire training set which had been acquired from different labs. It should be noted that while the Pix2Pix algorithm requires paired data for training, it is easy to satisfy this by applying grayscale transforms.

Training: Color/grayscale image pairs are derived from the H&E stained PCam dataset. The grayscale images serve as input to the generator. The generator outputs stained color images. The discriminator has two input pairs: the generated image/grayscale image pair serves as the fake input, the color/grayscale image pair is the real input. See the figure below.

The network is trained using most of the same parameters as the original Pix2Pix paper. The structure of Pix2Pix is shown in Figure16.

Evaluation: We plot the target image and generated image during the training cycle. In real world implementations, the target image and generated image are compared using metrics such as PSNR, SSIM as well as human visual perception. The generated image can also be validated on a clinical use-case such as classification.(reference)

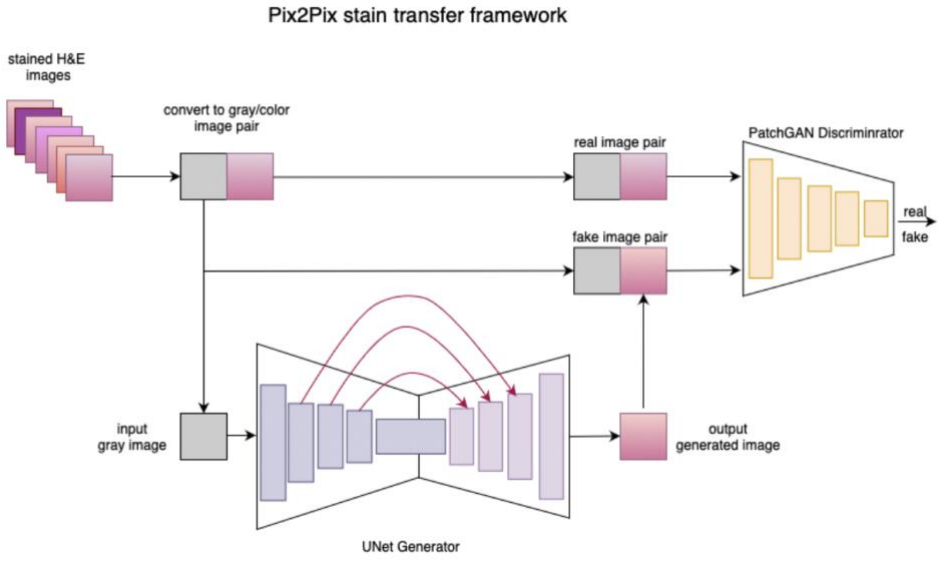
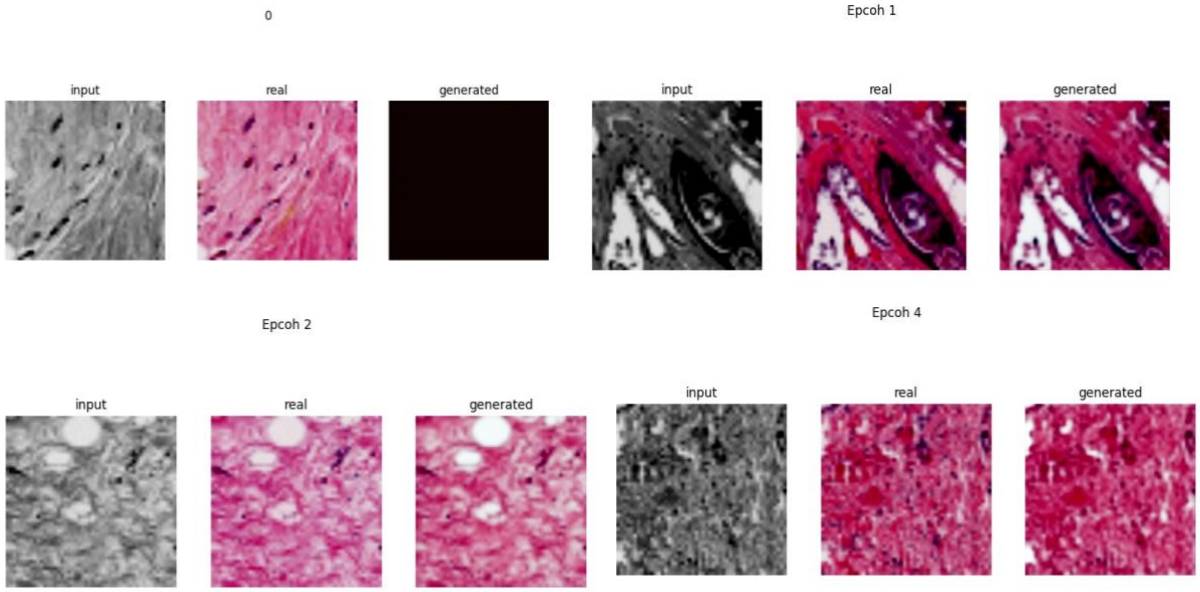


Figure 16. Pix2Pix network Structure

We implemented Pix2Pix to our Tissue slice dataset. We selected 20,000 images from the train set to train the network. Since the training process was really time consuming, the epoch was set to 6. In Figure 17, the results were shown. The input of the generator is a grayscale image and output of the generator is a colored image. After three epochs, the generated image was more detailed and intense in color. Therefore, this Pix2Pix method can help to normalize the color of the stained images in the train set.



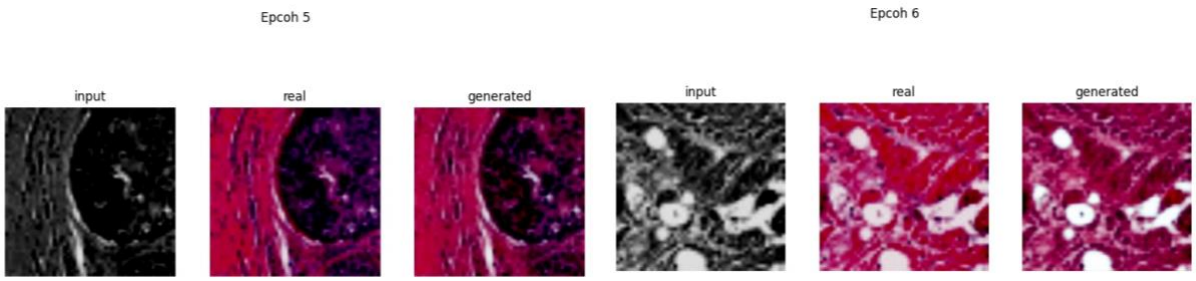


Figure 17. Generated fake images from Pix2Pix network from different Epochs

Metric selection

Our model is to predict the IDC for each patch. The target distribution is imbalance. We select the F1-score as the evaluation metric. F1-score is a harmonic mean of precision and recall. In this project, recall refers to how many selected cases are positive and precision refers to how many positive cases are selected.

$$f_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} \tag{3}$$

$$recall = \frac{TruePositives}{TruePositives + FalsePositives} \tag{4}$$

$$precision = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{5}$$

Model selection

In this project, we selected VGG16 and ResNet18 models. The reason is VGG16 is the first model to have the repeated blocks in the framework. It was a milestone at that time. The second model is ResNet. The reason for selecting ResNet is it is the first framework to have a lot of layers by using residual units. It solved the issue that a deep network with too many layers didn't have a better result than the shallow one.

Hence what we did was, we took a pretrained Resnet18 and VGG16 and changed the last layer of them, by adding some dense layers of our own to it (randomly initialized) and trained this new model. The input of models were images and their targets. The output will be the classification prediction.

Learning rate search

Deep learning models are typically trained by a stochastic gradient descent optimizer. There are many variations of stochastic gradient descent: Adam, RMSProp, Adagrad, etc. All of them let you set the learning rate. This parameter tells the optimizer how far to move the weights in the direction opposite of the gradient for a mini-batch.

Thus, by using the skill mentioned above, cyclical learning rates method, our learning rate changed cyclically while we trained our model. The cycle was set to the number of the epoch. Specifically, the number of cycles is the image number divided by the batch size. In this project, the number of cycles was around 6000 steps. In this method, The maximum learning rate could be selected by running one epoch. At first, the maximum learning rate was set to 0.01, and the minimum learning rate was set to 1e-6. In Figure 18, when the learning rate was 0.06, the loss was increasing. Therefore, the maximum learning rate was changed to 0.06.

Figure 18. Search for the optimal learning rate

Train processing

In this project, two models were trained with CLR method and without CLR method. The parameters of training were in table 2. A batch size of 32 is a good default value, also he stated that the larger batch size will quicken the computation of the network but will decrease the updates required for the network to reach convergence. The author stated that the batch size likely impacts the convergence time and not network performance. The authors noted the small batch sizes are more robust than the large batch sizes. Each epoch was trained for around 45 minutes. In this project, the number epoch was set to 30.

Table 1

Model	**CLR**
ResNet18	True
ResNet18	False
VGG16	True

VGG16	False

Table 2

Parameter	**Value**
Epoch	30
Batch size	32
Learning Rate	`max_lr = 0.06, min_lr = 1e-6`
Optimizer	SGD
Criterion	CrossEntropyLoss(weighted)
Evaluation Metrics	F1-Score

Just as the regular process, first we set up the train, valid, and test loss, then use model.train() function to calculate the loss. After getting the loss, the differences between prediction and the real label, we go backward using the gradient to set up a new set of weight which the changing ratio is appendant to the learning rate we set. While we got our loss of valid and test data, if the loss accuracy performed better than last time we saved the model and continued to repeat this process again and again until it reached the epoch number we set.

The task to predict the presence of invasive ductal carcinoma given a tissue patch is a binary classification. A common loss for this problem is the binary cross entropy function. Using it we could only use one single output neuron to make predictions. As we like to compute the f1-score during training and prediction and we need to compute false and true positives/negatives in a simple manner we will use the cross entropy loss with K=2 output neurons or 2 classes .

$$L = -\sum_{n=1}^{N}\sum_{k=1}^{K=2} w_k \cdot t_{n,k} \cdot \ln(y_{nk})$$

Figure 19. Loss function

For the criterion, we have applied the weighted process into our training process**.** The major reason for this step is because our dataset is unbalanced and after applying we set the two classes with different weights during the training process.

In our case we have much more negative (healthy) patches than those with cancer. To deal with this imbalance we like to increase the impact of the gradients of positive cases during training and we can do so with a higher weight than for the negative cases.

Results & Summary

In table 3, the results of the training process were shown. ResNet18 had better accuracy and less training time than VGG16. CLR method decreased the training time for both ResNet18 and VGG16. In Figure 20 and Figure 21, the training processing was more stable and converged faster with CLR.

Table 3

Model	CLR	Validation Accuracy	Training Time
ResNet18	Yes	0.85%	265m51s
ResNet18	No	0.77%	270m14s
VGG16	Yes	0.79%	390m44s
VGG16	No	0.73%	393m43s

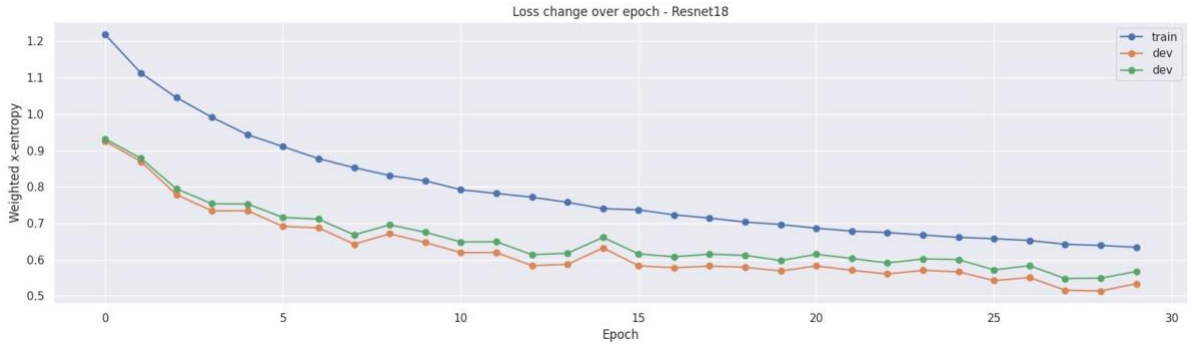


Figure 20. Training process with CLR on ResNet18

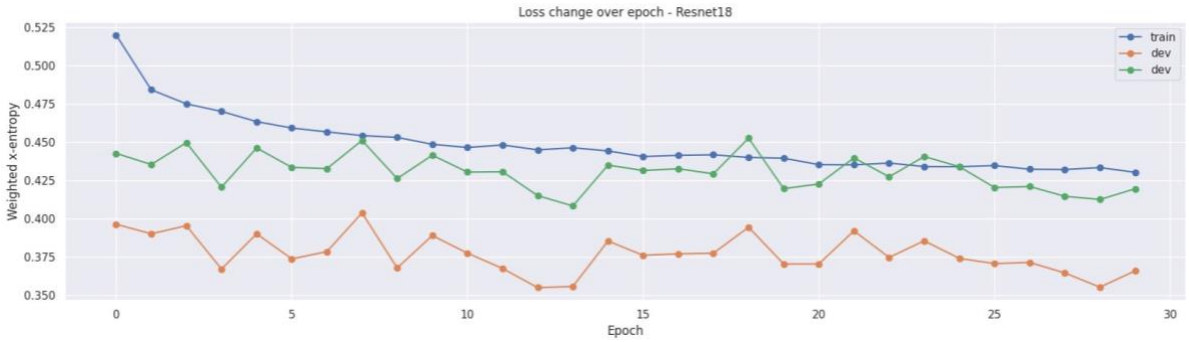


Figure 21. Training process without CLR on ResNet18

In this project, ResNet18 has 25.5 million parameters and VGG16 has around 138 million parameters. However, the number of parameters is not the factor that influences the training speed. The architecture is. ResNet18 has a large kernel size 7 * 7 at the beginning, it helped to reduce the image size to half. Compared with VGG16, the kernel size is 3*3 at the first layer. Therefore, it would be more computationally expensive than ResNet18. ResNet18 follows the rule that is much thinner and much deeper in the architecture of the network. Cyclic learning rate method helped to improve the accuracy on the validation set and converged faster both on VGG16 and ResNet18. With the Cyclic learning rate method, the training process was more stable.

Code:

- VGG.py: Implement Basic VGG16 architecture
- cGan.py: Generate fake images for data augmentation
- read data.py: Rules to read in the original data
- train_code.py: Training process