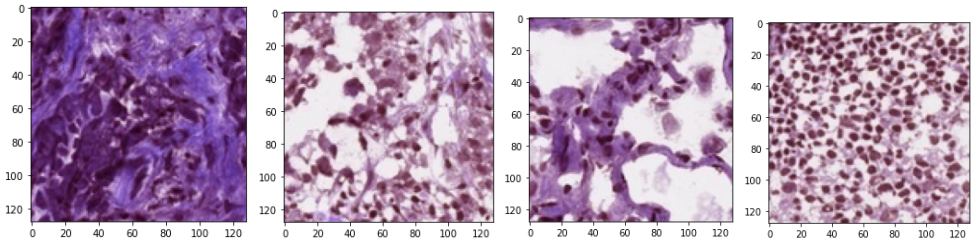
Li Yonghao (Assignment 4)

1. Load the data and visualization

Amount of images 6929: Artifacts: 2380, cancer: 2878, normal: 1187, other:484. I used to used keras, so I use the os.listdir( ) and img_to_array( ) function to load the data, than use the np.transpose( ) to reshape the data fit for pytorch.



imshow( ) to visualize the data

I add labels to each category, artifacts: 0, cancer: 1, normal: 2, other:3.

Then I extract 100 images of each category as test set, rest will be training set and validation set with the rate of 0.8.

I normalize the data in to [0,1] to get a better result of training.

Below is my CNN model with 3 conv2d layers and 2 linear layers:

```
class cnn(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.cnn_layers = torch.nn.Sequential(

            # Defining a 2D convolution layer
            torch.nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            torch.nn.BatchNorm2d(64),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),

            # Defining another 2D convolution layer
            torch.nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1),
            torch.nn.BatchNorm2d(128),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),

            torch.nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1),
            torch.nn.BatchNorm2d(256),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.linear_layers = torch.nn.Sequential(
            torch.nn.Flatten(1),
            torch.nn.Linear(256 * 2 *2, 512),
            torch.nn.ReLU(inplace=True),
            torch.nn.Dropout(p=0.2),
            torch.nn.Linear(512, 4)
        )

    # Defining the forward pass
    def forward(self, x):
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x
```

For training, I try different parameters:

Learning rate: 0.1, 0.01, 0.001 (best one with lowest loss score under 0.1).

Rate decay: I use Adam and try Adagrad, Adadelta, RMSprop and Time-Based Decay: lr *= (1. / (1. + self.decay * self.iterations)) in SGD optimizer.

Drop out: I try 0.1, 0.2 (best one), 0.3, 0.4, 0.5.

Weight decay: 0.01

Early stop: I do then by using loss of validation and training, if the validation loss doesn't decrease but increase, I will save the best model to avoid that.

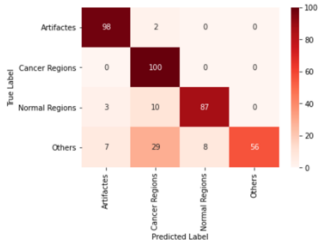
Network architecture adjustment: I try some and use the best one of my tries.

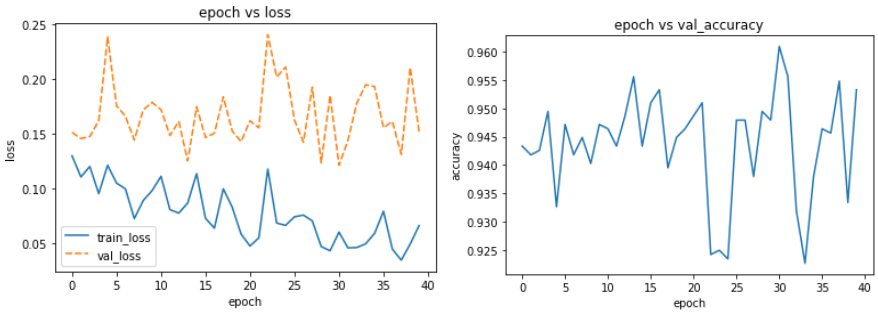
Detail of training:

```
import torch.optim as optim
loss_func = torch.nn.CrossEntropyLoss()
lr = 0.001
weight_decay=0.01
optimizer = optim.Adam(net.parameters(), lr=lr, betas=(0.9, 0.999), eps=1e-08, weight_decay=weight_decay)
optimizer_name = 'Adam'
```

```
epoch5 train loss: 0.10503230807257861, val loss: 0.1757908747954802 val accuracy: 0.947166921898928, patience: 2
epoch6 train loss: 0.09976871511558207, val loss: 0.1662780845707113 val accuracy: 0.9418070444104135, patience: 3
epoch7 train loss: 0.07249298469140762, val loss: 0.14441461962732402 val accuracy: 0.9448698315467075, patience: 4
epoch8 train loss: 0.08901015566617587, val loss: 0.17206342111934314 val accuracy: 0.9402756508422665, patience: 5
epoch9 train loss: 0.09813730950217421, val loss: 0.1789132058620453 val accuracy: 0.947166921898928, patience: 6
epoch10 train loss: 0.11122148256839776, val loss: 0.17234653302214362 val accuracy: 0.9464012251148545, patience: 7
epoch11 train loss: 0.0806708085555129, val loss: 0.1486456888643178 val accuracy: 0.9433384379785605, patience: 8
epoch12 train loss: 0.077532178381594 22, val loss: 0.16138073801994324 val accuracy: 0.9486983154670751, patience: 9
epoch13 train loss: 0.08681874749500577, val loss: 0.12537297233939 17 val accuracy: 0.9555895865237366, patience: 0
epoch14 train loss: 0.11362083047264959, val loss: 0.17494092881679535 val accuracy: 0.9433384379785605, patience: 1
epoch15 train loss: 0.07273438318473537, val loss: 0.146782631223852 val accuracy: 0.9509954058192955, patience: 2
epoch16 train loss: 0.06373720738764216, val loss: 0.15071229162541303 val accuracy: 0.9532924961715161, patience: 3
epoch17 train loss: 0.09979676791444057, val loss: 0.1840338801795786 val accuracy: 0.9395099540581929, patience: 4
epoch18 train loss: 0.08295710430276103, val loss: 0.15275009721517563 val accuracy: 0.9448698315467075, patience: 5
epoch19 train loss: 0.05835377456756627, val loss: 0.1429820563333802 val accuracy: 0.9464012251148545, patience: 6
epoch20 train loss: 0.04733092235628425, val loss: 0.1620648642155257 val accuracy: 0.9486983154670751, patience: 7
epoch21 train loss: 0.05494103239985501, val loss: 0.1556076644496484 4 val accuracy: 0.9509954058192955, patience: 8
epoch22 train loss: 0.11784715141828467, val loss: 0.2408665350892327 val accuracy: 0.9241960183767228, patience: 9
epoch23 train loss: 0.06839014062794244, val loss: 0.20216940747977528 val accuracy: 0.9249617151607963, patience: 10
epoch24 train loss: 0.06627383128535456, val loss: 0.21138350665569305 val accuracy: 0.9234303215926493, patience: 11
epoch25 train loss: 0.07414591694023 55, val loss: 0.16271070323207162 val accuracy: 0.9479326186830015, patience: 12
epoch26 train loss: 0.07567812802224624, val loss: 0.14247851276939566 val accuracy: 0.9479326186830015, patience: 13
epoch27 train loss: 0.07040098373119424, val loss: 0.19290945340286603 val accuracy: 0.9379785604900459, patience: 14
epoch28 train loss: 0.04689115681117627 5, val loss: 0.12374418804591353 val accuracy: 0.9494640122511485, patience: 1
5
epoch29 train loss: 0.04298551203455866 5, val loss: 0.18533993580124594 val accuracy: 0.9479326186830015, patience: 1
6
epoch30 train loss: 0.06011595436167426, val loss: 0.12141064757650549 val accuracy: 0.9609494640122511, patience: 0
epoch31 train loss: 0.04567546397447586, val loss: 0.1438532363284718 val accuracy: 0.9555895865237366, patience: 1
epoch32 train loss: 0.04597296504931 92, val loss: 0.17830072275616907 val accuracy: 0.9318529862174579, patience: 2
epoch33 train loss: 0.04940375258646 36, val loss: 0.19506299766627225 val accuracy: 0.9226646248085758, patience: 3
epoch34 train loss: 0.05894333492146 76, val loss: 0.19341136786070737 val accuracy: 0.9379785604900459, patience: 4
epoch35 train loss: 0.07919367321017312, val loss: 0.15581736781380393 val accuracy: 0.9464012251148545, patience: 5
epoch36 train loss: 0.04447611475863108, val loss: 0.16138289462436328 val accuracy: 0.945635528330781, patience: 6
epoch37 train loss: 0.03440266479624806, val loss: 0.13125647333353528 val accuracy: 0.9548238897396631, patience: 7
epoch38 train loss: 0.04916396400913959 6, val loss: 0.21083865653384815 val accuracy: 0.9333843797856048, patience: 8
epoch39 train loss: 0.06606831047229 65, val loss: 0.15201217274774204 val accuracy: 0.9532924961715161, patience: 9
```

	precision	recall	f1-score	support
Artifactes	0.91	0.98	0.94	100
Cancer Regions	0.71	1.00	0.83	100
Normal Regions	0.92	0.87	0.89	100
Others	1.00	0.56	0.72	100
accuracy			0.85	400
macro avg	0.88	0.85	0.85	400
weighted avg	0.88	0.85	0.85	400





When I reload the path of best model, I get the test accuracy of 95%. But if just use the final model after all epochs it will be 85%. (Early stop make sense)

Test accuracy vs size of train set without early stop: (less image can get high train accuracy, but test accuracy very low)