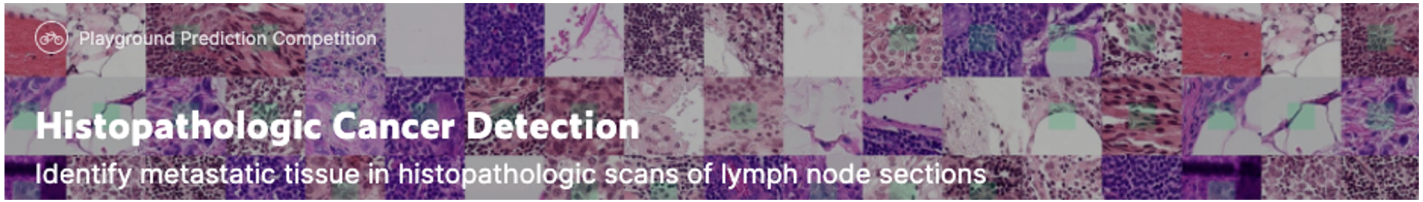
Final Project: Histopathologic Cancer Detection

Group 1

Mingyuan Li (Kevin), Silan Deng (Cyrus), Jaechul Roh (Harry), Hao Dong (Hao Dong)

Dataset - Histopathologic Cancer

Motivation & Overview



In [11]:

```
train_path = '../input/cancer-detection/train'
test_path = '../input/cancer-detection/test'

from sklearn.model_selection import train_test_split

train, val = train_test_split(df,
                              stratify=df.label,
                              test_size=0.2)

print("train size: ", len(train))
print("test size: ", len(val))
```

```
train size:  176020
test size:  44005
```

USER-FRIENDLY APPLICATION to *classify*

whether an image of a *pathology scan* contains *metastatic cancer*

"Metastatic Cancer"

Cancer that starts from one position and reach out to other parts of body

Kernel:	*Kaggle Notebook*
Framework used:	*PyTorch*
Training device:	CUDA
Model:	Custom-CNN vs. Resnet-18
Train : Test =	*8 : 2*

Custom Image Dataset

Pytorch "Dataset" class default functions

(__init__)

(__getitem__)

(__len__)

Image Transformation

Custom Image Dataset Class

```
class ImageDataset(Dataset):
    def __init__(self, df, image_dir, transform=None):
        super().__init__()
        self.df = df.values
        self.image_dir = image_dir
        self.transform = transform

    def __getitem__(self, idx):
        img_name, label = self.df[idx]
        img_path = os.path.join(self.image_dir, img_name + '.tif')

        img = Image.open(img_path)
        if self.transform is not None:
            img = self.transform(img)

        return img, label

    def __len__(self):
        return len(self.df)
```

+ Code + Markdown

```
transform = T.Compose([T.Resize(32),
                       T.RandomHorizontalFlip(p=0.5),
                       T.ToTensor(),
                       T.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

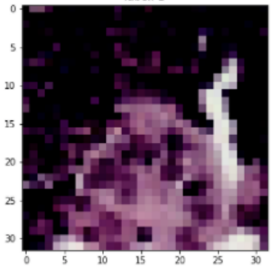
```
def imshow(image, label):
    label = str(label)
    plt.figure(figsize=(5, 5))
    plt.title("label: " + label)
    print(image.shape)
    plt.imshow(image.permute(1, 2, 0))
    plt.show()
```

Customized imshow function

```
train_path = '../input/cancer-detection/train'

train_img = ImageDataset(df=train, image_dir=train_path, transform=transform)
val_img = ImageDataset(df=val, image_dir=train_path, transform=transform)

train_loader = DataLoader(dataset=train_img, batch_size=128, shuffle=True)
val_loader = DataLoader(dataset=val_img, batch_size=64, shuffle=False)

image, label = val_img[110]
imshow(image, label)
```

Train and Validation Dataset & DataLoader

```
torch.Size([3, 32, 32])
label: 1
```



Output of the imshow function

Custom Convolutional Neural Network

```python
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class Net(nn.Module):
    def __init__(self):

        # batch = 128
        # input data shape: 128 x 32 x 32 x 3
        super(Net, self).__init__()

        # Conv layer
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5, stride=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.norm1 = nn.BatchNorm2d(6)

        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.norm2 = nn.BatchNorm2d(16)

        # FC layer
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 2)
        self.sigmoid = nn.Sigmoid()

        self.norm_fc1 = nn.BatchNorm1d(120)
        self.norm_fc2 = nn.BatchNorm1d(84)

    def forward(self, x):

        # relu do not require learning parameter
        # we don't need to make every pooling layer for block1 and 2

        x = self.conv1(x)
        x = F.relu(x)
        x = self.pool(x)
        x = self.norm1(x)

        x = self.conv2(x)
        x = self.pool(F.relu(x))
        x = self.norm2(x)

        # image flatten for FC
        x = x.view(-1, 16 * 5 * 5)

        # add batch normalization
        x = self.norm_fc1(F.relu(self.fc1(x)))
        x = self.norm_fc2(F.relu(self.fc2(x)))

        # OUTPUT Layer
        x = self.fc3(x)

        return x
```

Activate *nn.Module.__init__()*

Definition of Conv. layer features

Definition of FC_layer features

Forward Propagation of the input (x) through the network

Custom CNN STRUCTURE

 # (32x32x3)-> conv1 -> (28x28x6) -> pool -> (14x14x6) -> conv2 -> (10x10x16)->

 # pool -> (5x5x16) -> fc -> (120)-> fc -> (84) -> fc -> (2)

 # input channel = 3 (RGB)

 # default stride = 1

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

Example Calculation

*n(out) = [(32 + 2*0 - 5 / 1) + 1]*

= 28

= next n(in)

TRAINING the model

(Custom CNN)

Training function & CNN Result

```python
def train_model(net, train_loader, valid_loader,  num_epochs, criterion, optimizer, val_loss, device, save_name):

    if val_loss==None:
        best_val_loss = float("Inf")
    else:
        best_val_loss=val_loss
        print('Resume training')

    for epoch in range(num_epochs):  # loop over the dataset multiple times
        ############ Train phase ############
        net.train()
        running_loss = 0.0 # record the loss
        running_corrects = 0    # record correct prediction for classification
        for inputs, labels in tqdm(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)
Input image to CNN  outputs = net(inputs) ## step 1.

            loss = criterion(outputs, labels) ## step 2.  Calculate Loss

            optimizer.zero_grad() ## clear the previous gradients
            loss.backward() ## step 3. backpropagation - compute gradient
            optimizer.step() ## step 4. w = w - eta*w.grad -> update the model
            running_loss += loss.item()

            _, preds = torch.max(outputs.data, 1)  Output predicted label
            running_corrects += torch.sum(preds == labels.data)  ## step 2 - measure accuracy
        ## train epoch loss and accuracy
        train_loss = running_loss / len(train_loader)
        train_acc = running_corrects / float(len(train_loader.dataset))

        ############ Validation phase ############
        with torch.no_grad():  stop gradient computation
            net.eval()
            running_loss = 0.0  # loss for validation dataset
            running_corrects = 0    # correct prediction for validation dataset
            for inputs, labels in tqdm(valid_loader):
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = net(inputs) ## step 1
                loss = criterion(outputs, labels) ## step 2 - loss
                running_loss += loss.item()

                _, preds = torch.max(outputs.data, 1)
                running_corrects += torch.sum(preds == labels.data) ## step 2 - measure accuracy
            ## validation epoch loss and accuracy
            valid_loss = running_loss / len(valid_loader)
            valid_acc = running_corrects / float(len(valid_loader.dataset))

        print('Epoch [{}/{}], Train Loss: {:.4f}, Train Acc: {:.4f}, Valid Loss: {:.4f},  Valid Acc: {:.4f}'
              .format(epoch+1, num_epochs, train_loss, train_acc, valid_loss, valid_acc))

        if valid_loss < best_val_loss: ## save checkpoint when the validation loss is reduced
            best_val_loss = valid_loss
            save_checkpoint(save_name, net, optimizer, best_val_loss)

    print('Finished Training')
```

```python
import torch

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

num_epochs = 2
best_val_loss = None
criterion = nn.CrossEntropyLoss()    # Classification task
optimizer = optim.Adam(model.parameters(), lr=0.001)
save_path = f'cifar_net.pt'
model = model.to(device)

train_model(model, train_loader, val_loader, num_epochs, criterion, optimizer, best_val_loss, device, save_path)
```

Adam = AdaGrad + RMSProp

"optimization algorithm that can handle sparse gradients on noisy problems"

```
cuda
100% 1376/1376 [22:54<00:00, 1.34it/s]
100% 688/688 [05:41<00:00, 2.29it/s]
Epoch [1/2], Train Loss: 0.4001, Train Acc: 0.8204, Valid Loss: 0.4058,  Valid Acc: 0.8205
Model saved to cifar_net.pt
100% 1376/1376 [06:14<00:00, 3.63it/s]
100% 688/688 [01:35<00:00, 7.43it/s]
Epoch [2/2], Train Loss: 0.3477, Train Acc: 0.8485, Valid Loss: 0.3769,  Valid Acc: 0.8352
```

TRAINING the model

(Modified Resnet-18)

ORIGINAL Resnet-18

```
import torchvision
resnet18 = torchvision.models.resnet18(pretrained=True)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100% 44.7M/44.7M [00:01<00:00, 33.5MB/s]
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
resnet18.to(device)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

Built-in BasicBlock Class
(nn.Module with nn.Sequential)

Skip Connection: F(x) + x
(self.downsample)

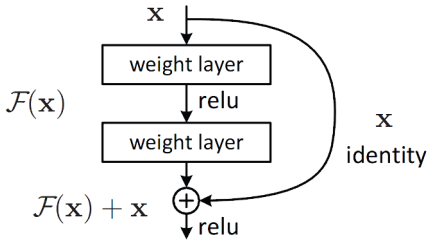
```
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Pretrained Resnet-18 Class (nn.Module)

Original output_features (number of classes): 1000

Model send to "CUDA" for faster learning process

Example code for BasicBlock (ResBlock in this case)




```python
class ResBlock(nn.Module):
  # Before Residual
  def __init__(self, in_channels, out_channels, D = False):
    super(ResBlock, self).__init__()
    self.stride = 1
    self.D = D

    if self.D == True:   # ResBlock-D: stride = 2
      self.stride = 2

    self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=self.stride, padding=1, bias=False)
    self.bn1 = nn.BatchNorm2d(out_channels)
    self.relu = nn.ReLU()
    self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
    self.bn2 = nn.BatchNorm2d(out_channels)

    # C1 == C2
    self.downsample = nn.Sequential() # Identity

    if in_channels != out_channels: # NOT Identity mapping
      self.downsample = nn.Sequential(
          nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=self.stride, padding=0, bias=False),
          nn.BatchNorm2d(out_channels)
      )

  def forward(self, x):

    # F(x)
    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)
    out = self.conv2(out)
    out = self.bn2(out)

    out = self.downsample(x) + out  # Skip connection F(x) + x

    out = self.relu(out)

    return out
```

Providing *specific condition*

when to NOT process the *RESIDUAL (Skip) connection*

```python
# Original Resnet
class Modified_Resnet(nn.Module):
  # Before Residual
  def __init__(self, train_mode=True):
    super(Modified_Resnet, self).__init__()
    self.train_mode = train_mode

    self.conv1 = nn.Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    self.bn1 = nn.BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    self.relu = nn.ReLU(inplace=True)
    self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)

    # Feature layer modify
    self.layer1 = nn.Sequential(
        ResBlock(64, 64),
        ResBlock(64, 64)
        )

    self.layer2 = nn.Sequential(
        ResBlock(64, 128, D=True),   # Residual
        ResBlock(128, 128)
        )

    self.layer3 = nn.Sequential(
        ResBlock(128, 256, D=True),  # Residual
        ResBlock(256, 256)
        )

    self.layer4 = nn.Sequential(
        ResBlock(256, 512),          # Residual
        ResBlock(512, 512)
        )

    self.avgpool = nn.AdaptiveAvgPool2d((1, 1))

    # FC
    self.FC = nn.Sequential(
        new_FC(512, 512),
        new_FC(512, 256)
    )

    self.final_train_classifier = nn.Linear(in_features=256, out_features=230)
    self.final_val_classifier = nn.Linear(in_features=256, out_features=49)
    self.final_triplet_classifier = nn.Linear(in_features=256, out_features=256)

  def forward(self, x):
    # Resnet18 start
    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)
```

"Code written just for the understanding of Resnet Skip Connection"

ORIGINAL Resnet-18 (using torchsummary)

```
from torchsummary import summary
summary(resnet18.to(device), (3, 32, 32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 16, 16]           9,408
       BatchNorm2d-2           [-1, 64, 16, 16]             128
              ReLU-3           [-1, 64, 16, 16]               0
         MaxPool2d-4             [-1, 64, 8, 8]               0
            Conv2d-5             [-1, 64, 8, 8]          36,864
       BatchNorm2d-6             [-1, 64, 8, 8]             128
              ReLU-7             [-1, 64, 8, 8]               0
            Conv2d-8             [-1, 64, 8, 8]          36,864
       BatchNorm2d-9             [-1, 64, 8, 8]             128
             ReLU-10             [-1, 64, 8, 8]               0
       BasicBlock-11             [-1, 64, 8, 8]               0
           Conv2d-12             [-1, 64, 8, 8]          36,864
      BatchNorm2d-13             [-1, 64, 8, 8]             128
             ReLU-14             [-1, 64, 8, 8]               0
           Conv2d-15             [-1, 64, 8, 8]          36,864
      BatchNorm2d-16             [-1, 64, 8, 8]             128
             ReLU-17             [-1, 64, 8, 8]               0
       BasicBlock-18             [-1, 64, 8, 8]               0
           Conv2d-19            [-1, 128, 4, 4]          73,728
      BatchNorm2d-20            [-1, 128, 4, 4]             256
             ReLU-21            [-1, 128, 4, 4]               0
           Conv2d-22            [-1, 128, 4, 4]         147,456
      BatchNorm2d-23            [-1, 128, 4, 4]             256
           Conv2d-24            [-1, 128, 4, 4]           8,192
      BatchNorm2d-25            [-1, 128, 4, 4]             256
             ReLU-26            [-1, 128, 4, 4]               0
       BasicBlock-27            [-1, 128, 4, 4]               0
           Conv2d-28            [-1, 128, 4, 4]         147,456
      BatchNorm2d-29            [-1, 128, 4, 4]             256
             ReLU-30            [-1, 128, 4, 4]               0
           Conv2d-31            [-1, 128, 4, 4]         147,456
      BatchNorm2d-32            [-1, 128, 4, 4]             256
             ReLU-33            [-1, 128, 4, 4]               0
       BasicBlock-34            [-1, 128, 4, 4]               0
           Conv2d-35            [-1, 256, 2, 2]         294,912
      BatchNorm2d-36            [-1, 256, 2, 2]             512
             ReLU-37            [-1, 256, 2, 2]               0
           Conv2d-38            [-1, 256, 2, 2]         589,824
      BatchNorm2d-39            [-1, 256, 2, 2]             512
           Conv2d-40            [-1, 256, 2, 2]          32,768
      BatchNorm2d-41            [-1, 256, 2, 2]             512
             ReLU-42            [-1, 256, 2, 2]               0
       BasicBlock-43            [-1, 256, 2, 2]               0
           Conv2d-44            [-1, 256, 2, 2]         589,824
      BatchNorm2d-45            [-1, 256, 2, 2]             512
             ReLU-46            [-1, 256, 2, 2]               0
           Conv2d-47            [-1, 256, 2, 2]         589,824
      BatchNorm2d-48            [-1, 256, 2, 2]             512
             ReLU-49            [-1, 256, 2, 2]               0
       BasicBlock-50            [-1, 256, 2, 2]               0
           Conv2d-51            [-1, 512, 1, 1]       1,179,648
      BatchNorm2d-52            [-1, 512, 1, 1]           1,024
             ReLU-53            [-1, 512, 1, 1]               0
           Conv2d-54            [-1, 512, 1, 1]       2,359,296
      BatchNorm2d-55            [-1, 512, 1, 1]           1,024
           Conv2d-56            [-1, 512, 1, 1]         131,072
      BatchNorm2d-57            [-1, 512, 1, 1]           1,024
             ReLU-58            [-1, 512, 1, 1]               0
       BasicBlock-59            [-1, 512, 1, 1]               0
           Conv2d-60            [-1, 512, 1, 1]       2,359,296
      BatchNorm2d-61            [-1, 512, 1, 1]           1,024
             ReLU-62            [-1, 512, 1, 1]               0
           Conv2d-63            [-1, 512, 1, 1]       2,359,296
      BatchNorm2d-64            [-1, 512, 1, 1]           1,024
             ReLU-65            [-1, 512, 1, 1]               0
       BasicBlock-66            [-1, 512, 1, 1]               0
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                 [-1, 1000]         513,000
```

Modified Resnet-18

```python
class modified_resnet18(nn.Module):
    def __init__(self):
        super(modified_resnet18, self).__init__()
        self.resnet18 = torchvision.models.resnet18(pretrained=True)

        for param in self.resnet18.parameters():
            param.requres_grad = False

        modified_fc = nn.Linear(in_features=512, out_features=2)
        self.resnet18.fc = modified_fc

    def forward(self, x):
        return self.resnet18(x)
```

FIX the feature extraction layer (conv, maxpool, relu, conv, …)

JUST modify the FC layers

MODIFIED output_features (number of classes):

```
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=2, bias=True)
)
```

"2"

BatchNorm2d-55	[-1, 512, 1, 1]	1,024
Conv2d-56	[-1, 512, 1, 1]	131,072
BatchNorm2d-57	[-1, 512, 1, 1]	1,024
ReLU-58	[-1, 512, 1, 1]	0
BasicBlock-59	[-1, 512, 1, 1]	0
Conv2d-60	[-1, 512, 1, 1]	2,359,296
BatchNorm2d-61	[-1, 512, 1, 1]	1,024
ReLU-62	[-1, 512, 1, 1]	0
Conv2d-63	[-1, 512, 1, 1]	2,359,296
BatchNorm2d-64	[-1, 512, 1, 1]	1,024
ReLU-65	[-1, 512, 1, 1]	0
BasicBlock-66	[-1, 512, 1, 1]	0
AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
Linear-68	[-1, 2]	1,026

TESTING the output of the model

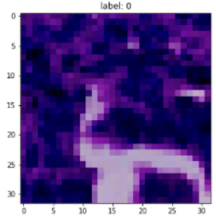
TRAIN & TEST the Modified Resnet-18

```
num_epochs = 2
resnet18_optimizer = optim.Adam(modified_Resnet.parameters())

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

best_val_loss = None
criterion = nn.CrossEntropyLoss()

train_model(modified_Resnet,
            train_loader,
            val_loader,
            num_epochs,
            criterion,
            resnet18_optimizer,
            best_val_loss,
            device,
            save_name=None)
```

```
cuda
100% 1376/1376 [26:54<00:00, 1.15it/s]
100% 688/688 [06:35<00:00, 2.09it/s]
Epoch [1/2], Train Loss: 0.3246, Train Acc: 0.8617, Valid Loss: 0.3212,  Valid Acc: 0.8666
100% 1376/1376 [06:41<00:00, 3.12it/s]
100% 688/688 [01:41<00:00, 12.70it/s]
Epoch [2/2], Train Loss: 0.2638, Train Acc: 0.8915, Valid Loss: 0.2790,  Valid Acc: 0.8877
Finished Training
```

```
test_path = "../input/cancer-detection/test"
submission_df = pd.read_csv("../input/cancer-detection/sample_submission.csv")

test_img = ImageDataset(df=submission_df, image_dir=test_path, transform=transform)
test_loader = DataLoader(dataset=test_img, batch_size=32, shuffle=False)

image, label = test_img[20]
imshow(image, label)
```

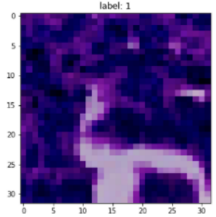
```
torch.Size([3, 32, 32])
label: 0
```



+ Code + Markdown

```
img = image.cuda()
img = img.unsqueeze(0)
output = modified_Resnet(img)
_, preds = torch.max(output.data, 1)
pred = preds.detach().cpu()
pred = int(pred)

imshow(image, pred)
```

```
torch.Size([3, 32, 32])
label: 1
```



Test data DataLoader

Expected output on the *GUI*

Chatbot &

Graphical User Interface (GUI)

Chatbot & GUI

The goal of this GUI is to make it easier for users to use models to predict cancer. Therefore, we made the interface and steps as simple as possible. Besides, when a user uploads a file in an invalid format, the GUI will give an error and guide the user to upload the correct form.

1. create the GUI window

```
root = Tk()
root.title('Metastatic Cancer Identification')
root.resizable(width = False, height = False)
##set window's width and height unresizable
```

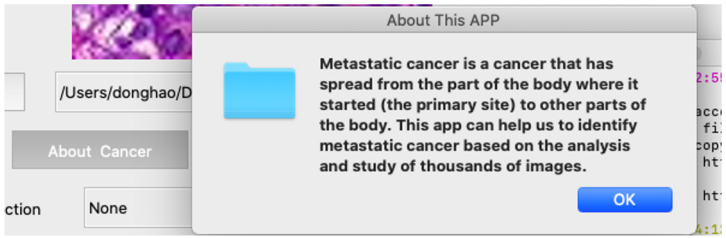
2. Set a button that allows user upload their picture from their computer and show it in the canvas

```
##Input part
input_frm = Frame(root)
input_frm.grid(row = 1, column = 0)
upload_btn = Button(input_frm, text = 'Upload An Image', command = process) ##the button calls the process()
upload_btn.grid(row = 1, column = 0, ipadx = '15', ipady = '5', padx = '10', pady = '5')$
input_var = StringVar(input_frm, 'Please click the button on the left to upload your image')
input_entry = Entry(input_frm, textvariable = input_var, width = 50)
input_entry['state'] = 'readonly'
input_entry.grid(row = 1, column = 1, ipadx = '30', ipady = '5', padx = '10', pady = '5')
```

3. Set an area to show the result of our prediction


```
##Output part
output_frm = Frame(root)
output_frm.grid(row = 2, column = 0)
output_var = StringVar(output_frm, 'Result:')
##output_var.set('Result:')
output_entry = Entry(output_frm, textvariable = output_var)
output_entry['state'] = 'readonly'
output_entry.grid(row = 2, column = 1, ipadx = '20', ipady = '5', padx = '10', pady = '5')
```

4. Create a button to show the information about cancers and this APP.




```
about_btn2 = tk.Button(output_frm, text = 'About  Cancer', command = okCallBack)
about_btn2.grid(row = 2, column = 0, ipadx = '15', ipady = '5', padx = '10', pady = '5')
```

```
medi_frm = Frame(root)
medi_frm.grid(row = 3, column = 0)
info = Label(medi_frm, text = 'Medical Prediction', anchor = W, justify = 'left')
info.grid(row = 3, column = 0, ipadx = '10', ipady = '5', padx = '5', pady = '5')
medi_var = StringVar(medi_frm,'None')
medi_entry = Entry(medi_frm, textvariable = medi_var, width = 40)
medi_entry['state'] = 'readonly'
medi_entry.grid(row = 3, column = 1, ipadx = '30', ipady = '5', padx = '5', pady = '5')
```

5. Add a “medical prediction” part. In this part, we use You probably have/ do not cancer to replace the prediction of our model, which makes easier for users to understand the result.

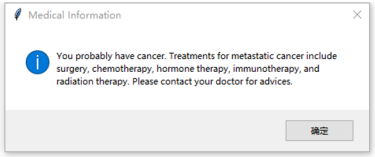
6. Defined a process() function

```
def process(): ##generate diagnoses
    ## load file
    selectFile = tk.filedialog.askopenfilename()

    ## check file type
    filelist = ['tiff', 'png', 'jpeg'] ##支持的文件格式，可以添加其它的
    if  imghdr.what (selectFile) not in filelist:
        ##当用户上传非特定格式时使用弹窗报错
        app = Tk()
        app.withdraw()
        showerror(title = "ERROR",
              message = "Please upload a valid image ")
        canvas.delete('all')
        canvas.create_text(150, 150, text = 'Upload An Image', font = ('Arial Bold',25))
        input_var.set( 'Please click the button on the left to upload your image')
        return

    else:
       input_var.set(selectFile)

    ##show image
    img_file = Image.open(selectFile)
    img_file = img_file.resize((300, 300))#enlarge the picture size
    render = ImageTk.PhotoImage(img_file)
    img = Label(canvas_frm, imag = render)
    img.image = render
    img.grid(row = 0, column = 0)
```

When user uploads an invalid input, the message box will show an error.

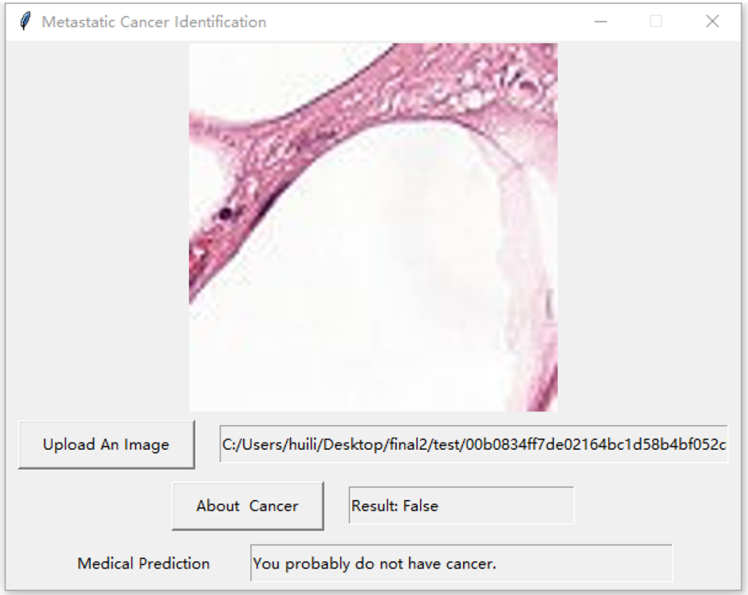
```
    #analyze image
    try:
        resnet_img = Image.open(selectFile)
        resnet_img = transform(resnet_img)
        resnet_img = resnet_img.cuda()
        resnet_img = resnet_img.unsqueeze(0)
        output = modified_Resnet(resnet_img)
        _, preds = torch.max(output.data, 1)
        pred = preds.detach().cpu()
        pred = int(pred)
    except:
        ##pred = 1 ## for test if needed
        tk.messagebox.showinfo('ERROR','Model use is failed.')
        return

    if pred == 1:
        output_var.set('Result: True')
        medi_var.set('You probably have cancer.')
        tk.messagebox.showinfo('Medical Information','You probably have cancer. \
Treatments for metastatic cancer include surgery, chemotherapy, hormone therapy, \
immunotherapy, and radiation therapy. Please contact your doctor for advices.')
    elif pred == 0:
        output_var.set('Result: False')
        medi_var.set('You probably do not have cancer.')
        tk.messagebox.showinfo('Medical Information','You probably do not have cancer. \
Treatments for metastatic cancer include surgery, chemotherapy, hormone therapy, \
immunotherapy, and radiation therapy. Please contact your doctor for advices.')
```

Analyze the image and give the prediction or show an error if it failed to analyze the image.



The information in message box makes it easier for users to understand the predicted results

An overview of the GUI



Further Improvement and Studies

Improvements and Evaluation:

- Could have used higher number of epochs for trainig:
  - Increae training and validation ACCURACY
  - OVERFITTING issue
- More COMPLEX chatbot system with INTRICATE conversations possible

---

Further Studies:

- Construct neural network model SUITABLE for medical applications
  - Combination of differnet neural networks

Reference

- “Histopathologic Cancer Detection | Kaggle.” *Kaggle*, 22 Jan. 2022, www.kaggle.com/c/histopathologic-cancer-detection.
- “Understanding Advanced and Metastatic Cancer.” *Understanding Advanced and Metastatic Cancer*, www.cancer.org/treatment/understanding-your-diagnosis/advanced-cancer/what-is.html. Accessed 29 Jan. 2022.
- Contributor, TechTarget. “Convolutional Neural Network.” *SearchEnterpriseAI*, 27 Apr. 2018, www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network.
- Shakhadri, Syed Abdul Gaffar. “What Is ResNet | Build ResNet from Scratch With Python.” *Analytics Vidhya*, 8 June 2021, www.analyticsvidhya.com/blog/2021/06/build-resnet-from-scratch-with-python.
- “Torchvision.Models — Torchvision 0.11.0 Documentation.” *Torchvision*, pytorch.org/vision/stable/models.html. Accessed 29 Jan. 2022.
- “Training a Classifier — PyTorch Tutorials 1.10.1+cu102 Documentation.” *PyTorch*, pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. Accessed 29 Jan. 2022.
- “How to Calculate Output Sizes after a Convolution Layer in a Configuration File?” *Stack Overflow*, 4 June 2019, stackoverflow.com/questions/56450969/how-to-calculate-output-sizes-after-a-convolution-layer-in-a-configuration-file/56452756.