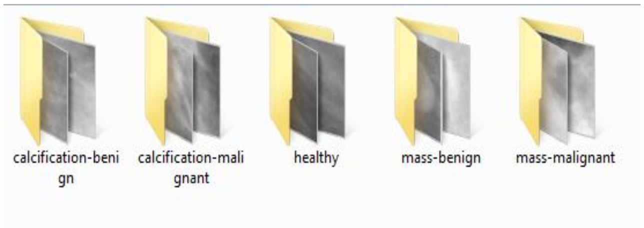
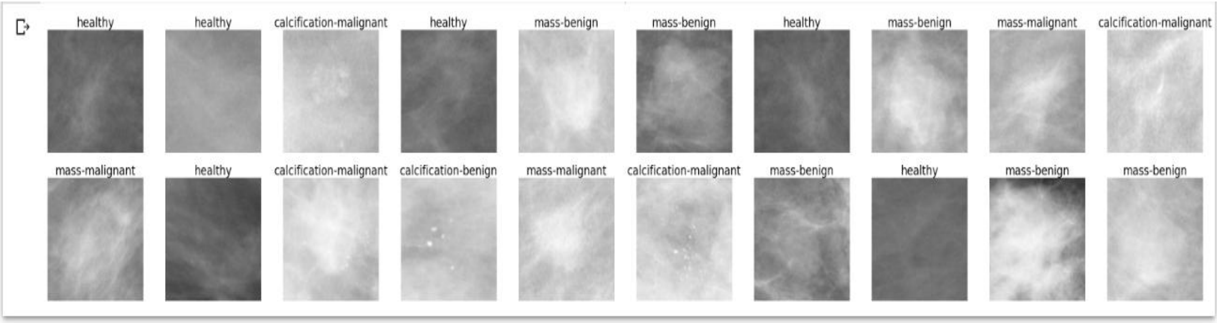
Breast Cancer Classification Based on Mammograms with Deep Learning:

There are 5 classes for each mammogram:

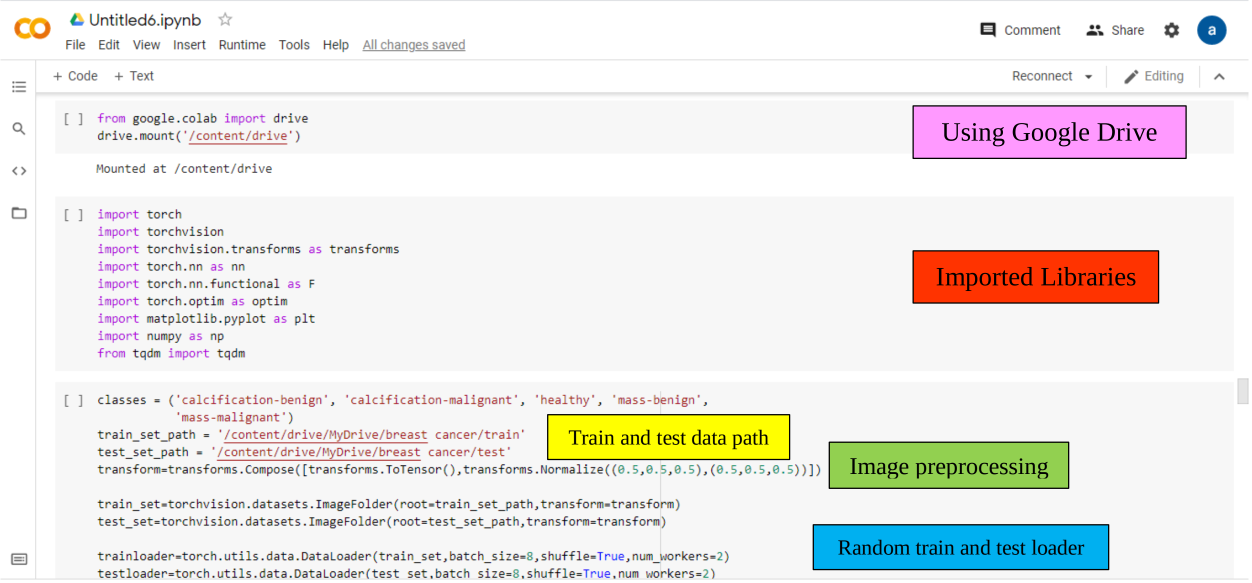


0 1 2 3 4

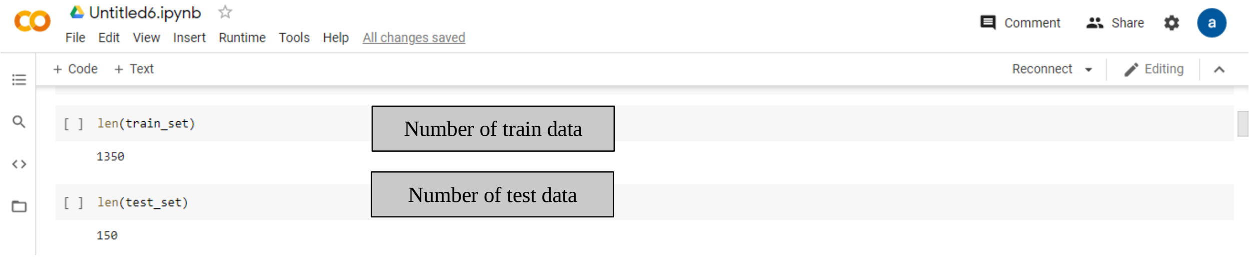
Random examples of images:



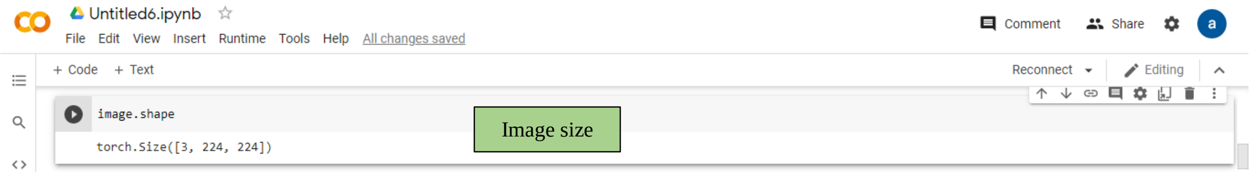
Here are the codes for mammogram classification in Python in Google Colab with PyTorch:



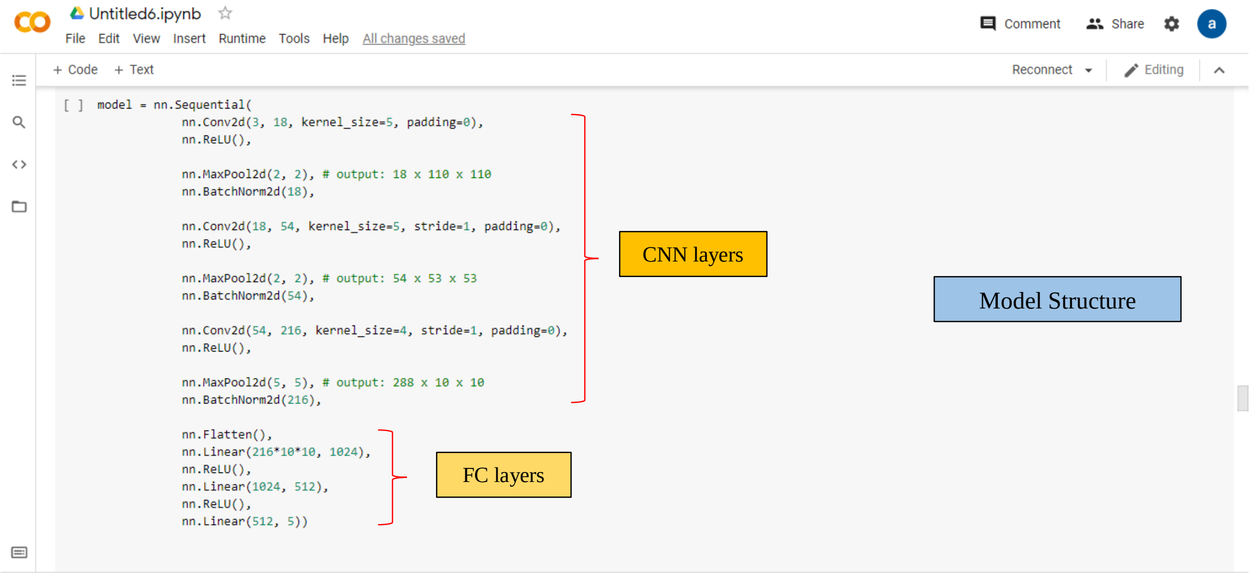
```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[ ] import torch
    import torchvision
    import torchvision.transforms as transforms
    import torch.nn as nn
    import torch.nn.functional as F
    import torch.optim as optim
    import matplotlib.pyplot as plt
    import numpy as np
    from tqdm import tqdm

[ ] classes = ('calcification-benign', 'calcification-malignant', 'healthy', 'mass-benign',
               'mass-malignant')
    train_set_path = '/content/drive/MyDrive/breast cancer/train'
    test_set_path = '/content/drive/MyDrive/breast cancer/test'
    transform=transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))])

    train_set=torchvision.datasets.ImageFolder(root=train_set_path,transform=transform)
    test_set=torchvision.datasets.ImageFolder(root=test_set_path,transform=transform)

    trainloader=torch.utils.data.DataLoader(train_set,batch_size=8,shuffle=True,num_workers=2)
    testloader=torch.utils.data.DataLoader(test_set,batch_size=8,shuffle=True,num_workers=2)
```



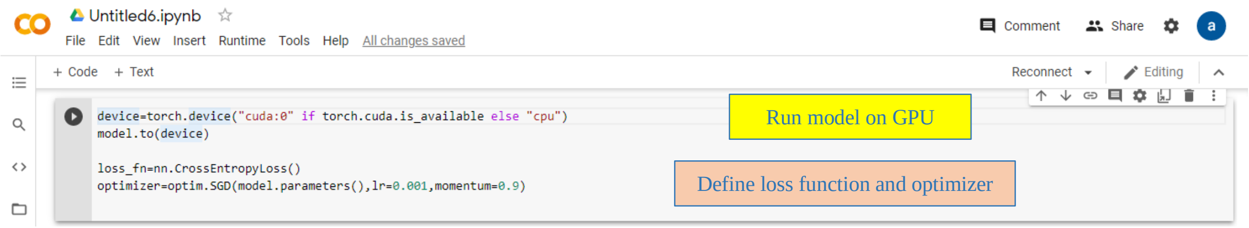
```
[ ] len(train_set)

    1350

[ ] len(test_set)

    150
```



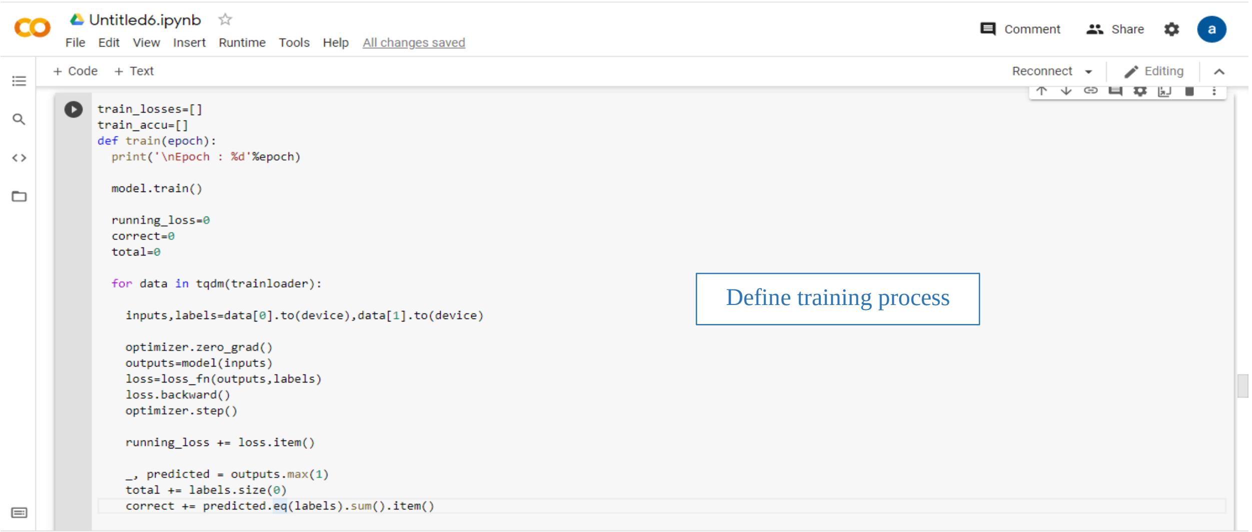
```
image.shape

torch.Size([3, 224, 224])
```



```
[ ] model = nn.Sequential(
                nn.Conv2d(3, 18, kernel_size=5, padding=0),
                nn.ReLU(),

                nn.MaxPool2d(2, 2), # output: 18 x 110 x 110
                nn.BatchNorm2d(18),

                nn.Conv2d(18, 54, kernel_size=5, stride=1, padding=0),
                nn.ReLU(),

                nn.MaxPool2d(2, 2), # output: 54 x 53 x 53
                nn.BatchNorm2d(54),

                nn.Conv2d(54, 216, kernel_size=4, stride=1, padding=0),
                nn.ReLU(),

                nn.MaxPool2d(5, 5), # output: 288 x 10 x 10
                nn.BatchNorm2d(216),

                nn.Flatten(),
                nn.Linear(216*10*10, 1024),
                nn.ReLU(),
                nn.Linear(1024, 512),
                nn.ReLU(),
                nn.Linear(512, 5))
```

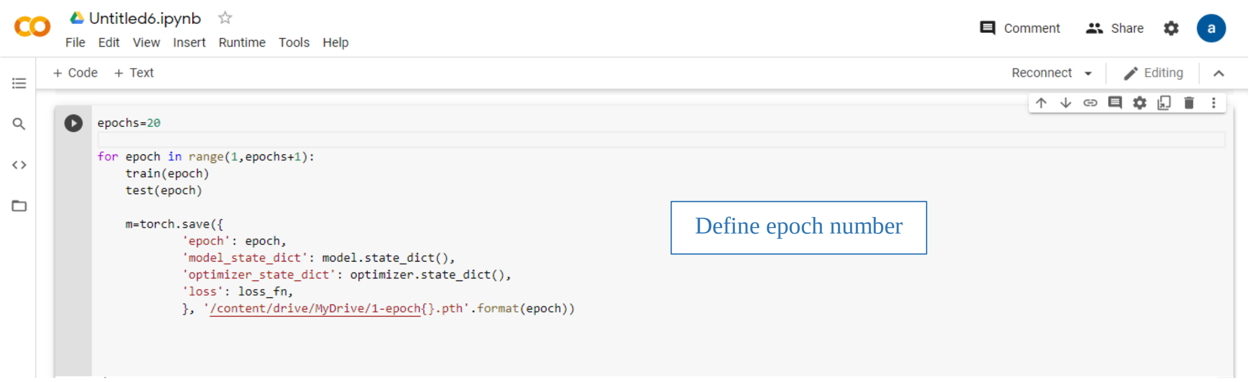

Untitled6.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Reconnect
Editing
Run model on GPU
Define loss function and optimizer

```
device=torch.device("cuda:0" if torch.cuda.is_available else "cpu")
model.to(device)

loss_fn=nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=0.001,momentum=0.9)
```



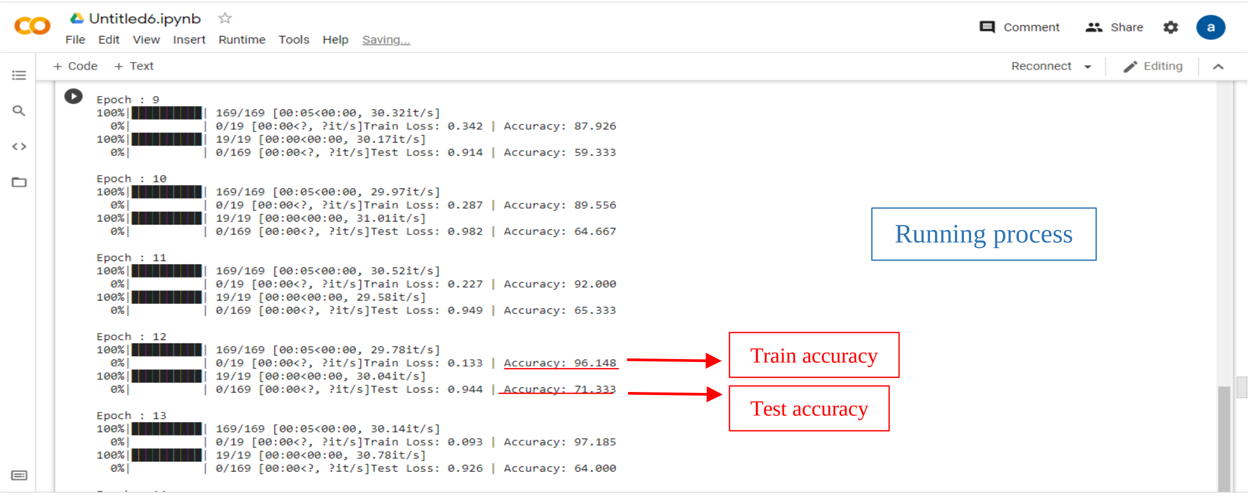
```
train_losses=[]
train_accu=[]
def train(epoch):
  print('\nEpoch : %d'%epoch)

  model.train()

  running_loss=0
  correct=0
  total=0

  for data in tqdm(trainloader):

    inputs,labels=data[0].to(device),data[1].to(device)

    optimizer.zero_grad()
    outputs=model(inputs)
    loss=loss_fn(outputs,labels)
    loss.backward()
    optimizer.step()

    running_loss += loss.item()

    _, predicted = outputs.max(1)
    total += labels.size(0)
    correct += predicted.eq(labels).sum().item()
```

Define training process



```
epochs=20

for epoch in range(1,epochs+1):
    train(epoch)
    test(epoch)

    m=torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss_fn,
            }, '/content/drive/MyDrive/1-epoch{}.pth'.format(epoch))
```

Define epoch number

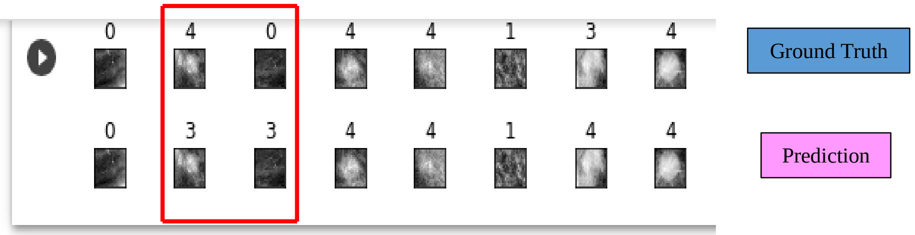


```
Epoch : 9
100%|██████████| 169/169 [00:05<00:00, 30.32it/s]
  0%|          | 0/19 [00:00<?, ?it/s]Train Loss: 0.342 | Accuracy: 87.926
100%|██████████| 19/19 [00:00<00:00, 30.17it/s]
  0%|          | 0/169 [00:00<?, ?it/s]Test Loss: 0.914 | Accuracy: 59.333

Epoch : 10
100%|██████████| 169/169 [00:05<00:00, 29.97it/s]
  0%|          | 0/19 [00:00<?, ?it/s]Train Loss: 0.287 | Accuracy: 89.556
100%|██████████| 19/19 [00:00<00:00, 31.01it/s]
  0%|          | 0/169 [00:00<?, ?it/s]Test Loss: 0.982 | Accuracy: 64.667

Epoch : 11
100%|██████████| 169/169 [00:05<00:00, 30.52it/s]
  0%|          | 0/19 [00:00<?, ?it/s]Train Loss: 0.227 | Accuracy: 92.000
100%|██████████| 19/19 [00:00<00:00, 29.58it/s]
  0%|          | 0/169 [00:00<?, ?it/s]Test Loss: 0.949 | Accuracy: 65.333

Epoch : 12
100%|██████████| 169/169 [00:05<00:00, 29.78it/s]
  0%|          | 0/19 [00:00<?, ?it/s]Train Loss: 0.133 | Accuracy: 96.148
100%|██████████| 19/19 [00:00<00:00, 30.04it/s]
  0%|          | 0/169 [00:00<?, ?it/s]Test Loss: 0.944 | Accuracy: 71.333

Epoch : 13
100%|██████████| 169/169 [00:05<00:00, 30.14it/s]
  0%|          | 0/19 [00:00<?, ?it/s]Train Loss: 0.093 | Accuracy: 97.185
100%|██████████| 19/19 [00:00<00:00, 30.78it/s]
  0%|          | 0/169 [00:00<?, ?it/s]Test Loss: 0.926 | Accuracy: 64.000
```

Running process

Train accuracy

Test accuracy



Wrong prediction