fina_project

May 14, 2020

Created on Tue Apr 21 11:10:11 2020

@author: Deeps

[1]:
```
import warnings
warnings.filterwarnings("ignore")
#!pip install torch
#!pip install torchsummary
#!pip install torchvision
```

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm # Displays a progress bar

import torch
from torch.utils import data
from torchsummary import summary
from torch import nn
from torch import optim
from PIL import Image, ImageFilter
from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matrix
import torch.nn.functional as F
from torchvision import datasets,models, transforms
from torch.utils.data import Dataset, Subset, DataLoader, random_split
from data_gen import Dataset
from augment import Augmentations as aug
from visualization  import visualizer
```

0.1 Load the dataset and train, val, test splits

[3]:
```
num_epochs = 10
num_classes = 2
batch_size = 5
num_rotations = 2 #180
rotations = [0,180]
```

```
learning_rate = 0.001
image_size = (748,500)
resnet_resize = (374,250)
datacsv = pd.read_csv("data.csv")
cv_results = []
```

0.1.1 ————Data Augmentation———

Since we only have 119 images, and I want to keep the test set fixed at 20, with exactly 3 examples of class 0, I first split the data into 99 and 20. I then only apply augmentation to the 99 for training. I do not apply this augmentation function to the test data. only the transformations are applied.

[4]:
```
all_ = len(datacsv['image'])
train_size = int(all_ *(5/6))
test_size = all_-train_size

traincsv = datacsv[:train_size]
testcsv = datacsv[train_size:]
```

[5]:
```
data_base_aug = dict(traincsv)
data_base_aug["image"] = list(data_base_aug["image"])
data_base_aug["label"] = list(data_base_aug["label"])

if num_rotations>0:
    rot = aug(rotations)
    for k in range(len(data_base_aug["image"])):
        result = rot.
↪rotate_append(data_base_aug["image"][k],data_base_aug["label"][k])
        traincsv = traincsv.append(result,ignore_index = True)
```

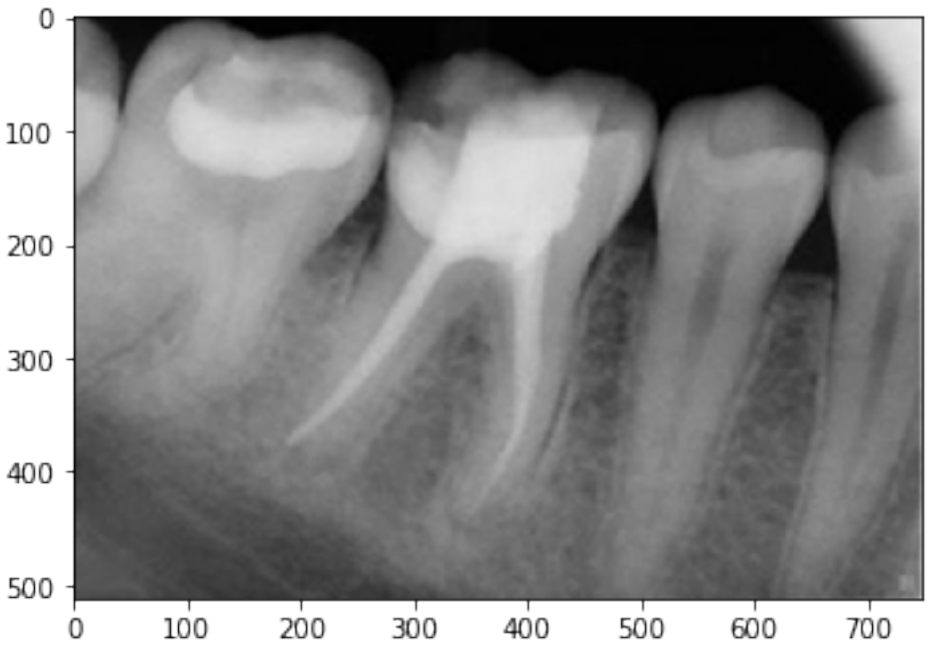
0.2 ————inspect————

[14]:
```
x_train = traincsv['image']
x_test = testcsv['image']
y_train = traincsv['label']
y_test = testcsv['label']

#x_train,x_test,y_train,y_test =tts(datacsv["image"],␣
↪datacsv["label"],test_size=1/6, shuffle=False, random_state=42)
partition = {'train':list(x_train),'validation':list(x_test)}
labels = {}

combinedforlabels = traincsv.append(testcsv,ignore_index = True)

for k in range(len(combinedforlabels['image'])):
    labels['%s'%combinedforlabels['image'][k]] = combinedforlabels['label'][k]
```

```
print("Total original Images: %s"%all_)
print('Train images( after augmentation): %s'%(len(y_train)), ', Test images:␣
 ↪%s'%(len(y_test)))
print('Train Positives(with augmentation): %s'%(np.sum(y_train)/len(y_train)),␣
 ↪', Test Positives(No augmentation): %s'%(np.sum(y_test)/len(y_test)))
print('Total Real dataset positives: %s'%(np.sum(datacsv["label"])/
 ↪len(datacsv["label"])))

view_image = Image.open('Dataset/' + str(int(combinedforlabels["image"][88])) +␣
 ↪'.jpg').convert('LA')
plt.imshow(view_image)
```

```
Total original Images: 119
Train images( after augmentation): 183 , Test images: 20
Train Positives(with augmentation): 0.819672131147541 , Test Positives(No
augmentation): 0.85
Total Real dataset positives: 0.8823529411764706
```

[14]: <matplotlib.image.AxesImage at 0x2ef2a968f08>



1 ————-Create dataset generator————-

[16]:
```
for cv in range(1):
    #x_train,x_test =tts(datacsv["image"],test_size=1/6, shuffle=False)

    ########################################################################

    training_set = Dataset(partition["train"],labels, image_size)
    training_generator = data.DataLoader(training_set, batch_size=batch_size,␣
↪shuffle=True)

    validation_set = Dataset(partition['validation'], labels, image_size)
    validation_generator = data.DataLoader(validation_set,␣
↪batch_size=batch_size, shuffle=True)



    ########################################################
    class Network(nn.Module):
        def __init__(self):
            super().__init__()
            self.resnet =  torch.hub.load('pytorch/vision', 'resnet34',␣
↪pretrained=True)
            self.num_ftrs = self.resnet.fc.in_features
            self.resnet.fc = nn.Linear(self.num_ftrs,8)
            self.fc1 = nn.Linear(8, num_classes)

        def forward(self,x):
            # TODO: Design your own network, implement forward pass here
            x = F.dropout(F.relu(self.resnet(x)),0.2)#3*748*512 -> 6*744*508 ->␣
↪6*372*254
            out = F.softmax(self.fc1(x))
            return out

    device = "cuda" if torch.cuda.is_available() else "cpu" # Configure device
    model = Network().to(device)

    ct = 0
    model_size = len([1 for k in model.children()])
    print("model components: ", model_size)
    for child in model.children():
        ct+=1
        if ct==1:
            for param in child.parameters():
                param.requires_grad=False

    model.resnet.fc.weight.requires_grad = True
    model.resnet.fc.bias.requires_grad = True
```

```
   #model.resnet.layer4[1].conv2.weight.requires_grad = True

   model.fc1.weight.requires_grad = True
   model.fc1.bias.requires_grad = True


   weights = torch.tensor([0.6,0.4])
   criterion = nn.CrossEntropyLoss(weight = weights) # Specify the loss layer
   optimizer = optim.Adam(model.parameters()) # Specify optimizer and assign␣
↪trainable parameters to it, weight_decay is L2 regularization strength


   def train(model, training_generator, num_epoch = num_epochs): # Train the␣
↪model
       print("Start training...")
       model.train() # Set the model to training mode
       train_val_loss = {'train':[], 'validation':[]}
       for i in range(num_epoch):
           running_loss = []
           accuracy = []
           for batch, label in tqdm(training_generator):
               batch = batch.to(device)
               label = label.to(device)
               optimizer.zero_grad() # Clear gradients from the previous␣
↪iteration
               pred = model(batch) # This will call Network.forward() that you␣
↪implement
               loss = criterion(pred, label) # Calculate the loss
               running_loss.append(loss.item())
               correct = (torch.argmax(pred,dim=1)==label).sum().item()
               accuracy.append(correct/batch_size)

               loss.backward() # Backprop gradients to all tensors in the␣
↪network
               optimizer.step() # Update trainable weights

           val_loss, val_accuracy = evaluate(model, validation_generator)
           print("Epoch {}   loss:{}   eval_loss:{}   accuracy:{}   ␣
↪eval_accuracy: {}".format(i+1,np.mean(running_loss), val_loss, np.
↪mean(accuracy),val_accuracy)) # Print the average loss for this epoch

           train_val_loss['train'].append(np.mean(running_loss))
           train_val_loss['validation'].append(np.mean(val_loss))

           if (i+1)%10==0 or i==(num_epoch-1) or val_accuracy>0.85:
```

```
                try:
                    torch.save(model, "Trained_model/teeth_model_%s.pth"%(i+1))
                    print("Model saved at Trained _model")
                except:
                    print("Could not save model")
                if val_accuracy>0.85:
                    print("early stop")
                    break
        print("Done!")
        return train_val_loss


    def evaluate(model, validation_generator): # Evaluate accuracy on validation / test set
        model.eval() # Set the model to evaluation mode
        correct = 0
        total = 0
        val_run_loss = []
        with torch.set_grad_enabled(True): # Do not calculate grident to speed up computation
            for batch, label in tqdm(validation_generator):
                batch = batch.to(device)
                label = label.to(device)
                pred = model(batch)
                loss = criterion(pred, label).item()
                val_run_loss.append(loss)
                correct += (torch.argmax(pred,dim=1)==label).sum().item()
                total+=batch_size
        acc = correct/total
        #print("Evaluation accuracy: {}".format(acc))
        return (np.mean(loss),acc)

    def predict(model, validation_generator): # Evaluate accuracy on validation / test set
        model.eval() # Set the model to evaluation mode
        results = {'pred':[],'real':[]}
        with torch.set_grad_enabled(True): # Do not calculate grident to speed up computation
            for batch, label in tqdm(validation_generator):
                batch = batch.to(device)
                label = label.to(device)
                pred = model(batch)
                pred = torch.argmax(pred,dim=1)
                for k in range(len(pred)):
                    results['pred'].append(pred[k].item())
                    results['real'].append(label[k].item())
        return results
```

```
Using cache found in C:\Users\Deeps/.cache\torch\hub\pytorch_vision_master

model components:  2
```

1.1 train and eval

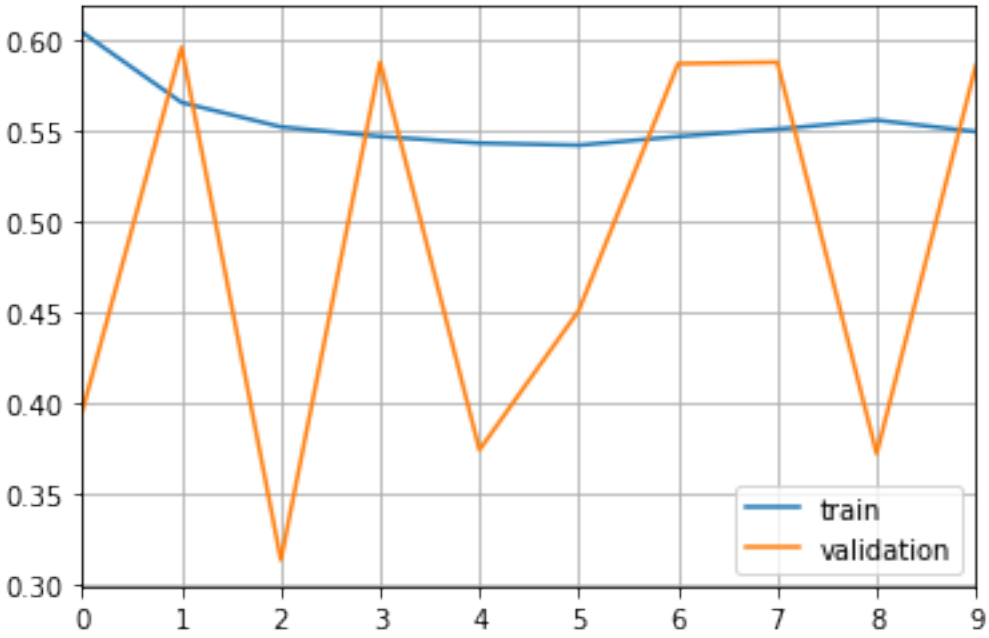
[17]:
```
train_history = train(model, training_generator, num_epochs)
```

```
Start training…

100%|
    | 37/37 [01:42<00:00,  2.76s/it]
100%|
     | 4/4 [00:10<00:00,  2.59s/it]

Epoch 1   loss:0.6036993192659842   eval_loss:0.3954412639141083
accuracy:0.7837837837837839   eval_accuracy: 0.85

100%|
    | 37/37 [01:38<00:00,  2.65s/it]
100%|
     | 4/4 [00:11<00:00,  2.80s/it]

Epoch 2   loss:0.5651485364179354   eval_loss:0.5955150127410889
accuracy:0.8108108108108109   eval_accuracy: 0.85

100%|
    | 37/37 [01:58<00:00,  3.21s/it]
100%|
     | 4/4 [00:13<00:00,  3.35s/it]

Epoch 3   loss:0.5517468508836385   eval_loss:0.3136637508869171
accuracy:0.8108108108108109   eval_accuracy: 0.85

100%|
    | 37/37 [02:07<00:00,  3.45s/it]
100%|
     | 4/4 [00:16<00:00,  4.22s/it]

Epoch 4   loss:0.5464425755513681   eval_loss:0.587364673614502
accuracy:0.810810810810811   eval_accuracy: 0.85

100%|
    | 37/37 [02:19<00:00,  3.76s/it]
100%|
     | 4/4 [00:13<00:00,  3.49s/it]

Epoch 5   loss:0.5428367458485268   eval_loss:0.37417370080947876
accuracy:0.810810810810811   eval_accuracy: 0.85

100%|
    | 37/37 [02:19<00:00,  3.78s/it]
100%|
     | 4/4 [00:15<00:00,  3.89s/it]
```

```
Epoch 6   loss:0.5416768104643435   eval_loss:0.45081761479377747
accuracy:0.8108108108108109   eval_accuracy: 0.85

100%|
    | 37/37 [02:18<00:00,  3.76s/it]
100%|
     | 4/4 [00:15<00:00,  3.92s/it]

Epoch 7   loss:0.5464279063650079   eval_loss:0.5864800214767456
accuracy:0.8108108108108109   eval_accuracy: 0.85

100%|
    | 37/37 [02:16<00:00,  3.69s/it]
100%|
     | 4/4 [00:15<00:00,  3.94s/it]

Epoch 8   loss:0.5505075970211545   eval_loss:0.5872970819473267
accuracy:0.8108108108108109   eval_accuracy: 0.85

100%|
    | 37/37 [02:25<00:00,  3.92s/it]
100%|
     | 4/4 [00:13<00:00,  3.46s/it]

Epoch 9   loss:0.5554544079948116   eval_loss:0.3721107244491577
accuracy:0.8108108108108109   eval_accuracy: 0.85

100%|
    | 37/37 [02:22<00:00,  3.86s/it]
100%|
     | 4/4 [00:13<00:00,  3.39s/it]

Epoch 10   loss:0.5490918199758272   eval_loss:0.5857167840003967
accuracy:0.8108108108108109   eval_accuracy: 0.85
Model saved at Trained _model
Done!
```

1.2 Evaluation

[18]:
```
kk = pd.DataFrame(train_history).plot(kind='line',grid=True)
```

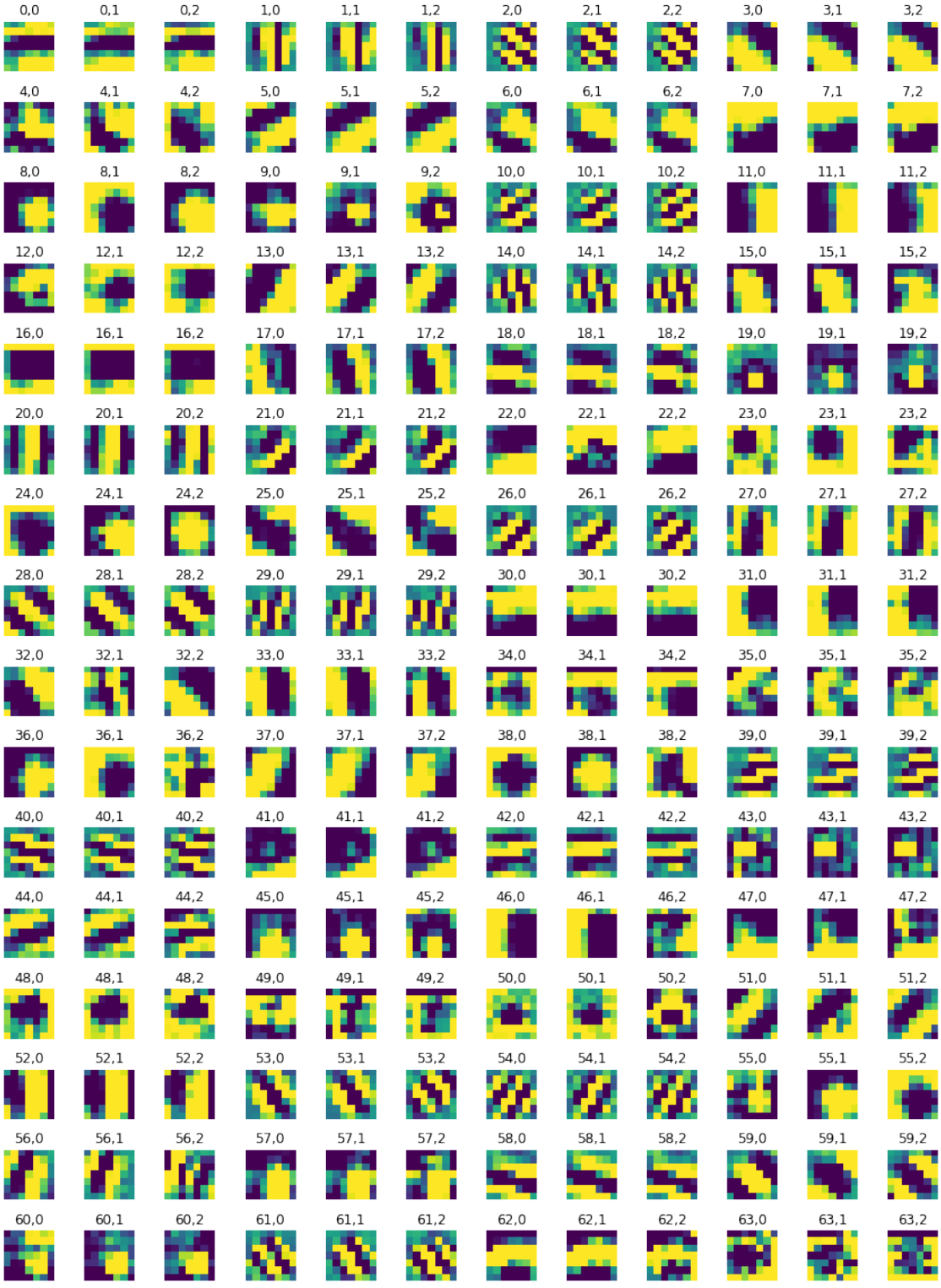


1.3 See Confusion matrix

[19]:
```
see_pred = pd.DataFrame(predict(model, validation_generator))
print("Confusion matrix shows the TP, FP,TN, FN rates of the model")
print(confusion_matrix(see_pred['real'],see_pred['pred']))
```

```
100%|
     | 4/4 [00:09<00:00,  2.33s/it]

Confusion matrix shows the TP, FP,TN, FN rates of the model
[[ 0  3]
 [ 0 17]]
```

[20]:
```
vis=visualizer()
vis.plot_filter(layer=model.resnet.conv1,single_channel = True)
```


0,0 0,1 0,2 1,0 1,1 1,2 2,0 2,1 2,2 3,0 3,1 3,2
4,0 4,1 4,2 5,0 5,1 5,2 6,0 6,1 6,2 7,0 7,1 7,2
8,0 8,1 8,2 9,0 9,1 9,2 10,0 10,1 10,2 11,0 11,1 11,2
12,0 12,1 12,2 13,0 13,1 13,2 14,0 14,1 14,2 15,0 15,1 15,2
16,0 16,1 16,2 17,0 17,1 17,2 18,0 18,1 18,2 19,0 19,1 19,2
20,0 20,1 20,2 21,0 21,1 21,2 22,0 22,1 22,2 23,0 23,1 23,2
24,0 24,1 24,2 25,0 25,1 25,2 26,0 26,1 26,2 27,0 27,1 27,2
28,0 28,1 28,2 29,0 29,1 29,2 30,0 30,1 30,2 31,0 31,1 31,2
32,0 32,1 32,2 33,0 33,1 33,2 34,0 34,1 34,2 35,0 35,1 35,2
36,0 36,1 36,2 37,0 37,1 37,2 38,0 38,1 38,2 39,0 39,1 39,2
40,0 40,1 40,2 41,0 41,1 41,2 42,0 42,1 42,2 43,0 43,1 43,2
44,0 44,1 44,2 45,0 45,1 45,2 46,0 46,1 46,2 47,0 47,1 47,2
48,0 48,1 48,2 49,0 49,1 49,2 50,0 50,1 50,2 51,0 51,1 51,2
52,0 52,1 52,2 53,0 53,1 53,2 54,0 54,1 54,2 55,0 55,1 55,2
56,0 56,1 56,2 57,0 57,1 57,2 58,0 58,1 58,2 59,0 59,1 59,2
60,0 60,1 60,2 61,0 61,1 61,2 62,0 62,1 62,2 63,0 63,1 63,2

2 ———The END———

[21]:
```
summary(model,(3,resnet_resize[0],resnet_resize[1]))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 187, 125]           9,408
       BatchNorm2d-2         [-1, 64, 187, 125]             128
              ReLU-3         [-1, 64, 187, 125]               0
         MaxPool2d-4           [-1, 64, 94, 63]               0
            Conv2d-5           [-1, 64, 94, 63]          36,864
       BatchNorm2d-6           [-1, 64, 94, 63]             128
              ReLU-7           [-1, 64, 94, 63]               0
            Conv2d-8           [-1, 64, 94, 63]          36,864
       BatchNorm2d-9           [-1, 64, 94, 63]             128
             ReLU-10           [-1, 64, 94, 63]               0
       BasicBlock-11           [-1, 64, 94, 63]               0
           Conv2d-12           [-1, 64, 94, 63]          36,864
      BatchNorm2d-13           [-1, 64, 94, 63]             128
             ReLU-14           [-1, 64, 94, 63]               0
           Conv2d-15           [-1, 64, 94, 63]          36,864
      BatchNorm2d-16           [-1, 64, 94, 63]             128
             ReLU-17           [-1, 64, 94, 63]               0
       BasicBlock-18           [-1, 64, 94, 63]               0
           Conv2d-19           [-1, 64, 94, 63]          36,864
      BatchNorm2d-20           [-1, 64, 94, 63]             128
             ReLU-21           [-1, 64, 94, 63]               0
           Conv2d-22           [-1, 64, 94, 63]          36,864
      BatchNorm2d-23           [-1, 64, 94, 63]             128
             ReLU-24           [-1, 64, 94, 63]               0
       BasicBlock-25           [-1, 64, 94, 63]               0
           Conv2d-26          [-1, 128, 47, 32]          73,728
      BatchNorm2d-27          [-1, 128, 47, 32]             256
             ReLU-28          [-1, 128, 47, 32]               0
           Conv2d-29          [-1, 128, 47, 32]         147,456
      BatchNorm2d-30          [-1, 128, 47, 32]             256
           Conv2d-31          [-1, 128, 47, 32]           8,192
      BatchNorm2d-32          [-1, 128, 47, 32]             256
             ReLU-33          [-1, 128, 47, 32]               0
       BasicBlock-34          [-1, 128, 47, 32]               0
           Conv2d-35          [-1, 128, 47, 32]         147,456
      BatchNorm2d-36          [-1, 128, 47, 32]             256
             ReLU-37          [-1, 128, 47, 32]               0
           Conv2d-38          [-1, 128, 47, 32]         147,456
      BatchNorm2d-39          [-1, 128, 47, 32]             256
             ReLU-40          [-1, 128, 47, 32]               0
       BasicBlock-41          [-1, 128, 47, 32]               0
```

Conv2d-42	[-1, 128, 47, 32]	147,456
BatchNorm2d-43	[-1, 128, 47, 32]	256
ReLU-44	[-1, 128, 47, 32]	0
Conv2d-45	[-1, 128, 47, 32]	147,456
BatchNorm2d-46	[-1, 128, 47, 32]	256
ReLU-47	[-1, 128, 47, 32]	0
BasicBlock-48	[-1, 128, 47, 32]	0
Conv2d-49	[-1, 128, 47, 32]	147,456
BatchNorm2d-50	[-1, 128, 47, 32]	256
ReLU-51	[-1, 128, 47, 32]	0
Conv2d-52	[-1, 128, 47, 32]	147,456
BatchNorm2d-53	[-1, 128, 47, 32]	256
ReLU-54	[-1, 128, 47, 32]	0
BasicBlock-55	[-1, 128, 47, 32]	0
Conv2d-56	[-1, 256, 24, 16]	294,912
BatchNorm2d-57	[-1, 256, 24, 16]	512
ReLU-58	[-1, 256, 24, 16]	0
Conv2d-59	[-1, 256, 24, 16]	589,824
BatchNorm2d-60	[-1, 256, 24, 16]	512
Conv2d-61	[-1, 256, 24, 16]	32,768
BatchNorm2d-62	[-1, 256, 24, 16]	512
ReLU-63	[-1, 256, 24, 16]	0
BasicBlock-64	[-1, 256, 24, 16]	0
Conv2d-65	[-1, 256, 24, 16]	589,824
BatchNorm2d-66	[-1, 256, 24, 16]	512
ReLU-67	[-1, 256, 24, 16]	0
Conv2d-68	[-1, 256, 24, 16]	589,824
BatchNorm2d-69	[-1, 256, 24, 16]	512
ReLU-70	[-1, 256, 24, 16]	0
BasicBlock-71	[-1, 256, 24, 16]	0
Conv2d-72	[-1, 256, 24, 16]	589,824
BatchNorm2d-73	[-1, 256, 24, 16]	512
ReLU-74	[-1, 256, 24, 16]	0
Conv2d-75	[-1, 256, 24, 16]	589,824
BatchNorm2d-76	[-1, 256, 24, 16]	512
ReLU-77	[-1, 256, 24, 16]	0
BasicBlock-78	[-1, 256, 24, 16]	0
Conv2d-79	[-1, 256, 24, 16]	589,824
BatchNorm2d-80	[-1, 256, 24, 16]	512
ReLU-81	[-1, 256, 24, 16]	0
Conv2d-82	[-1, 256, 24, 16]	589,824
BatchNorm2d-83	[-1, 256, 24, 16]	512
ReLU-84	[-1, 256, 24, 16]	0
BasicBlock-85	[-1, 256, 24, 16]	0
Conv2d-86	[-1, 256, 24, 16]	589,824
BatchNorm2d-87	[-1, 256, 24, 16]	512
ReLU-88	[-1, 256, 24, 16]	0
Conv2d-89	[-1, 256, 24, 16]	589,824

```
      BatchNorm2d-90          [-1, 256, 24, 16]             512
             ReLU-91          [-1, 256, 24, 16]               0
       BasicBlock-92          [-1, 256, 24, 16]               0
           Conv2d-93          [-1, 256, 24, 16]         589,824
      BatchNorm2d-94          [-1, 256, 24, 16]             512
             ReLU-95          [-1, 256, 24, 16]               0
           Conv2d-96          [-1, 256, 24, 16]         589,824
      BatchNorm2d-97          [-1, 256, 24, 16]             512
             ReLU-98          [-1, 256, 24, 16]               0
       BasicBlock-99          [-1, 256, 24, 16]               0
          Conv2d-100           [-1, 512, 12, 8]       1,179,648
     BatchNorm2d-101           [-1, 512, 12, 8]           1,024
            ReLU-102           [-1, 512, 12, 8]               0
          Conv2d-103           [-1, 512, 12, 8]       2,359,296
     BatchNorm2d-104           [-1, 512, 12, 8]           1,024
          Conv2d-105           [-1, 512, 12, 8]         131,072
     BatchNorm2d-106           [-1, 512, 12, 8]           1,024
            ReLU-107           [-1, 512, 12, 8]               0
      BasicBlock-108           [-1, 512, 12, 8]               0
          Conv2d-109           [-1, 512, 12, 8]       2,359,296
     BatchNorm2d-110           [-1, 512, 12, 8]           1,024
            ReLU-111           [-1, 512, 12, 8]               0
          Conv2d-112           [-1, 512, 12, 8]       2,359,296
     BatchNorm2d-113           [-1, 512, 12, 8]           1,024
            ReLU-114           [-1, 512, 12, 8]               0
      BasicBlock-115           [-1, 512, 12, 8]               0
          Conv2d-116           [-1, 512, 12, 8]       2,359,296
     BatchNorm2d-117           [-1, 512, 12, 8]           1,024
            ReLU-118           [-1, 512, 12, 8]               0
          Conv2d-119           [-1, 512, 12, 8]       2,359,296
     BatchNorm2d-120           [-1, 512, 12, 8]           1,024
            ReLU-121           [-1, 512, 12, 8]               0
      BasicBlock-122           [-1, 512, 12, 8]               0
AdaptiveAvgPool2d-123             [-1, 512, 1, 1]               0
          Linear-124                   [-1, 8]           4,104
          ResNet-125                   [-1, 8]               0
          Linear-126                   [-1, 2]              18
================================================================
Total params: 21,288,794
Trainable params: 4,122
Non-trainable params: 21,284,672
----------------------------------------------------------------
Input size (MB): 1.07
Forward/backward pass size (MB): 183.55
Params size (MB): 81.21
Estimated Total Size (MB): 265.83
----------------------------------------------------------------
```