test case

September 24, 2022

[1]:

```
import numpy as np
import cv2

# Open the video
cap = cv2.VideoCapture('C://Users/Nikolay/Downloads/video/Camera␣
 ↪3_20220526_003249(2).mp4')

# Initialize frame counter
cnt = 1

w_frame, h_frame = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)), int(cap.get(cv2.
 ↪CAP_PROP_FRAME_HEIGHT))
fps, frames = cap.get(cv2.CAP_PROP_FPS), cap.get(cv2.CAP_PROP_FRAME_COUNT)

# define croping values
x,y,h,w = 115,210,235,235

# output
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('result.avi', fourcc, fps, (w, h))


while(cap.isOpened()):
    ret, frame = cap.read()

    cnt += 1 # Counting frames

    # Avoid problems when video finish
    if ret==True:
        # Croping the frame
        crop_frame = frame[y:y+h, x:x+w]

        # Percentage
        xx = cnt *100/frames
        print(int(xx),'%')

        #Saving from the desired frames
        if cnt % 15 == 0:
            out.write(crop_frame)

        # see the video in real time
        cv2.imshow('frame',frame)
        cv2.imshow('croped',crop_frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break


cap.release()
out.release()
cv2.destroyAllWindows()
```

```
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
0 %
```

```
99 %
99 %
99 %
99 %
99 %
99 %
99 %
99 %
99 %
99 %
99 %
99 %
99 %
99 %
99 %
100 %
```

[7]:

```
#size for reshape
down_width = 116

down_height = 116

down_points = (down_width, down_height)
```

[8]:

```
# reshape all images to 116*116
vidcap = cv2.VideoCapture('C://Users/Nikolay/jupyter_notebooks/result.avi')
success,image = vidcap.read()
count = 0
while success:
    image = cv2.resize(image, down_points, interpolation= cv2.INTER_LINEAR)
    cv2.imwrite("frame%d.jpg" % count, image)     # save frame as JPEG file    ␣
 ↪
    success,image = vidcap.read()
    print('Read a new frame: ', success)
    count += 1
```

```
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  False
```

[16]:

```
# checking the image
image = cv2.imread('C://Users/Nikolay/Desktop/test/1/frame9484.jpg')
cv2.imshow('Resized Down by defining height and width', image)
cv2.waitKey()
cv2.destroyAllWindows()
```

[3]:
```
import torch
import random
#import numpy as np
import os

torch.backends.cudnn.deterministic = True
from torchvision.datasets import ImageFolder
from torchvision.transforms import ToTensor
from torchvision import datasets

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torchvision
from skimage import io
import pandas as pd



# fix random seed
random.seed(0)
#np.random.seed(0)
torch.manual_seed(0)
torch.cuda.manual_seed(0)

# using cuda
device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
C:\Users\Nikolay\AppData\Local\Programs\Python\Python310\lib\site-
packages\torch\random.py:42: UserWarning: Failed to initialize NumPy: module
compiled against API version 0x10 but this version of numpy is 0xf (Triggered
internally at  ..\torch\csrc\utils\tensor_numpy.cpp:68.)
  return default_generator.manual_seed(seed)
```

0.1 Making labels

[31]:
```
X = os.listdir('C://Users/Nikolay/Desktop/test/no')
y = os.listdir('C://Users/Nikolay/Desktop/test/target')
```

[32]:
```
print(X[0], y[0])
```

```
frame0.jpg frame10214.jpg
```

[33]:
```
len(X), len(y)
```

```
[33]: (29105, 8578)
```

```
[54]: df_sit = pd.DataFrame({'Frame': y})
      df_Notsit = pd.DataFrame({'Frame': X})
```

```
[55]: df_sit['class'] = np.ones(len(y))
      df_Notsit['class'] = np.zeros(len(X))
```

```
[56]: df_Notsit.head()
```

```
[56]:            Frame  class
      0     frame0.jpg    0.0
      1     frame1.jpg    0.0
      2    frame10.jpg    0.0
      3   frame100.jpg    0.0
      4  frame1000.jpg    0.0
```

```
[59]: labels = pd.concat([df_sit, df_Notsit], axis = 0)
      labels.head()
```

```
[59]:             Frame  class
      0  frame10214.jpg    1.0
      1  frame10215.jpg    1.0
      2  frame10216.jpg    1.0
      3  frame10217.jpg    1.0
      4  frame10218.jpg    1.0
```

```
[136]: labels.to_excel('C://Users/Nikolay/jupyter_notebooks/labels.xlsx',header = True,
                       index=['Frame', 'class'],
                       columns=['Frame', 'class'])
```

```
[10]: pd.read_excel('C://Users/Nikolay/Desktop/test/images/labels.xlsx', index_col=0)
```

```
[10]:                 Frame  class
      0      frame10214.jpg      1
      1      frame10215.jpg      1
      2      frame10216.jpg      1
      3      frame10217.jpg      1
      4      frame10218.jpg      1
      …                   …    …
      29100   frame9995.jpg      0
      29101   frame9996.jpg      0
      29102   frame9997.jpg      0
      29103   frame9998.jpg      0
      29104   frame9999.jpg      0

      [37683 rows x 2 columns]
```

0.2 making class for Dataset

[9]:

```
class SitNotSitDataset(Dataset):
    def __init__(self, xlsx_file, root_dir, transform = None):
        self.annotations = pd.read_excel(xlsx_file, index_col=0)
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self,index):
        img_path = os.path.join(self.root_dir, self.annotations.iloc[index, 0])
        image = io.imread(img_path)
        y_label = torch.tensor(int(self.annotations.iloc[index, 1]))

        if self.transform:
            image = self.transform(image)

        return(image, y_label)
```

[10]:

```
dataset = SitNotSitDataset(xlsx_file = 'C://Users/Nikolay/Desktop/test/images/
 ↪labels.xlsx',
                          root_dir = 'C://Users/Nikolay/Desktop/test/images',
                          transform = transforms.ToTensor())
```

[11]:

```
len(dataset)
```

[11]:

```
37683
```

0.3 train_test_split

[12]:

```
train_size = int(0.7 * len(dataset))
test_size = len(dataset) - train_size
train_set, test_set = torch.utils.data.random_split(dataset, [train_size,␣
 ↪test_size])
```

[13]:

```
batch_size = 16
```

[14]:

```
train_loader = DataLoader(dataset = train_set, batch_size = batch_size, shuffle␣
 ↪= True)
test_loader = DataLoader(dataset = test_set, batch_size = batch_size, shuffle =␣
 ↪True)
```

0.4 using googlenet

[15]:

```
model = torchvision.models.googlenet(pretrained = True)
```

```
C:\Users\Nikolay\AppData\Local\Programs\Python\Python310\lib\site-
packages\torchvision\models\_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and will be removed in 0.15, please use
'weights' instead.
  warnings.warn(
C:\Users\Nikolay\AppData\Local\Programs\Python\Python310\lib\site-
packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and will be
removed in 0.15. The current behavior is equivalent to passing
`weights=GoogLeNet_Weights.IMAGENET1K_V1`. You can also use
`weights=GoogLeNet_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[18]:

```
model.to(device)
```

[18]:

```
GoogLeNet(
  (conv1): BasicConv2d(
    (conv): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (maxpool1): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=True)
  (conv2): BasicConv2d(
    (conv): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (conv3): BasicConv2d(
    (conv): Conv2d(64, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (maxpool2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=True)
  (inception3a): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(192, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(96, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(192, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception3b): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(128, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(256, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (maxpool3): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=True)
  (inception4a): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(480, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(480, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(96, 208, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
```

```
        (bn): BatchNorm2d(208, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(480, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(16, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(480, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4b): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(512, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 112, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(112, 224, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(224, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(24, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4c): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(24, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4d): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(512, 112, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 144, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(144, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(144, 288, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(288, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4e): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(528, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(528, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(160, 320, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(528, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
```

```
      (1): BasicConv2d(
        (conv): Conv2d(528, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (maxpool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=True)
  (inception5a): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(832, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(160, 320, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception5b): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(832, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(48, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (aux1): None
  (aux2): None
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (dropout): Dropout(p=0.2, inplace=False)
  (fc): Linear(in_features=1024, out_features=1000, bias=True)
)
```

0.5 Loss and optimizer

[19]:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr = 1e-3)
```

[20]:

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

[21]:

```
num_epochs = 2
```

0.6 training

[25]:

```
for epoch in range(num_epochs):
    losses = []

    for batch_idx, (data, targets) in enumerate(train_loader):
        data = data.to(device = device)
        targets = targets.to(device = device)

        #forward
        scores = model(data)
        loss = criterion(scores,targets)

        losses.append(loss.item())

        #backward
        optimizer.zero_grad()
        loss.backward()

        #adam step
        optimizer.step()
    print(f"Cost at epoch {epoch} is {sum(losses)/len(losses)}")
```

```
Cost at epoch 0 is 0.026988541723408633
Cost at epoch 1 is 0.007583270934344961
```

0.7 check accuracy

[30]:

```
def check_accuracy(loader, model):
    num_correct = 0
    num_samples = 0
    model.eval()
    with torch.no_grad():
        for x,y in loader:
            x = x.to(device = device)
            y = y.to(device = device)

            scores = model(x)
            _, predictions = scores.max(1)
            num_correct += (predictions == y).sum()
            num_samples += predictions.size(0)

        print(f"Got {num_correct} / {num_samples} with accuracy␣
 ↪{float(num_correct)/float(num_samples)*100}")
    model.train()
```

[31]:

```
print("Checking accuracy on Training Set")
check_accuracy(train_loader, model)
```

```
Checking accuracy on Training Set
Got 26369 / 26378 with accuracy 99.96588065812419
```

[32]:

```
print("Checking accuracy on Testing Set")
check_accuracy(test_loader, model)
```

```
Checking accuracy on Testing Set
Got 11302 / 11305 with accuracy 99.97346306943831
```

[ ]: