

Read Me for App

version 1.0.0

Contents

Application

Motivation for README

What Other Software Is needed to Run This Software

how install Python

How to Install TensorFlow and Pytorch

loading image files into networks

getting figures from the network

what to expect from the program

code explation

Check list

- ✓ UI updates when button is clicked
 - ✓ able to load new images into different trained models
 - ✓ can get figures back from the program
 - ✓ use loaded models to segment images
 - ✓ can only segment one image at a time
 - ✗ images classes are displayed
 - ✓ load images without labels through a trained model
 - ✗ can load more than one image into trained model
 - ✓ self conntained program for segmenting medical images
 - ✗ EXE file to run the code
-

Application

will allow any user to segment images of necks, brain cancer tumours or breast cancer tumours from scans. The rest of the document will tell you how to:

load an image into the trained models

get back the saved image from the model

load images of brains, neck and breast cancer scan

Motivation

this readme will outline everything you need to understand how to run the python program for segmenting a brain to outline a tumour, a neck scan to outline different muscles, a breast scan for outlining any tumours

Software

python 3.7

TensorFlow version 1.15.0 with keras

pytorch 1.4.0

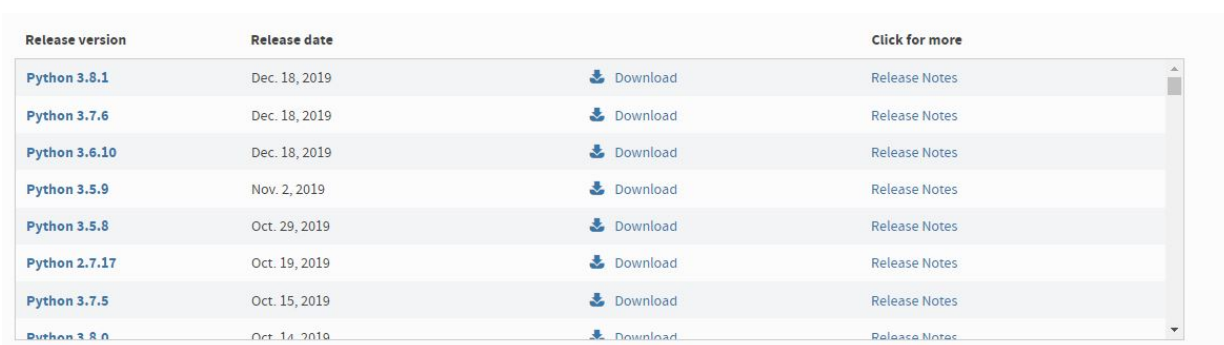
python TKinter

matplotlib

numpy

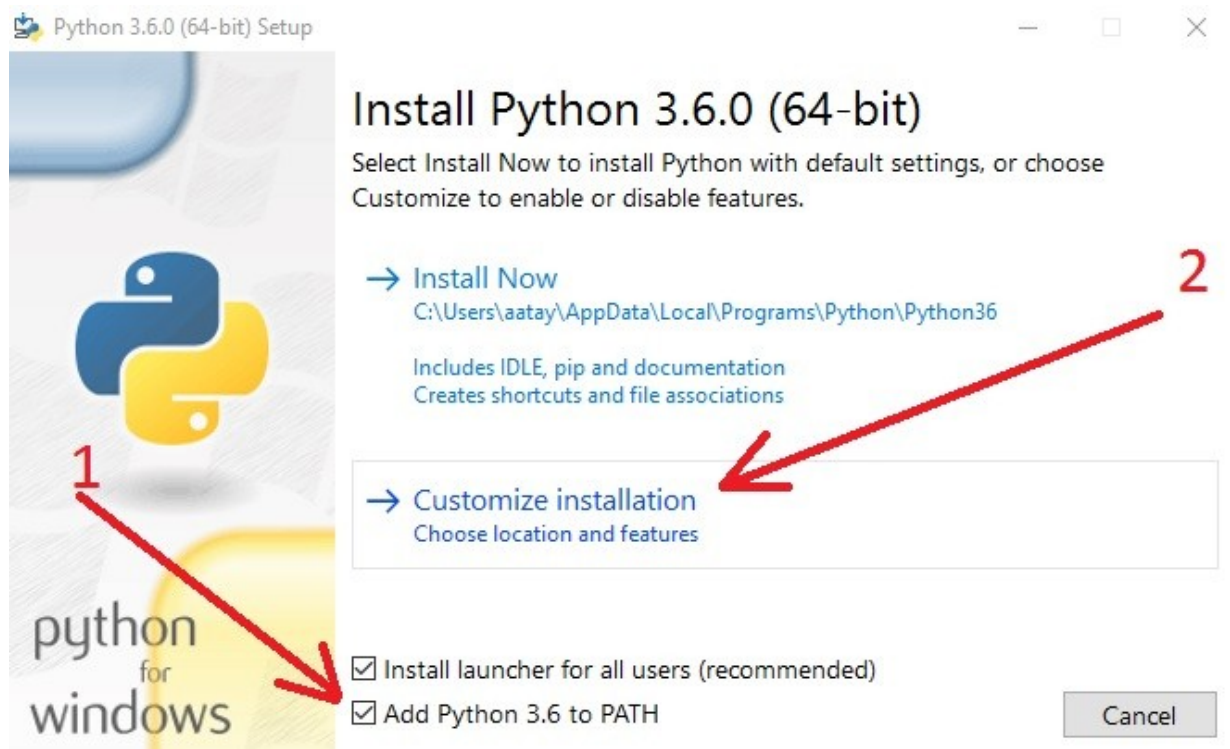
Python

you will need to install python 3.7 as seen in the image below. Click the "3.7.6" download and use the wizard to set up Python



Release version	Release date		Click for more
Python 3.8.1	Dec. 18, 2019	Download	Release Notes
Python 3.7.6	Dec. 18, 2019	Download	Release Notes
Python 3.6.10	Dec. 18, 2019	Download	Release Notes
Python 3.5.9	Nov. 2, 2019	Download	Release Notes
Python 3.5.8	Oct. 29, 2019	Download	Release Notes
Python 2.7.17	Oct. 19, 2019	Download	Release Notes
Python 3.7.5	Oct. 15, 2019	Download	Release Notes
Python 3.8.0	Oct. 14, 2019	Download	Release Notes

make sure you do a custom install of python and add it to path.



once you have installed python make sure to test it and by going into CMD and typing "python" you should see the python interpreter open in cmd like so:

```
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

TensorFlow&Pytorch

How To Install Pytorch Using Pip:

```
pip install torch==1.4.0 torchvision==0.5.0 -f
https://download.pytorch.org/whl/torch_stable.html
```

using conda

```
conda install pytorch torchvision cudatoolkit=10.1 -c pytorch
```

how to install tensorflow using pip

```
pip install tensorflow-gpu==1.15
```

using conda

```
conda tensorflow==1.15
```

How Install Matplotlib

using pip

```
pip install Matplotlib
```

using conda

```
conda install -c conda-forge matplotlib
```

you should now be ready to run the program.

Running the program

you can run the program by using the command prompt on windows, shown below

```
python ProjectApp.py
```

make sure you have python installed and you have navigated to the correct folder where to [ProjectApp.py](#) is contained

there is no EXE file, I tried to make a EXE file of the program but I sadly could not get it working.

Files

loading images into the network is done by putting the correct scan in the correct folder inside the "newimg" folder, which has corresponding labeled folders. you must put the image you wish to load into folder named "1" and in the correctly labeled folder:

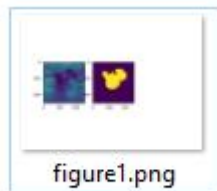


Name



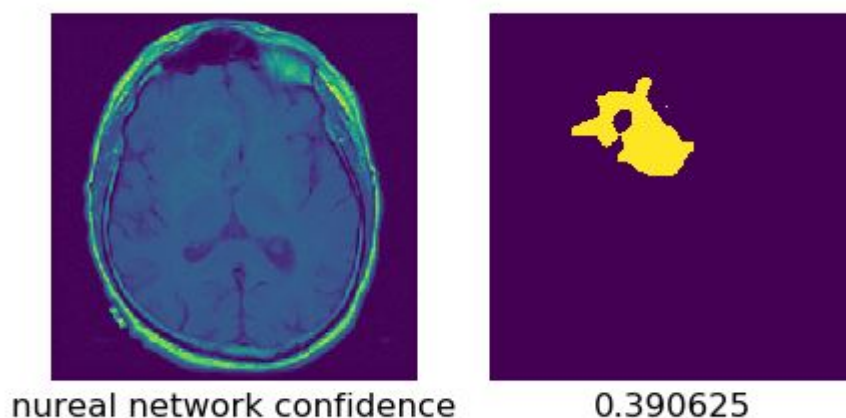
figuers

Figuers are saved in a folder so that a user can use them else where. each new image will be called "figure1" and the last image you loaded will be displayed in the app when you next open it



Each one of these figures will have the scan a user is wanting to segment and then the prediction from the trained model

expectations



When a user presses any of the buttons the program will take some time to pass the image through the trained model, the program will then restart thus updating the GUI, the program will then show the saved figure on the screen

as you can see from the image above the output of the network is displayed next to the input image. The Network has found a tumor in the image and outlined where it is (on the right) if the loaded scan did not contain a tumor the model would not output anything, the image on the right hand side would only contain the background (the image would be purple)

Code

```
model = torch.load("\\model\\101layerBrainModel.pth")
#put the model on ythe GPU
model.cuda()
#set the model to evaluation mode. sets dropout and batch normalization
layers to evaluation mode
model.eval()
```

the model is loaded from inside the app's dir, a new trained model is used for segmenting different images. Each model is trained on a different dataset and is tuned for only seggming a certain type of image

each of these models were 101 layer resnets with upsampling layers added to the end

preprocessing images for loading

```
img_path= '\\newimg\\brain\\'
#create a training image array to be loaded into the model
#rescale changes any images to 255 by 255
train_imgs =
tf.keras.preprocessing.image.ImageDataGenerator(rescale=1/255,
data_format='channels_first')

train_img_gen = train_imgs.flow_from_directory(
img_path,
class_mode=None,
batch_size=1,
target_size=(256,256))
```

using Keras to preprocess images as the keras data loader seems to work better than the inbuilt pytorch loader

loading images through the model

```
#convert images to tensor to be loaded into the network
images = torch.Tensor(next(train_img_gen))

#put images on the GPU
images = images.cuda()
#set the gradients for params to trye for layers that need it.
with torch.set_grad_enabled(True):

    #push the image through the network
    y_pred = model.forward(images)
```

it is recommended that you only load image at a time, as I have not fully implemented displaying more than one figure on the screen.

display predictions on screen and save to a local folder

```
#the argmax gets the The maximum value along a given axis.
#getting the argmax of the image this effectly the flatten the
image into the most prelevant colours
y_pred = y_pred.cpu().detach().numpy().argmax(1)
plt.subplot(1, 3, 1)
#the detach() method constructs a new view on a tensor which is
declared not to need gradients
plt.imshow(images.cpu().detach().numpy()[0, 0, :, :])
plt.subplot(1, 3, 2)
plt.imshow(y_pred[0, :, :])
plt.savefig('E://figs//figure1.png')
#show figure as block.
plt.show(block=True)
```

the program will convert the output to an image by getting the argmax of the tensor effectively flattening the tensor

The window has to reset this is done by restarting the program.

the gui will only ever display one image at a time when a new image has been passed through a trained model it will destroy the old image saved in the local folder so users must take the image as when they want it.

displaying images in the GUI

```
# Load saved figure into UI
file="E://figs//figure1.png"
print(file)
photo = tk.PhotoImage(file=file)
#make anew canvas on the root for the UI to display the in
canvas = tk.Canvas(root, width = 500, height = 500)
canvas.create_image(256,256, image=photo)
canvas.pack()
#.pack() command is used for displaying element to the GUI
```

UI buttons

```
#code for buttons,
BrstC = tk.Button(root, text="segment breast cancer img", fg="black",
command=loadmodelbreast)
BrstC.pack(side=tk.LEFT, padx=5, pady=10)
NECK = tk.Button(root, text="load model for neck", fg="black",
command=loadNeckModelImgs)
NECK.pack(side=tk.LEFT, padx=7, pady=10)
BrnC = tk.Button(root, text="load model for brain cancer", fg="black",
command=loadbrainCancerImgs)

BrnC.pack(side=tk.LEFT, padx=9, pady=10)
#call the mainloop method to display the program
tk.mainloop()
```

Code explanation for training a model

setting up the model and adding layers to end for upsampling

```
#create the model, pretrained fully connected resnet with 101 layers
model = models.segmentation.fcn_resnet101(True)
torch.cuda.empty_cache()

#adding more layers for upsampling and then adding Logsoftmax activation
function
#after experimenting with outputs and other activation functions this was
```

```

found to be the best
layers = []
#Loop through all models layers but the last layer
for i in list(model.children())[:-1]:
    #for all items in the array model.children
    for j in list(i):
        #append to the last layer if the type is the same as an empty
        string
        if type(j) == type(''):
            #append layers j to i
            layers.append(i[j])
        else:
            #else just append j
            layers.append(j)

layers = layers[:-1]
#these layers commented here can be used for other network shapes
"""
layers.append(torch.nn.Conv2d(1024, 256, kernel_size=(1, 1), stride=(1,
1)))
layers.append(torch.nn.Conv2d(256, 1, kernel_size=(1, 1), stride=(1, 1)))
"""
#up sample layers allow the image to be up sampled from 30 by 30, from
the output of the resnet up 256 by 256
#by using KNN up sampling
layers.append(torch.nn.UpsamplingNearest2d(scale_factor=2))
#match the new convsd layer we are adding to the previous layer by giving
it 512
layers.append(torch.nn.Conv2d(512, 512, kernel_size=(3, 3), padding=(1,
1)))

layers.append(torch.nn.UpsamplingNearest2d(scale_factor=2))

layers.append(torch.nn.Conv2d(512, 256, kernel_size=(3, 3), padding=(1,
1)))

layers.append(torch.nn.UpsamplingNearest2d(scale_factor=2))

layers.append(torch.nn.Conv2d(256, 256, kernel_size=(3, 3), padding=(1,
1)))
#16 output channels for the number of classes in each image
layers.append(torch.nn.Conv2d(256, 16, kernel_size=(3, 3), padding=(1, 1)))
#output activation function

layers.append(torch.nn.LogSoftmax(dim=1))

```

loading images and setting up the tensors for loading through the network

in this section I set up the tensors for training after loading them in from the dirs they are contained in

```

#set the model to training model
model.train()
#Loop through the training set
images = torch.Tensor(next(train_img_gen))

```



```

#loop through the Label set
labels = torch.Tensor(next(train_label_gen))

#make a long tensor out of the labels tensor
labels = labels.long()
#squeeze the 0th element of the tensor
labels = labels.squeeze(0)
#put all images on the GPU
images = images.cuda()
labels = labels.cuda()

```

pushing the images through the networking training

```

running_lossForTraining=0.0
#set the grad to true for the optimizer
with torch.set_grad_enabled(True):
    #push the images through the network
    y_pred = model.forward(images)
    #we need to set the gradients to zero before starting to do
backpropagation because
    #PyTorch accumulates the gradients on subsequent backward
passes
    optimizer.zero_grad()
    #compare the prediction to the labels
    loss = criterion(y_pred, labels)
    #backwards propagate through the network tweaking the weights of
the neurons to make the
    #network better
    loss.backward()
    optimizer.step()
    #loss.item gets the loss number not the loss tensor
    running_lossForTraining = loss.item()
    #write the training loss to a scalar and label graph "training
loss
    writer.add_scalar('training loss',
                      running_lossForTraining / 1000,
                      i* len(train_img_gen) + i
                      )

```

show an out and image along with its label to make sure the network is learning and for debugging

```

y_pred = y_pred.cpu().detach().numpy().argmax(axis=1)
plt.subplot(1, 3, 1)
plt.imshow(images.cpu().detach().numpy()[0, 0, :, :])
plt.subplot(1, 3, 2)
plt.imshow(y_pred[0, :, :])
plt.subplot(1, 3, 3)
plt.imshow(labels.cpu().detach().numpy()[0, :, :])
plt.show(block=True)

print("epoch: ", i, "train loss:" , loss.item())

```

testing phase

```
#set the model to testing mode
model.eval()

images = torch.Tensor(next(test_img_gen))
labels = torch.Tensor(next(test_label_gen))

labels = labels.long()
labels = labels.squeeze(0)

images = images.cuda()
labels = labels.cuda()

running_lossForTest = 0.0

y_pred = model.forward(images)
loss = criterion(y_pred, labels)

running_lossForTest = loss.item()
print("epoch: ", i, "test loss:" , loss.item())
writer.add_scalar('testing loss', running_lossForTest / 1000,
                  i* len(test_img_gen) + i)
```

saving the model

```
print("saving model...")
torch.save(model, "E:\\model\\101layerBrainmodel1.0.1.pth" )
print ("model saved.")
```