Final_Project_ADL

December 23, 2022

Mounting the drive and importing/downloading the necessary libraries

[1]:
```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)

import os
import tensorflow as tf

import cv2    ##used to save the patches in .jpeg format
from sklearn.model_selection import train_test_split

import IPython.display as display
import matplotlib.pyplot as plt
import random

from tensorflow.keras import datasets, layers, models
```

```
Mounted at /content/drive
```

[2]:
```
# Install the OpenSlide C library and Python bindings
!apt-get install openslide-tools
!pip install openslide-python
```

```
Reading package lists… Done
Building dependency tree
Reading state information… Done
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
The following additional packages will be installed:
  libopenslide0
Suggested packages:
  libtiff-tools
The following NEW packages will be installed:
  libopenslide0 openslide-tools
0 upgraded, 2 newly installed, 0 to remove and 20 not upgraded.
Need to get 92.5 kB of archives.
After this operation, 268 kB of additional disk space will be used.
```

```
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 libopenslide0 amd64
3.4.1+dfsg-2 [79.8 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/universe amd64 openslide-tools
amd64 3.4.1+dfsg-2 [12.7 kB]
Fetched 92.5 kB in 0s (191 kB/s)
Selecting previously unselected package libopenslide0.
(Reading database … 124016 files and directories currently installed.)
Preparing to unpack …/libopenslide0_3.4.1+dfsg-2_amd64.deb …
Unpacking libopenslide0 (3.4.1+dfsg-2) …
Selecting previously unselected package openslide-tools.
Preparing to unpack …/openslide-tools_3.4.1+dfsg-2_amd64.deb …
Unpacking openslide-tools (3.4.1+dfsg-2) …
Setting up libopenslide0 (3.4.1+dfsg-2) …
Setting up openslide-tools (3.4.1+dfsg-2) …
Processing triggers for libc-bin (2.27-3ubuntu1.6) …
Processing triggers for man-db (2.8.3-2ubuntu0.1) …
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting openslide-python
  Downloading openslide-python-1.2.0.tar.gz (338 kB)
     |                  | 338 kB 15.3 MB/s
Requirement already satisfied: Pillow in /usr/local/lib/python3.8/dist-
packages (from openslide-python) (7.1.2)
Building wheels for collected packages: openslide-python
  Building wheel for openslide-python (setup.py) … done
  Created wheel for openslide-python:
filename=openslide_python-1.2.0-cp38-cp38-linux_x86_64.whl size=27716
sha256=c17f09f5d43990110a3aeefc5c98797839728d283cddffed78c0fb698148c53a
  Stored in directory: /root/.cache/pip/wheels/ae/74/4e/f8654d229eb249d1345e3df0
191030ad77e7a6a2114df7cd28
Successfully built openslide-python
Installing collected packages: openslide-python
Successfully installed openslide-python-1.2.0
```

[3]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from openslide import open_slide, __library_version__ as openslide_version
import os
from PIL import Image
from skimage.color import rgb2gray
```

Using the address of the shortcut (created in my drive) for the folder containing the slides and the masks and selecting a few of them

[4]:
```
data_dir = "/content/drive/MyDrive/slides/"
files = os.listdir(data_dir)
```

[5]:
```
## adding all the tumor slide names to a list
all_image_names = []
for x in files:
  if (".xml" not in x) and ("mask" not in x):
    all_image_names.append(x)
```

[6]:
```
## adding the corresponding tumor mask file names to a list
all_image_masks = []
for x in files:
  if "mask" in x:
    all_image_masks.append(x)
```

[7]:
```
## we remove tumor_038.tif if present as it does not have it's corresponding␣
 ↪mask
remove = "tumor_038.tif"
for n,x in enumerate(all_image_names):
  if x == remove:
    all_image_names.pop(n)

## creating two new lists containing the image paths to the above tumors and␣
 ↪their correponding masks
image_paths = []
image_paths_mask = []
for x in range(len(all_image_names)):
  image_paths.append(data_dir + all_image_names[x])
  image_paths_mask.append(data_dir + all_image_masks[x])
```

Uploading all the slides and their correspoding masks from the folder to a GCP bucket for easier access and use

[ ]:
```
## configuring the project and user authentication to use GCP
from google.colab import auth
auth.authenticate_user()
project_id = 'computer-systems-341920'
!gcloud config set project {project_id}
!gsutil ls
```

[ ]:
```
## uploading the slides and their correspoding masks to a specified bucket
bucket_name = 'hw3_adl'
for x in zip(image_paths,image_paths_mask):
  n1 = x[0]
  n2 = x[1]
  !gsutil -m cp $n1 gs://{bucket_name}/
  !gsutil -m cp $n2 gs://{bucket_name}/
```

Download few slides and tumor masks from the bucket (this is optional, we have directly accessed the slides folder in google drive)

[8]:
```
colab_root = "/content/"
def download_if_missing(url, target, extract=True):
  if os.path.exists(target):
    return target
  return tf.keras.utils.get_file(target, origin=url, extract=extract)
```

[ ]:
```
## downloading a few slides and their masks from the bucket
BUCKET_URL ="https://storage.googleapis.com/hw3_adl/"
files = 15
count = 0
for x in zip(all_image_names,all_image_masks):
  count = count + 1
  n1 = x[0]
  n2 = x[1]
  file_path = colab_root + n1
  file_path_mask = colab_root + n2
  download_if_missing(BUCKET_URL + n1,file_path)
  download_if_missing(BUCKET_URL + n2,file_path_mask)
  if count >= files:
    break
```

```
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_002_mask.tif
97697336/97697336 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_110.tif
1485042280/1485042280 [==============================] - 21s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_012_mask.tif
95879690/95879690 [==============================] - 4s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_005.tif
1467903916/1467903916 [==============================] - 14s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_002.tif
1680307170/1680307170 [==============================] - 25s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_016_mask.tif
98694816/98694816 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_019.tif
1536279242/1536279242 [==============================] - 21s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_084_mask.tif
25949822/25949822 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_012.tif
1626791654/1626791654 [==============================] - 18s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_096_mask.tif
43253974/43253974 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_016.tif
1524584022/1524584022 [==============================] - 14s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_110_mask.tif
32922630/32922630 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_023.tif
1687969860/1687969860 [==============================] - 14s 0us/step
```

```
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_031_mask.tif
99493514/99493514 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_031.tif
1487082070/1487082070 [==============================] - 19s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_081_mask.tif
41073180/41073180 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_035.tif
1605582132/1605582132 [==============================] - 17s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_059_mask.tif
97703912/97703912 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_057.tif
1459755832/1459755832 [==============================] - 18s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_078_mask.tif
52204056/52204056 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_059.tif
1480993338/1480993338 [==============================] - 13s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_101_mask.tif
46053642/46053642 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_064.tif
1645432562/1645432562 [==============================] - 17s 0us/step
Downloading data from https://storage.googleapis.com/hw3_adl/tumor_094_mask.tif
54282736/54282736 [==============================] - 1s 0us/step
```

Read the slides using Openslide

[9]:
```
## printing the various zoom levels and their respective downsampling factors
## along with the dimensions of the image at every zoom level
## also applying certain sanity checks
file_path = data_dir+"tumor_101.tif"
file_path_mask = data_dir + "tumor_101_mask.tif"

slide = open_slide(file_path)
print ("Read WSI from %s with width: %d, height: %d" % (file_path,
                                                        slide.
 ↪level_dimensions[0][0],
                                                        slide.
 ↪level_dimensions[0][1]))

tumor_mask = open_slide(file_path_mask)
print ("Read tumor mask from %s" % (file_path_mask))

print("Slide includes %d levels", len(slide.level_dimensions))
for i in range(len(tumor_mask.level_dimensions)):
  print("Level %d, dimensions: %s downsample factor %d" % (i,
                                                           slide.
 ↪level_dimensions[i],
                                                           slide.
 ↪level_downsamples[i]))
```

```
  assert tumor_mask.level_dimensions[i][0] == slide.level_dimensions[i][0]
  assert tumor_mask.level_dimensions[i][1] == slide.level_dimensions[i][1]

# Verify downsampling works as expected
width, height = slide.level_dimensions[7]
assert width * slide.level_downsamples[7] == slide.level_dimensions[0][0]
assert height * slide.level_downsamples[7] == slide.level_dimensions[0][1]
```

```
Read WSI from /content/drive/MyDrive/slides/tumor_101.tif with width: 139264,
height: 71680
Read tumor mask from /content/drive/MyDrive/slides/tumor_101_mask.tif
Slide includes %d levels 10
Level 0, dimensions: (139264, 71680) downsample factor 1
Level 1, dimensions: (69632, 35840) downsample factor 2
Level 2, dimensions: (34816, 17920) downsample factor 4
Level 3, dimensions: (17408, 8960) downsample factor 8
Level 4, dimensions: (8704, 4480) downsample factor 16
Level 5, dimensions: (4352, 2240) downsample factor 32
Level 6, dimensions: (2176, 1120) downsample factor 64
Level 7, dimensions: (1088, 560) downsample factor 128
Level 8, dimensions: (544, 280) downsample factor 256
```

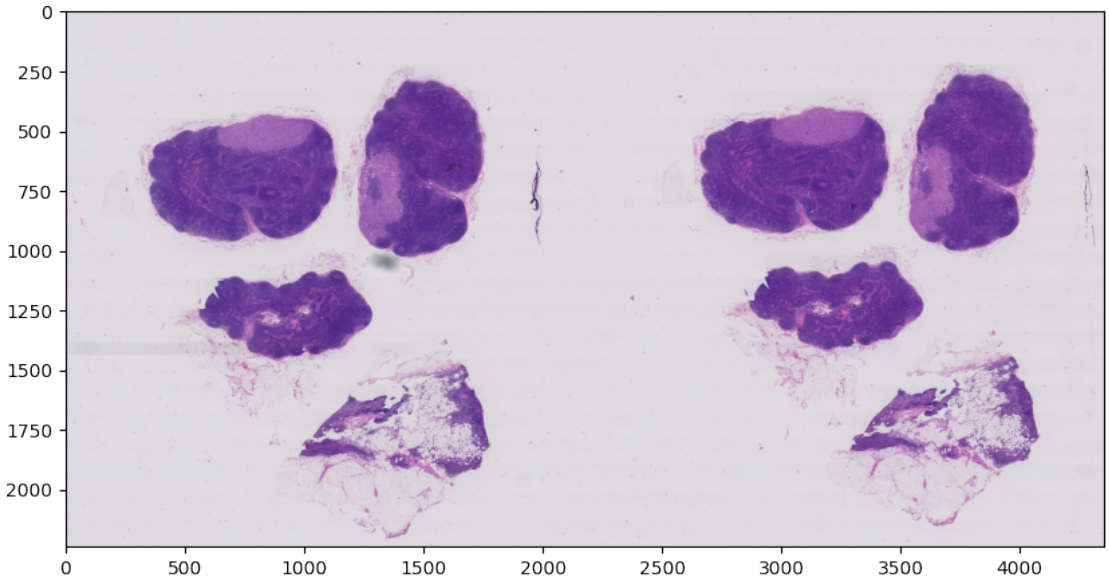
[ ]:
```
## applying certain checks on all the files in the slides folder
## making sure that the mask and the slide have same dimensions at same levels
## also making sure that the downsampling factor is correct
for x in all_image_names:
  file_path = data_dir + x
  file_path_mask = data_dir + x.split(".")[0] + "_mask." + x.split(".")[1]
  slide = open_slide(file_path)
  tumor_mask = open_slide(file_path_mask)
  for i in range(len(tumor_mask.level_dimensions)):
    try:
      tumor_mask.level_dimensions[i][0] == slide.level_dimensions[i][0]
      tumor_mask.level_dimensions[i][1] == slide.level_dimensions[i][1]
    except:
      print("File" , x , "is corrupted")
  # Verify downsampling works as expected
  width, height = slide.level_dimensions[7]
  assert width * slide.level_downsamples[7] == slide.level_dimensions[0][0]
  assert height * slide.level_downsamples[7] == slide.level_dimensions[0][1]
```

[11]:
```
# Note: x,y coords are with respect to level 0.
# Read a region from the slide
# Return a numpy RBG array
def read_slide(slide, x, y, level, width, height, as_float=False):
    ## top left on the image is the origin.
    ## (x,y) -- coordinates of the top left point on the patch
```

```
    ## height and width of the patch are also passed onto the fn.
    im = slide.read_region((x,y), level, (width, height))
    im = im.convert('RGB') # drop the alpha channel
    if as_float:
        im = np.asarray(im, dtype=np.float32)
    else:
        im = np.asarray(im)
    ## making sure the read_region function and conversion to RGB has taken␣
 ↪place
    assert im.shape == (height, width, 3)
    return im
```

[11]:
```
# reading the entire slide at level 5
slide_image = read_slide(slide,
                         x=0,
                         y=0,
                         level=5,
                         width=slide.level_dimensions[5][0],
                         height=slide.level_dimensions[5][1])

plt.figure(figsize=(10,10), dpi=100)
plt.imshow(slide_image)
```

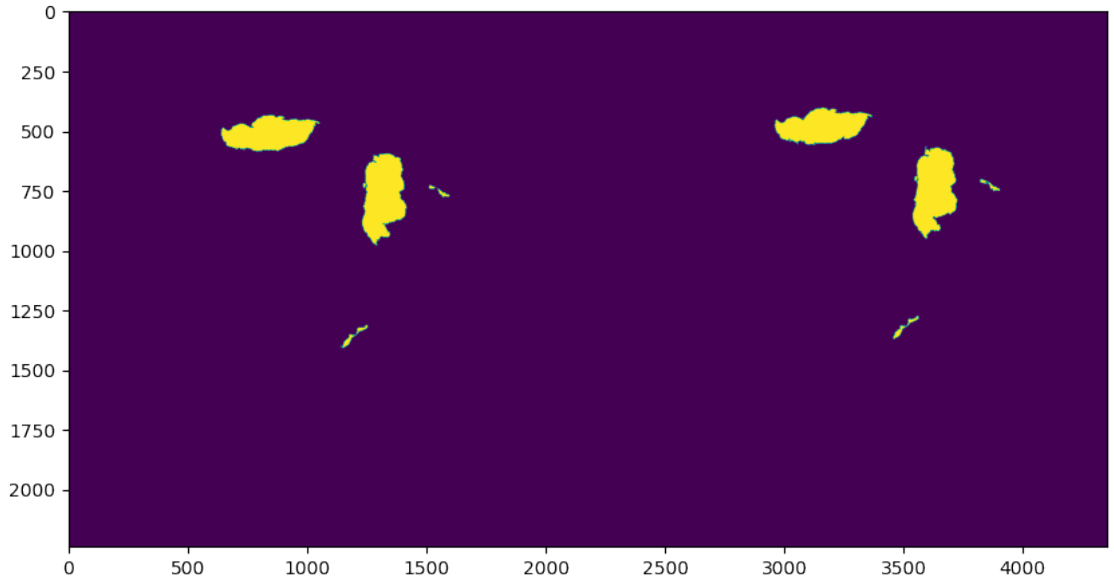
[11]:
```
<matplotlib.image.AxesImage at 0x7f6186d23580>
```



[12]:
```
#read the entire mask at the same zoom level (NOTE: the mask is binary, pixel␣
 ↪values are either 0 or 1)
```

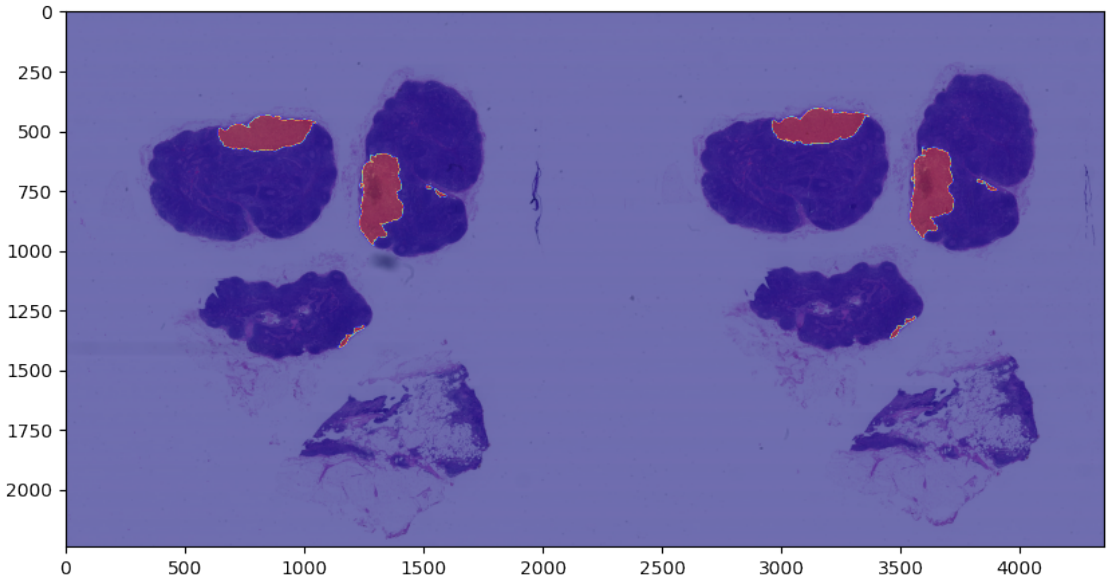
```
mask_image = read_slide(tumor_mask,
                        x=0,
                        y=0,
                        level=5,
                        width=slide.level_dimensions[5][0],
                        height=slide.level_dimensions[5][1])

# the above generates a mask with R,G,B channels.
# The mask info we need is in the first channel only.
mask_image = mask_image[:,:,0]

plt.figure(figsize=(10,10), dpi=100)
plt.imshow(mask_image)
```

[12]: <matplotlib.image.AxesImage at 0x7f6186816490>



[13]:
```
# Overlay them. The idea is that the mask shows the region of the slide that
# contain cancerous cells.
plt.figure(figsize=(10,10), dpi=100)
plt.imshow(slide_image)
plt.imshow(mask_image, cmap='jet', alpha=0.5) # Red regions contains cancer.
```

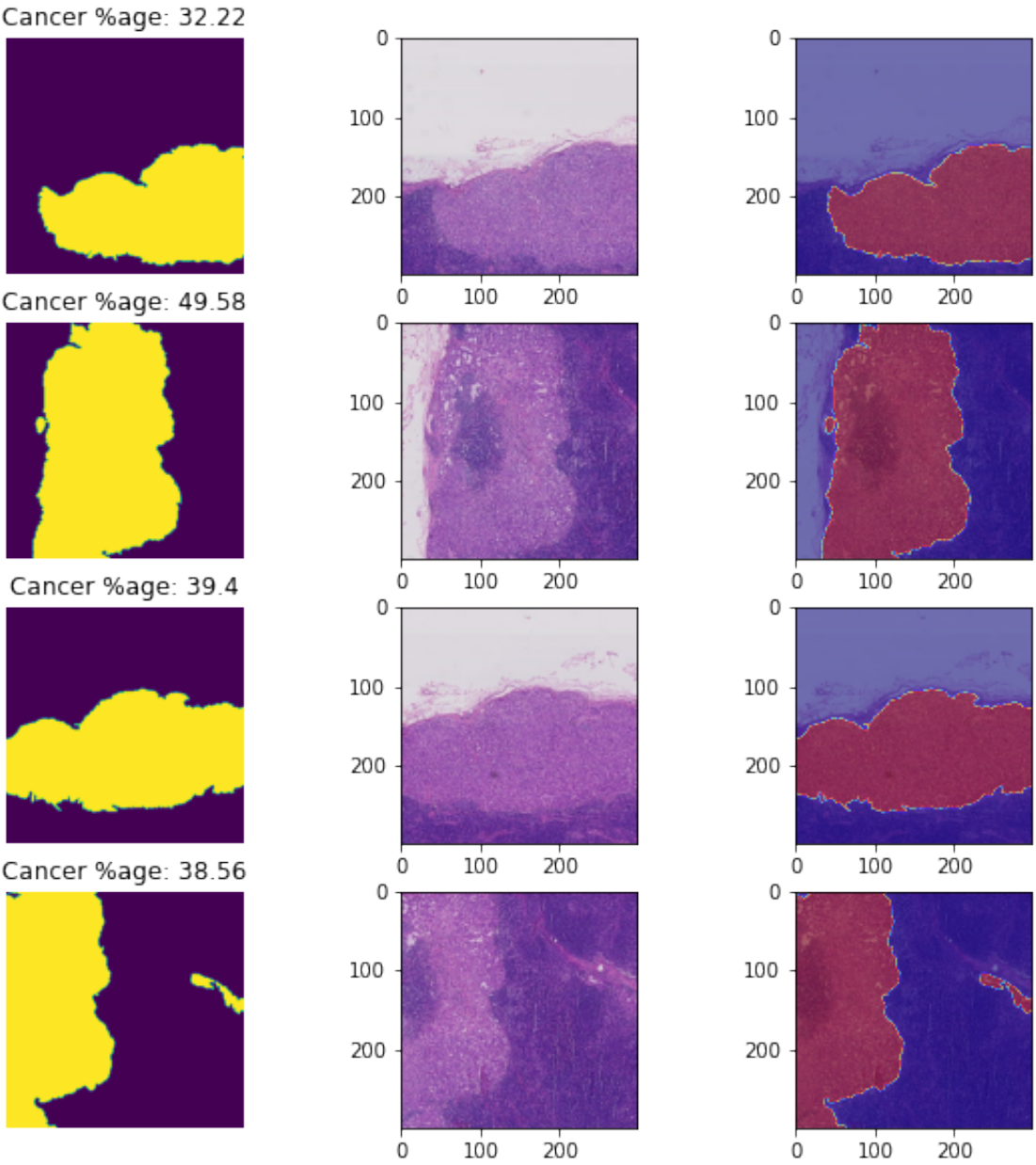
[13]: <matplotlib.image.AxesImage at 0x7f61857db3d0>



Performing the sliding window operation at Level 5 zoom and visualizing the cancerous patches (based on a 20% threshold) along with their masks and overlaps for one example to check if the sliding window operation is working properly

[14]:
```
## Rolling a non overlapping sliding window of size 299X299 (size taken from␣
 ↪the paper)
## Setting the threshold at 20%, the patch will be labelled as cancerous if␣
 ↪more than 20% of the pixels are cancerous
## Visualizing the cancerous patches, their corresponding masks and the overlap␣
 ↪of the two images to verify if the results look correct

slide_dim = 299
level = 5
dwn_smpls = int(slide.level_downsamples[level])
x_range = slide.level_dimensions[level][0]
y_range = slide.level_dimensions[level][1]
x_lt = int(x_range/slide_dim)
y_lt = int(y_range/slide_dim)
fig = plt.figure(figsize=(10, 10))
count = 0
for x in range(x_lt):
  for y in range(y_lt):
    # region = read_slide(slide, x=slide_dim*x, y=slide_dim *y, level=level,␣
 ↪width=slide_dim, height=slide_dim)
    region_mask = read_slide(tumor_mask, x=x*slide_dim*dwn_smpls,␣
 ↪y=y*slide_dim*dwn_smpls, level=level, width=slide_dim, height=slide_dim)[:,:
 ↪,0]
```

```
   cancer_tissue = round((sum(map(sum, region_mask))/slide_dim**2)*100,2)
   ## if there is more than 20% caneruous pixels in the patch, consider it␣
↪cancerous
   if cancer_tissue > 20:
     count = count + 1
     fig.add_subplot(4, 3, count)
     plt.imshow(region_mask)
     plt.axis('off')
     plt.title("Cancer %age: " + str(round((sum(map(sum, region_mask))/
↪slide_dim**2)*100,2)))
     region = read_slide(slide, x=x*slide_dim*dwn_smpls,␣
↪y=y*slide_dim*dwn_smpls, level=level, width=slide_dim, height=slide_dim)
     count = count + 1
     fig.add_subplot(4, 3, count)
     plt.imshow(region)
     count = count + 1
     fig.add_subplot(4, 3, count)
     plt.imshow(region)
     plt.imshow(region_mask, cmap='jet', alpha=0.5)
```



Function definitions

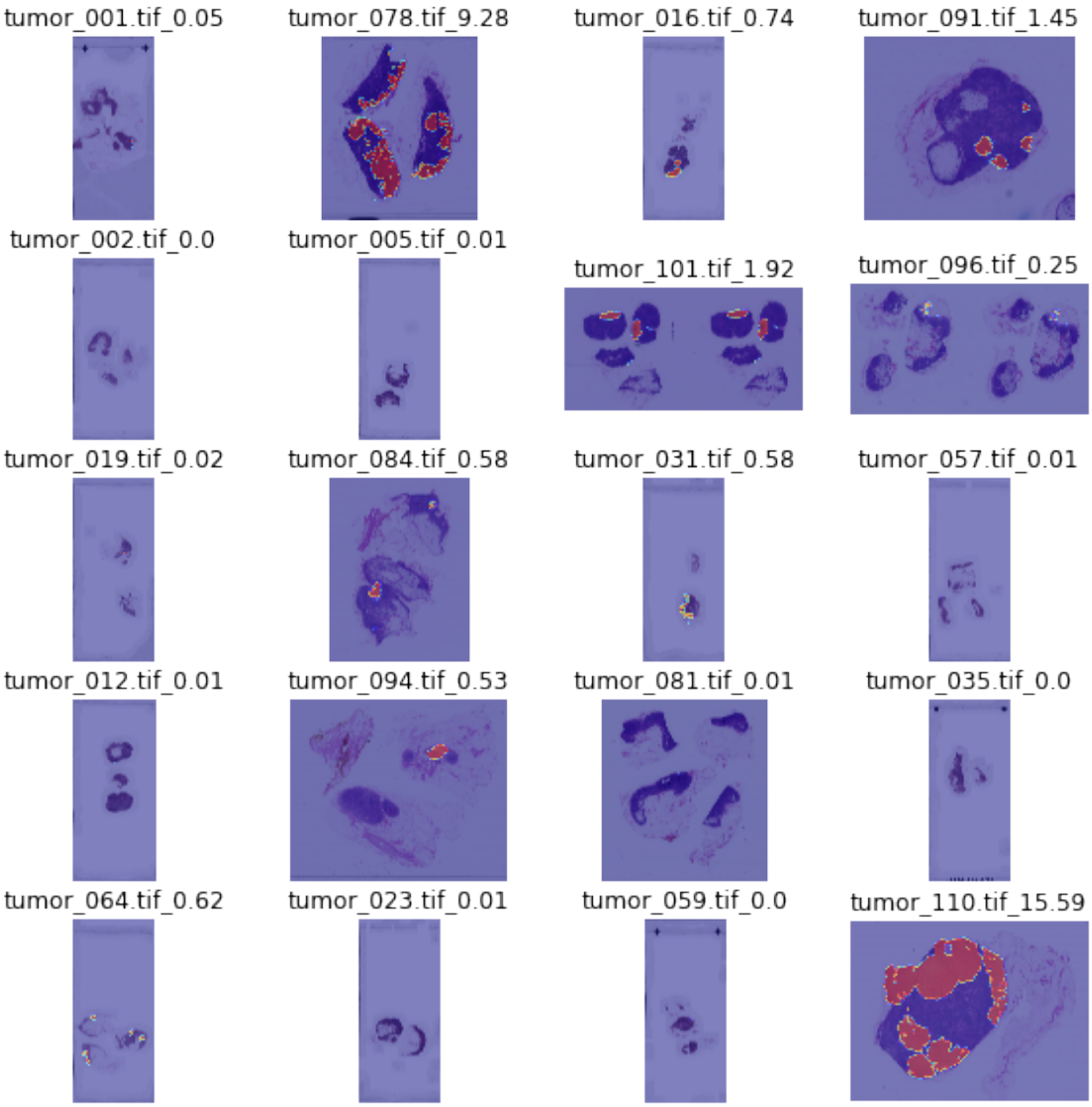
[10]:
```
## the below function returns the pixels that belong to a tissue using an␣
 ↪intensity threshold on the grayscale img
def find_tissue_pixels(image, intensity=0.8):
    im_gray = rgb2gray(image)
    assert im_gray.shape == (image.shape[0], image.shape[1])
    indices = np.where(im_gray <= intensity)
    return list(zip(indices[0], indices[1]))
```

```
## defining a function to perform the sliding window operation on a slide and␣
 ↪save the patches along with the labels
## the function takes in the following inputs
## -- filename (e.g. tumor_075.tif)
## -- zoom level
## -- height and width of the sliding window
## -- thresholds for %age tissue and %age cancer
## -- paths to the slide directory and where to save the patches

def␣
 ↪patch_save(filename,level,height,width,tissue_thresh,cancer_thresh,path_to_dir,path_to_save
 ↪

  slide = open_slide(path_to_dir + filename)
  tumor_mask = open_slide(path_to_dir + filename.split(".")[0] + "_mask." +␣
 ↪filename.split(".")[1])
  dwn_smpls = int(slide.level_downsamples[level])
  x_range = slide.level_dimensions[level][0]
  y_range = slide.level_dimensions[level][1]
  x_lt = int(x_range/width)
  y_lt = int(y_range/height)
  count = 0
  for x in range(x_lt):
    for y in range(y_lt):
      region = read_slide(slide, x= x*width*dwn_smpls, y= y*height*dwn_smpls,␣
 ↪level=level, width=width, height=height)
      region_mask = read_slide(tumor_mask, x=x*width*dwn_smpls,␣
 ↪y=y*height*dwn_smpls, level=level, width=width, height=height)[:,:,0]

      cancer_tissue = round((sum(map(sum, region_mask))/height*width)*100,2)
      percent_tissue = len(find_tissue_pixels(region)) / float(height * width)␣
 ↪* 100
      count = count + 1
      if (percent_tissue > tissue_thresh):
        if cancer_tissue  > cancer_thresh:
          save_file = filename.split(".")[0] + "_patch_" + str(count) + "_lvl_"␣
 ↪+ str(level) + "_" + "01.jpeg"
          cv2.imwrite(path_to_save + save_file,region)
        else:
          save_file = filename.split(".")[0] + "_patch_" + str(count) + "_lvl_"␣
 ↪+ str(level) + "_" + "00.jpeg"
          cv2.imwrite(path_to_save + save_file,region)
      else:
        continue
```

Due to limitation of computational power we look at all the slides and their cancer percentage and select the 9 most decent looking samples just by visual inception (as

we only have 22 slides this works here, in case of large datasets a threshold on tissue and cancer percentage can be created)

[ ]:
```
fig, axs = plt.subplots(5,4, figsize=(10, 10))
for i,x in enumerate(all_image_names):
  slide = open_slide(data_dir + x)
  tumor_mask = open_slide(data_dir + x.split(".")[0] + "_mask."+ x.split(".
 ↪")[1])
  region_mask = read_slide(tumor_mask, x=0, y=0, level=5,  width=slide.
 ↪level_dimensions[5][0], height=slide.level_dimensions[5][1])[:,:,0]
  slide_image = read_slide(slide,
                           x=0,
                           y=0,
                           level=5,
                           width=slide.level_dimensions[5][0],
                           height=slide.level_dimensions[5][1])
  # percent_tissue = round(len(find_tissue_pixels(slide_image)) /␣
 ↪float(slide_image.shape[0] * slide_image.shape[1]),2) * 100
  cancer_tissue = round((sum(map(sum, region_mask))/float(slide_image.shape[0]␣
 ↪* slide_image.shape[1]))*100,2)

  axs[i%5,i%4].axis('off')
  axs[i%5,i%4].set_title(str(x)+"_"+str(cancer_tissue))
  axs[i%5,i%4].imshow(slide_image)
  axs[i%5,i%4].imshow(region_mask, cmap='jet', alpha=0.5)
```

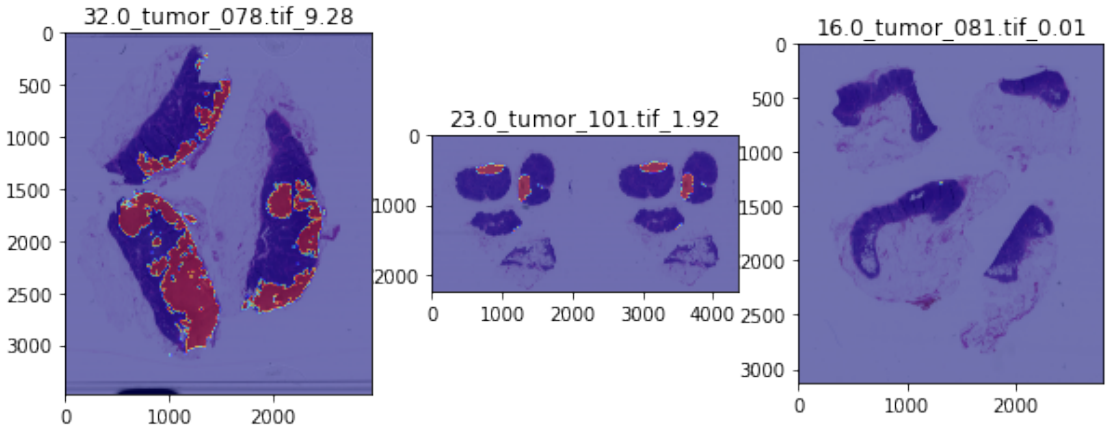


Creating a split of 4, 3, 2 for train, validation and test from the 9 slides above

[12]:
```
train_slide_names = ["tumor_016.tif","tumor_110.tif","tumor_075.tif","tumor_096.
 ↪tif"]
val_slide_names = ["tumor_078.tif","tumor_101.tif","tumor_081.tif"]
test_slide_names = ["tumor_091.tif","tumor_084.tif"]
```

[ ]:
```
## printing the overlap of mask and slides in the validation set
fig, axs = plt.subplots(1,3, figsize=(10, 10))
for i,x in enumerate(val_slide_names):
  slide = open_slide(data_dir + x)
  tumor_mask = open_slide(data_dir + x.split(".")[0] + "_mask."+ x.split(".
 ↪")[1])
```

```
 region_mask = read_slide(tumor_mask, x=0, y=0, level=5,  width=slide.
↪level_dimensions[5][0], height=slide.level_dimensions[5][1])[:,:,0]
 slide_image = read_slide(slide,
                          x=0,
                          y=0,
                          level=5,
                          width=slide.level_dimensions[5][0],
                          height=slide.level_dimensions[5][1])
 percent_tissue = round(len(find_tissue_pixels(slide_image)) /␣
↪float(slide_image.shape[0] * slide_image.shape[1]),2) * 100
 cancer_tissue = round((sum(map(sum, region_mask))/float(slide_image.shape[0]␣
↪* slide_image.shape[1]))*100,2)

 axs[i].set_title(str(percent_tissue)+"_"+str(x)+"_"+str(cancer_tissue))
 axs[i].imshow(slide_image)
 axs[i].imshow(region_mask, cmap='jet', alpha=0.5)
```



[13]:
```
## location of saving the patches
labelled_data_dir = "/content/drive/MyDrive/Final_Project_Data/"
```

[14]:
```
## sanity check
for x in train_slide_names:
  if (x in test_slide_names) or (x in val_slide_names):
    print("Overlapping slides in data split")
for x in val_slide_names:
  if x in test_slide_names:
    print("Overlapping slides in data split")
```

Creating the training, validation and test dataset by calling the patch save function at two different zoom levels and saving the patches in respective folders in Google Drive

[ ]:
```
## running the patch_save function on all the training slides at Zoom level 4␣
 ↪and 5 and saving the patches in GDrive
## we run this with a 25% tissue threshold and a 0% cancer threshold
for x in train_slide_names:
  patch_save(x,4,299,299,25,0,data_dir,labelled_data_dir + "Level4_Train/")
  patch_save(x,3,299,299,25,0,data_dir,labelled_data_dir + "Level3_Train/")
```

[15]:
```
print("We have",len(os.listdir(labelled_data_dir + "Level4_Train")),"Patches in␣
 ↪the Training set at zoom level 4")
print("We have",len(os.listdir(labelled_data_dir + "Level3_Train")),"Patches in␣
 ↪the Training set at zoom level 3")

# saving all the patch paths and their labels in a list
train_image_paths_zoom1 = []
train_labels_zoom1 = []
train_image_paths_zoom2 = []
train_labels_zoom2 = []
for x in os.listdir(labelled_data_dir + "Level4_Train"):
  ## files names have label after the last "_" in the file name (00 or 01)
  label = int((x.split(".")[0]).split("_")[-1])
  train_labels_zoom1.append(label)

  image_path = labelled_data_dir + "Level4_Train/" + x
  train_image_paths_zoom1.append(image_path)

for x in os.listdir(labelled_data_dir + "Level3_Train"):
  ## files names have label after the last "_" in the file name (00 or 01)
  label = int((x.split(".")[0]).split("_")[-1])
  train_labels_zoom2.append(label)

  image_path = labelled_data_dir + "Level3_Train/" + x
  train_image_paths_zoom2.append(image_path)
print("% age of patches having cancer in the training set at Level 4 is: ",␣
 ↪sum(train_labels_zoom1)*100/len(train_labels_zoom1))
print("% age of patches having cancer in the training set at Level 3 is: ",␣
 ↪sum(train_labels_zoom2)*100/len(train_labels_zoom2))
```

```
We have 317 Patches in the Training set at zoom level 4
We have 1194 Patches in the Training set at zoom level 3
% age of patches having cancer in the training set at Level 4 is:
36.90851735015773
% age of patches having cancer in the training set at Level 3 is:
30.569514237855948
```

[ ]:
```
## similarly running the save_patch function to create validation patches at␣
 ↪the same zoom levels
for x in val_slide_names:
```

```
  patch_save(x,4,299,299,25,0,data_dir,labelled_data_dir + "Level4_Val/")
  patch_save(x,3,299,299,25,0,data_dir,labelled_data_dir + "Level3_Val/")
```

[16]:
```
print("We have",len(os.listdir(labelled_data_dir + "Level4_Val")),"Patches in␣
 ↪the Validation set at zoom level 4")
print("We have",len(os.listdir(labelled_data_dir + "Level3_Val")),"Patches in␣
 ↪the Validation set at zoom level 3")

# saving all the patch paths and their labels in a list
val_image_paths_zoom1 = []
val_labels_zoom1 = []
val_image_paths_zoom2 = []
val_labels_zoom2 = []
for x in os.listdir(labelled_data_dir + "Level4_Val"):
  ## files names have label after the last "_" in the file name (00 or 01)
  label = int((x.split(".")[0]).split("_")[-1])
  val_labels_zoom1.append(label)

  image_path = labelled_data_dir + "Level4_Val/" + x
  val_image_paths_zoom1.append(image_path)

for x in os.listdir(labelled_data_dir + "Level3_Val"):
  ## files names have label after the last "_" in the file name (00 or 01)
  label = int((x.split(".")[0]).split("_")[-1])
  val_labels_zoom2.append(label)

  image_path = labelled_data_dir + "Level3_Val/" + x
  val_image_paths_zoom2.append(image_path)
print("% age of patches having cancer in the Validation set at Level 4 is: ",␣
 ↪sum(val_labels_zoom1)*100/len(val_labels_zoom1))
print("% age of patches having cancer in the Validation set at Level 3 is: ",␣
 ↪sum(val_labels_zoom2)*100/len(val_labels_zoom2))
```

```
We have 383 Patches in the Validation set at zoom level 4
We have 1385 Patches in the Validation set at zoom level 3
% age of patches having cancer in the Validation set at Level 4 is:
35.50913838120105
% age of patches having cancer in the Validation set at Level 3 is:
28.231046931407942
```

[ ]:
```
## similarly running the save_patch function to create validation patches at␣
 ↪the same zoom levels
for x in test_slide_names:
  patch_save(x,4,299,299,25,0,data_dir,labelled_data_dir + "Level4_Test/")
  patch_save(x,3,299,299,25,0,data_dir,labelled_data_dir + "Level3_Test/")
```

[17]:
```
print("We have",len(os.listdir(labelled_data_dir + "Level4_Test")),"Patches in␣
 ↪the Test set at zoom level 4")
print("We have",len(os.listdir(labelled_data_dir + "Level3_Test")),"Patches in␣
 ↪the Test set at zoom level 3")

# saving all the patch paths and their labels in a list
test_image_paths_zoom1 = []
test_labels_zoom1 = []
test_image_paths_zoom2 = []
test_labels_zoom2 = []
for x in os.listdir(labelled_data_dir + "Level4_Test"):
  ## files names have label after the last "_" in the file name (00 or 01)
  label = int((x.split(".")[0]).split("_")[-1])
  test_labels_zoom1.append(label)

  image_path = labelled_data_dir + "Level4_Test/" + x
  test_image_paths_zoom1.append(image_path)

for x in os.listdir(labelled_data_dir + "Level3_Test"):
  ## files names have label after the last "_" in the file name (00 or 01)
  label = int((x.split(".")[0]).split("_")[-1])
  test_labels_zoom2.append(label)

  image_path = labelled_data_dir + "Level3_Test/" + x
  test_image_paths_zoom2.append(image_path)
print("% age of patches having cancer in the Validation set at Level 4 is: ",␣
 ↪sum(test_labels_zoom1)*100/len(test_labels_zoom1))
print("% age of patches having cancer in the Validation set at Level 3 is: ",␣
 ↪sum(test_labels_zoom2)*100/len(test_labels_zoom2))
```

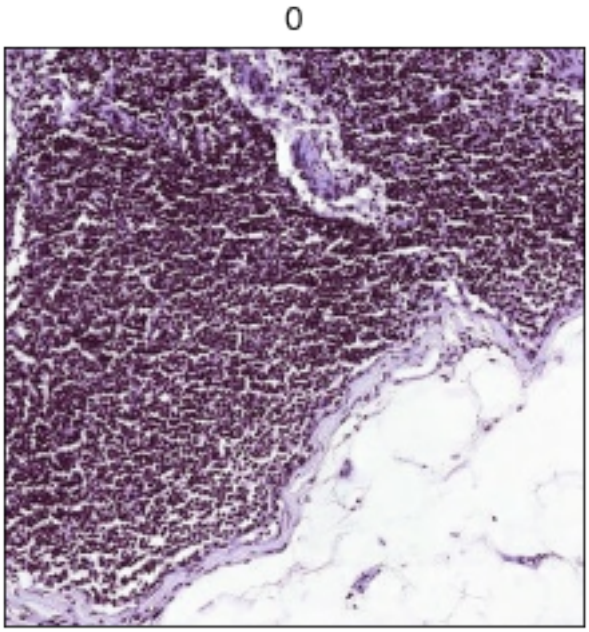
```
We have 126 Patches in the Test set at zoom level 4
We have 466 Patches in the Test set at zoom level 3
% age of patches having cancer in the Validation set at Level 4 is:
19.047619047619047
% age of patches having cancer in the Validation set at Level 3 is:
11.587982832618026
```

Packaging the data into tensorflow format for the model to digest

[18]:
```
## function to read the image from the image paths we pass
## also normalizes the image
def load_and_preprocess_image(img):
  img = tf.io.read_file(img)
  img = tf.image.decode_jpeg(img, channels=3)
  #convert image pixel data type to float and normalize pixels to 0,1
  img_final = tf.cast(img, tf.float32) / 255.0
  return img_final
```

[19]:
```
def show(img, label):
    plt.imshow(img)
    plt.title(label)
    plt.xticks([])
    plt.yticks([])
    print()
```

[20]:
```
## visualizing a patch after preprocessing
img = load_and_preprocess_image(train_image_paths_zoom2[0])
label = train_labels_zoom2[0]
show(img, label)
```



[21]:
```
## having different batch and shuffle sizes
## this is because as we go to higher zoom levels we get more patches
SHUFFLE_SIZE_1 = 25
SHUFFLE_SIZE_2 =  100
BATCH_SIZE_1 = 16
BATCH_SIZE_2 = 32
IMG_SIZE = 299
```

[22]:
```
# a dataset that returns image paths
path_ds = tf.data.Dataset.from_tensor_slices(train_image_paths_zoom1)
# a dataset that returns images (loaded off disk, decoded, and preprocessed)
```

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
image_ds = path_ds.map(load_and_preprocess_image, num_parallel_calls=AUTOTUNE)
# a dataset that returns labels
label_ds = tf.data.Dataset.from_tensor_slices(tf.cast(train_labels_zoom1, tf.
 ↪int64))
# a dataset that returns images and labels
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
# checking the concatenated dataset by prining image shape and label
for img, label in image_label_ds.take(2):
    print(img.shape, label.numpy())
```

```
(299, 299, 3) 0
(299, 299, 3) 0
```

[23]:
```
## using in memory caching as we do not have a large dataset right now
train_ds_zoom1 = image_label_ds.cache()
train_ds_zoom1 = train_ds_zoom1.shuffle(SHUFFLE_SIZE_1)
train_ds_zoom1 = train_ds_zoom1.batch(BATCH_SIZE_1).
 ↪prefetch(buffer_size=AUTOTUNE)
```

[24]:
```
## following the similar steps as above for zoom level 2 for training
path_ds = tf.data.Dataset.from_tensor_slices(train_image_paths_zoom2)
image_ds = path_ds.map(load_and_preprocess_image, num_parallel_calls=AUTOTUNE)
label_ds = tf.data.Dataset.from_tensor_slices(tf.cast(train_labels_zoom2, tf.
 ↪int64))
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
train_ds_zoom2 = image_label_ds.cache()
train_ds_zoom2 = train_ds_zoom2.shuffle(SHUFFLE_SIZE_2)
train_ds_zoom2 = train_ds_zoom2.batch(BATCH_SIZE_2).
 ↪prefetch(buffer_size=AUTOTUNE)
```

[25]:
```
## following the similar steps as above for zoom level 1 for validation
path_ds = tf.data.Dataset.from_tensor_slices(val_image_paths_zoom1)
image_ds = path_ds.map(load_and_preprocess_image, num_parallel_calls=AUTOTUNE)
label_ds = tf.data.Dataset.from_tensor_slices(tf.cast(val_labels_zoom1, tf.
 ↪int64))
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
val_ds_zoom1 = image_label_ds.cache().batch(BATCH_SIZE_1)

## following the similar steps as above for zoom level 2 for validation
path_ds = tf.data.Dataset.from_tensor_slices(val_image_paths_zoom2)
image_ds = path_ds.map(load_and_preprocess_image, num_parallel_calls=AUTOTUNE)
label_ds = tf.data.Dataset.from_tensor_slices(tf.cast(val_labels_zoom2, tf.
 ↪int64))
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
val_ds_zoom2 = image_label_ds.cache().batch(BATCH_SIZE_2)
```

[26]:
```
## following the similar steps as above for zoom level 1 for test
path_ds = tf.data.Dataset.from_tensor_slices(test_image_paths_zoom1)
image_ds = path_ds.map(load_and_preprocess_image, num_parallel_calls=AUTOTUNE)
label_ds = tf.data.Dataset.from_tensor_slices(tf.cast(test_labels_zoom1, tf.
↪int64))
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
test_ds_zoom1 = image_label_ds.cache().batch(BATCH_SIZE_1)

## following the similar steps as above for zoom level 2 for test
path_ds = tf.data.Dataset.from_tensor_slices(test_image_paths_zoom2)
image_ds = path_ds.map(load_and_preprocess_image, num_parallel_calls=AUTOTUNE)
label_ds = tf.data.Dataset.from_tensor_slices(tf.cast(test_labels_zoom2, tf.
↪int64))
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
test_ds_zoom2 = image_label_ds.cache().batch(BATCH_SIZE_2)
```

Below we create two identical Models (one for each zoom level) by using Tranfer Learning and doing fine tuning of the InceptionV3 model

[27]:
```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.layers import GlobalAveragePooling2D, concatenate, Input,␣
↪Dense, Dropout, BatchNormalization, Flatten
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.utils import plot_model

inception_base_zoom1 = InceptionV3(weights='imagenet',
                                   include_top=False,
                                   input_shape=(IMG_SIZE, IMG_SIZE, 3))

inception_base_zoom2 = InceptionV3(weights='imagenet',
                                   include_top=False,
                                   input_shape=(IMG_SIZE, IMG_SIZE, 3))

# freeze the inception model to increase training speed
inception_base_zoom1.trainable = True
inception_base_zoom2.trainable = True

# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(inception_base_zoom1.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer for zoom1 model
for layer in inception_base_zoom1.layers[:fine_tune_at]:
  layer.trainable = False
```

```
# Freeze all the layers before the `fine_tune_at` layer for zoom2 model
for layer in inception_base_zoom2.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 3s 0us/step
Number of layers in the base model:  311
```

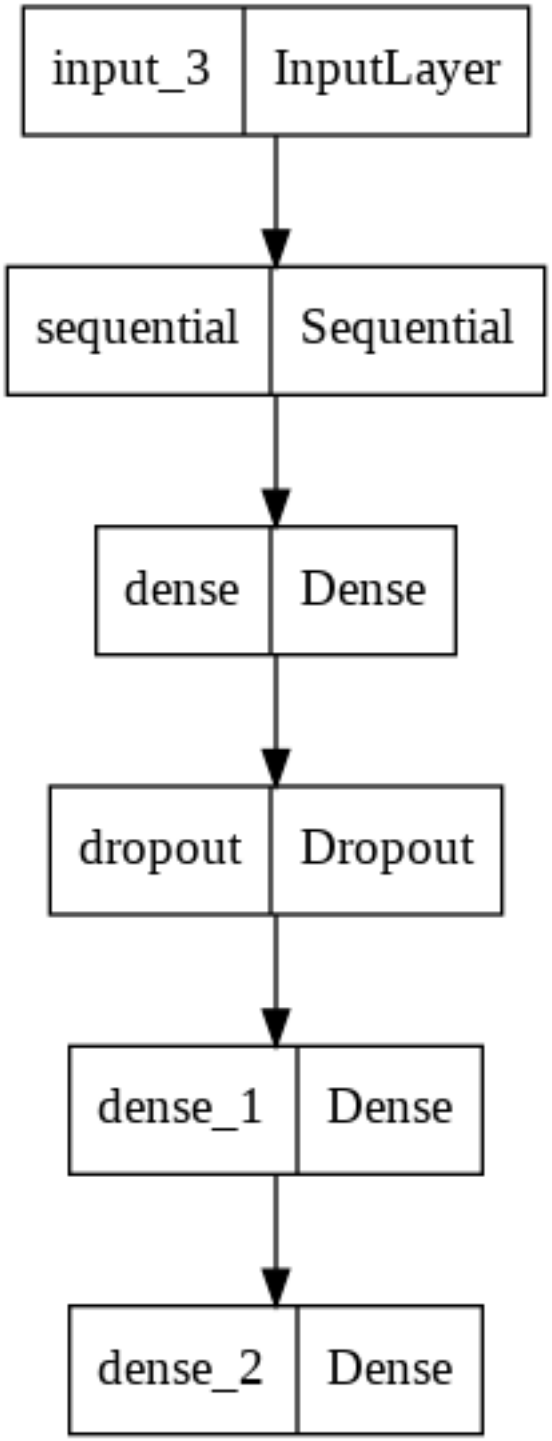
[28]:
```
input_z1 = Input(shape=(IMG_SIZE, IMG_SIZE, 3))
input_z2 = Input(shape=(IMG_SIZE, IMG_SIZE, 3))


## we do global average pooling after adding the inception model to reduce the␣
 ↪number of trainable params
model_z1 = Sequential()
model_z1.add(inception_base_zoom1)
model_z1.add(GlobalAveragePooling2D())

model_z2 = Sequential()
model_z2.add(inception_base_zoom2)
model_z2.add(GlobalAveragePooling2D())

encoded_input_z1 = model_z1(input_z1)
encoded_input_z2 = model_z2(input_z2)
```

[29]:
```
## create a model for the 1st zoom level
dense1 = Dense(256, activation='relu')(encoded_input_z1)
drop_layer = Dropout(0.5)(dense1)
dense2 = Dense(126, activation='relu')(drop_layer)

output = Dense(1, activation='sigmoid')(dense2)
model1 = Model(inputs=input_z1, outputs=output)
```

[30]:
```
## create a model for the 2nd zoom level
dense1 = Dense(256, activation='relu')(encoded_input_z2)
drop_layer = Dropout(0.5)(dense1)
dense2 = Dense(126, activation='relu')(drop_layer)

output = Dense(1, activation='sigmoid')(dense2)
model2 = Model(inputs=input_z2, outputs=output)
```

[31]:
```
## we have two models like the one shown below in the plot
plot_model(model1, to_file='model.png')
```

[31]:



[32]:
```
model1.summary()
```

```
Model: "model"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 299, 299, 3)]     0

 sequential (Sequential)     (None, 2048)              21802784

 dense (Dense)               (None, 256)               524544

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 126)               32382

 dense_2 (Dense)             (None, 1)                 127

=================================================================
Total params: 22,359,837
Trainable params: 20,183,421
Non-trainable params: 2,176,416
_________________________________________________________________
```

[33]:
```
model1.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['accuracy'])
model2.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['accuracy'])
```

Below we train both the models and create checkpoints to save the weights after every epoch

[39]:
```
## creating checkpoint directory
checkpoint_path1 =  labelled_data_dir + "training2_dec22_batch_8_lvl_4/cp.ckpt"
checkpoint_dir1 = os.path.dirname(checkpoint_path1)

## loading the weights from the last checkpoint before we further continue␣
 ↪training
latest1 = tf.train.latest_checkpoint(checkpoint_dir1)
if latest1 != None:
  print("Loading weights from", latest1)
  model1.load_weights(latest1)
else:
  print("Checkpoint not found. Starting from scratch")
```

```
Loading weights from
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
```

[ ]:
```
# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path1,
                                                 save_weights_only=True,
                                                 verbose=1)

# Train the model with the new callback
hist1 = model1.
 ↪fit(train_ds_zoom1,validation_data=val_ds_zoom1,epochs=20,callbacks=[cp_callback])
```

```
Epoch 1/20
40/40 [==============================] - ETA: 0s - loss: 0.7118 - accuracy:
0.6688
Epoch 1: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 177ms/step - loss: 0.7118 -
accuracy: 0.6688 - val_loss: 6.4968 - val_accuracy: 0.6710
Epoch 2/20
40/40 [==============================] - ETA: 0s - loss: 0.6050 - accuracy:
0.7476
Epoch 2: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 170ms/step - loss: 0.6050 -
accuracy: 0.7476 - val_loss: 2.6530 - val_accuracy: 0.6762
Epoch 3/20
40/40 [==============================] - ETA: 0s - loss: 0.6341 - accuracy:
0.7192
Epoch 3: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 167ms/step - loss: 0.6341 -
accuracy: 0.7192 - val_loss: 2.0366 - val_accuracy: 0.6475
Epoch 4/20
40/40 [==============================] - ETA: 0s - loss: 0.5804 - accuracy:
0.7855
Epoch 4: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 179ms/step - loss: 0.5804 -
accuracy: 0.7855 - val_loss: 3.3922 - val_accuracy: 0.6841
Epoch 5/20
40/40 [==============================] - ETA: 0s - loss: 0.6062 - accuracy:
0.6562
Epoch 5: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 172ms/step - loss: 0.6062 -
accuracy: 0.6562 - val_loss: 2.4476 - val_accuracy: 0.6762
Epoch 6/20
40/40 [==============================] - ETA: 0s - loss: 0.5770 - accuracy:
0.7539
```

```
Epoch 6: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 169ms/step - loss: 0.5770 -
accuracy: 0.7539 - val_loss: 1.2721 - val_accuracy: 0.6867
Epoch 7/20
40/40 [==============================] - ETA: 0s - loss: 0.5552 - accuracy:
0.7603
Epoch 7: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 8s 193ms/step - loss: 0.5552 -
accuracy: 0.7603 - val_loss: 0.5684 - val_accuracy: 0.7180
Epoch 8/20
40/40 [==============================] - ETA: 0s - loss: 0.3707 - accuracy:
0.8549
Epoch 8: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 174ms/step - loss: 0.3707 -
accuracy: 0.8549 - val_loss: 2.6376 - val_accuracy: 0.6919
Epoch 9/20
40/40 [==============================] - ETA: 0s - loss: 0.5659 - accuracy:
0.7792
Epoch 9: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 168ms/step - loss: 0.5659 -
accuracy: 0.7792 - val_loss: 0.8598 - val_accuracy: 0.6110
Epoch 10/20
40/40 [==============================] - ETA: 0s - loss: 0.4609 - accuracy:
0.8170
Epoch 10: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 177ms/step - loss: 0.4609 -
accuracy: 0.8170 - val_loss: 3.7010 - val_accuracy: 0.6449
Epoch 11/20
40/40 [==============================] - ETA: 0s - loss: 0.6006 - accuracy:
0.7886
Epoch 11: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 180ms/step - loss: 0.6006 -
accuracy: 0.7886 - val_loss: 0.9316 - val_accuracy: 0.5979
Epoch 12/20
40/40 [==============================] - ETA: 0s - loss: 0.6139 - accuracy:
0.6909
Epoch 12: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 171ms/step - loss: 0.6139 -
accuracy: 0.6909 - val_loss: 0.9265 - val_accuracy: 0.7441
Epoch 13/20
40/40 [==============================] - ETA: 0s - loss: 0.5046 - accuracy:
```

```
0.8139
Epoch 13: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 180ms/step - loss: 0.5046 -
accuracy: 0.8139 - val_loss: 13.8190 - val_accuracy: 0.3969
Epoch 14/20
40/40 [==============================] - ETA: 0s - loss: 0.5641 - accuracy:
0.7413
Epoch 14: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 172ms/step - loss: 0.5641 -
accuracy: 0.7413 - val_loss: 1.7725 - val_accuracy: 0.6527
Epoch 15/20
40/40 [==============================] - ETA: 0s - loss: 0.4687 - accuracy:
0.7918
Epoch 15: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 170ms/step - loss: 0.4687 -
accuracy: 0.7918 - val_loss: 1.3162 - val_accuracy: 0.6815
Epoch 16/20
40/40 [==============================] - ETA: 0s - loss: 0.3652 - accuracy:
0.8517
Epoch 16: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 169ms/step - loss: 0.3652 -
accuracy: 0.8517 - val_loss: 0.9764 - val_accuracy: 0.6423
Epoch 17/20
40/40 [==============================] - ETA: 0s - loss: 0.4148 - accuracy:
0.8486
Epoch 17: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 168ms/step - loss: 0.4148 -
accuracy: 0.8486 - val_loss: 0.9407 - val_accuracy: 0.7389
Epoch 18/20
40/40 [==============================] - ETA: 0s - loss: 0.2821 - accuracy:
0.8927
Epoch 18: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 172ms/step - loss: 0.2821 -
accuracy: 0.8927 - val_loss: 0.7549 - val_accuracy: 0.6449
Epoch 19/20
40/40 [==============================] - ETA: 0s - loss: 0.3888 - accuracy:
0.8959
Epoch 19: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 183ms/step - loss: 0.3888 -
accuracy: 0.8959 - val_loss: 1.3595 - val_accuracy: 0.6867
Epoch 20/20
```

```
40/40 [==============================] - ETA: 0s - loss: 0.2920 - accuracy:
0.9401
Epoch 20: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_8_lvl_4/cp.ckpt
40/40 [==============================] - 7s 167ms/step - loss: 0.2920 -
accuracy: 0.9401 - val_loss: 0.7707 - val_accuracy: 0.7050
```

[34]:
```
checkpoint_path2 =  labelled_data_dir + "training2_dec22_batch_16_lvl_3/cp.ckpt"
checkpoint_dir2 = os.path.dirname(checkpoint_path2)

latest2 = tf.train.latest_checkpoint(checkpoint_dir2)
if latest2 != None:
  print("Loading weights from", latest2)
  model2.load_weights(latest2)
else:
  print("Checkpoint not found. Starting from scratch")
```

```
Checkpoint not found. Starting from scratch
```

[35]:
```
# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path2,
                                                 save_weights_only=True,
                                                 verbose=1)

# Train the model with the new callback
hist2 = model2.
 ↪fit(train_ds_zoom2,validation_data=val_ds_zoom2,epochs=20,callbacks=[cp_callback])
```

```
Epoch 1/20
38/38 [==============================] - ETA: 0s - loss: 0.4481 - accuracy:
0.8208
Epoch 1: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 657s 16s/step - loss: 0.4481 -
accuracy: 0.8208 - val_loss: 16.7383 - val_accuracy: 0.7177
Epoch 2/20
38/38 [==============================] - ETA: 0s - loss: 0.5809 - accuracy:
0.8015
Epoch 2: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 454ms/step - loss: 0.5809 -
accuracy: 0.8015 - val_loss: 1095.0538 - val_accuracy: 0.7177
Epoch 3/20
38/38 [==============================] - ETA: 0s - loss: 0.5520 - accuracy:
0.7680
Epoch 3: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 451ms/step - loss: 0.5520 -
```

```
accuracy: 0.7680 - val_loss: 39.6817 - val_accuracy: 0.3495
Epoch 4/20
38/38 [==============================] - ETA: 0s - loss: 0.4302 - accuracy:
0.8509
Epoch 4: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 454ms/step - loss: 0.4302 -
accuracy: 0.8509 - val_loss: 70.6496 - val_accuracy: 0.7126
Epoch 5/20
38/38 [==============================] - ETA: 0s - loss: 0.5743 - accuracy:
0.7831
Epoch 5: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 459ms/step - loss: 0.5743 -
accuracy: 0.7831 - val_loss: 5.9755 - val_accuracy: 0.7076
Epoch 6/20
38/38 [==============================] - ETA: 0s - loss: 0.4044 - accuracy:
0.8585
Epoch 6: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 448ms/step - loss: 0.4044 -
accuracy: 0.8585 - val_loss: 15.4169 - val_accuracy: 0.7112
Epoch 7/20
38/38 [==============================] - ETA: 0s - loss: 0.4533 - accuracy:
0.8425
Epoch 7: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 461ms/step - loss: 0.4533 -
accuracy: 0.8425 - val_loss: 3.4799 - val_accuracy: 0.7509
Epoch 8/20
38/38 [==============================] - ETA: 0s - loss: 0.3730 - accuracy:
0.8760
Epoch 8: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 446ms/step - loss: 0.3730 -
accuracy: 0.8760 - val_loss: 2.2902 - val_accuracy: 0.7025
Epoch 9/20
38/38 [==============================] - ETA: 0s - loss: 0.3127 - accuracy:
0.8911
Epoch 9: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 447ms/step - loss: 0.3127 -
accuracy: 0.8911 - val_loss: 0.3807 - val_accuracy: 0.8708
Epoch 10/20
38/38 [==============================] - ETA: 0s - loss: 0.2460 - accuracy:
0.9305
Epoch 10: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
```

```
38/38 [==============================] - 17s 457ms/step - loss: 0.2460 -
accuracy: 0.9305 - val_loss: 0.8465 - val_accuracy: 0.7422
Epoch 11/20
38/38 [==============================] - ETA: 0s - loss: 0.2441 - accuracy:
0.9196
Epoch 11: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 454ms/step - loss: 0.2441 -
accuracy: 0.9196 - val_loss: 1.8561 - val_accuracy: 0.8079
Epoch 12/20
38/38 [==============================] - ETA: 0s - loss: 0.2465 - accuracy:
0.9213
Epoch 12: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 446ms/step - loss: 0.2465 -
accuracy: 0.9213 - val_loss: 0.3539 - val_accuracy: 0.9249
Epoch 13/20
38/38 [==============================] - ETA: 0s - loss: 0.2013 - accuracy:
0.9447
Epoch 13: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 460ms/step - loss: 0.2013 -
accuracy: 0.9447 - val_loss: 3.4616 - val_accuracy: 0.5292
Epoch 14/20
38/38 [==============================] - ETA: 0s - loss: 0.1771 - accuracy:
0.9414
Epoch 14: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 449ms/step - loss: 0.1771 -
accuracy: 0.9414 - val_loss: 1.2235 - val_accuracy: 0.8137
Epoch 15/20
38/38 [==============================] - ETA: 0s - loss: 0.1767 - accuracy:
0.9481
Epoch 15: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 460ms/step - loss: 0.1767 -
accuracy: 0.9481 - val_loss: 0.8978 - val_accuracy: 0.8570
Epoch 16/20
38/38 [==============================] - ETA: 0s - loss: 0.2025 - accuracy:
0.9389
Epoch 16: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 449ms/step - loss: 0.2025 -
accuracy: 0.9389 - val_loss: 0.5122 - val_accuracy: 0.8245
Epoch 17/20
38/38 [==============================] - ETA: 0s - loss: 0.1857 - accuracy:
0.9464
Epoch 17: saving model to
```

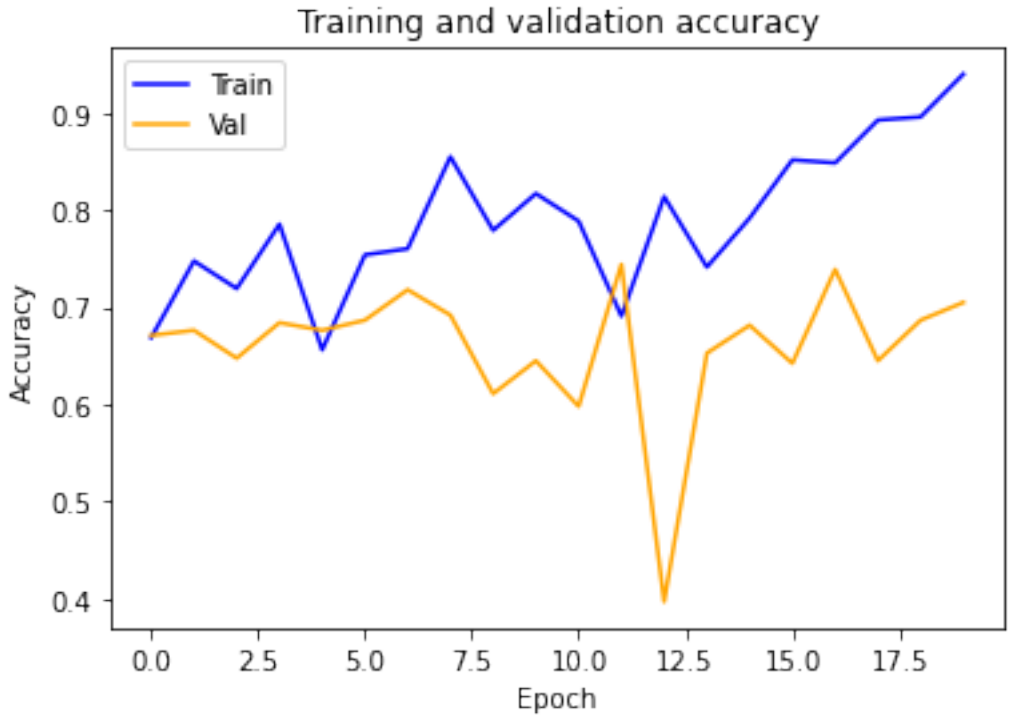
```
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 447ms/step - loss: 0.1857 -
accuracy: 0.9464 - val_loss: 0.2280 - val_accuracy: 0.9350
Epoch 18/20
38/38 [==============================] - ETA: 0s - loss: 0.0982 - accuracy:
0.9665
Epoch 18: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 457ms/step - loss: 0.0982 -
accuracy: 0.9665 - val_loss: 1.1710 - val_accuracy: 0.6628
Epoch 19/20
38/38 [==============================] - ETA: 0s - loss: 0.1137 - accuracy:
0.9673
Epoch 19: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 449ms/step - loss: 0.1137 -
accuracy: 0.9673 - val_loss: 0.3826 - val_accuracy: 0.8845
Epoch 20/20
38/38 [==============================] - ETA: 0s - loss: 0.1162 - accuracy:
0.9665
Epoch 20: saving model to
/content/drive/MyDrive/Final_Project_Data/training2_dec22_batch_16_lvl_3/cp.ckpt
38/38 [==============================] - 17s 447ms/step - loss: 0.1162 -
accuracy: 0.9665 - val_loss: 0.4748 - val_accuracy: 0.8621
```

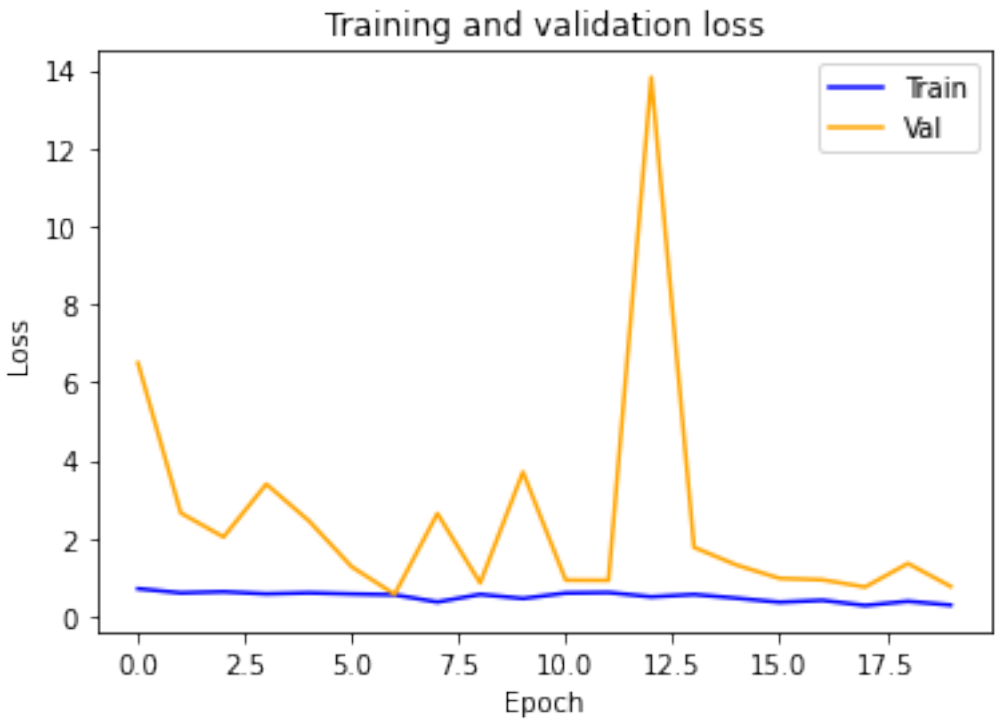
Plotting the validation and training accuracy/loss

[37]:
```python
def plot(history):
  # The history object contains results on the training and test
  # sets for each epoch
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    # Get the number of epochs
    epochs = range(len(acc))

    plt.title('Training and validation accuracy')
    plt.plot(epochs, acc, color='blue', label='Train')
    plt.plot(epochs, val_acc, color='orange', label='Val')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    _ = plt.figure()
    plt.title('Training and validation loss')
    plt.plot(epochs, loss, color='blue', label='Train')
```

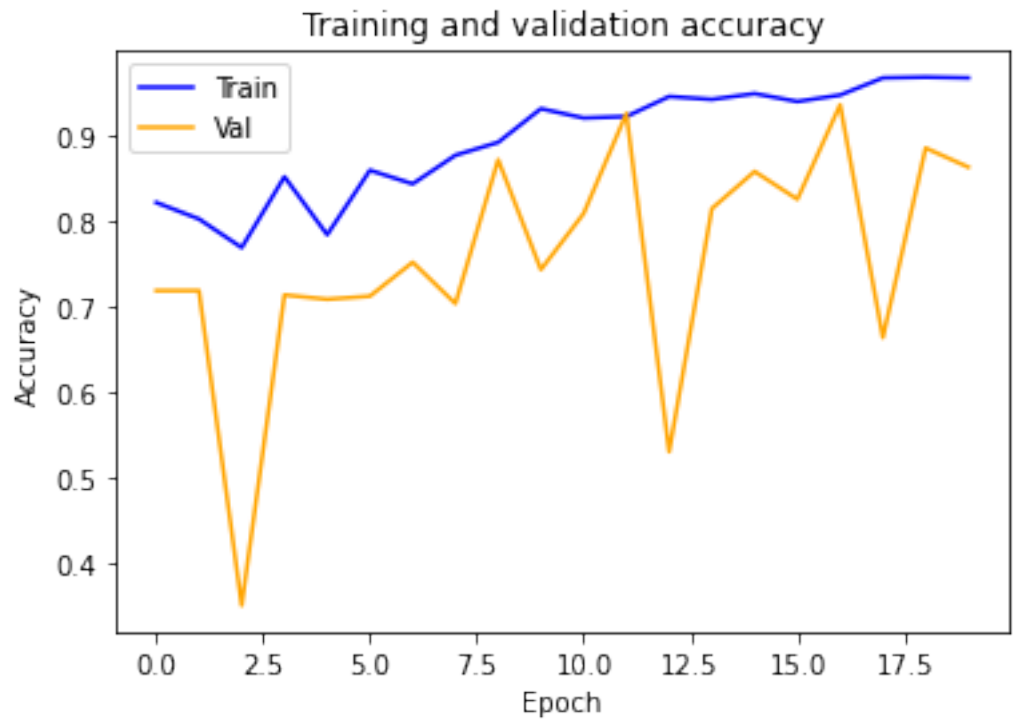
```
    plt.plot(epochs, val_loss, color='orange', label='Val')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
```

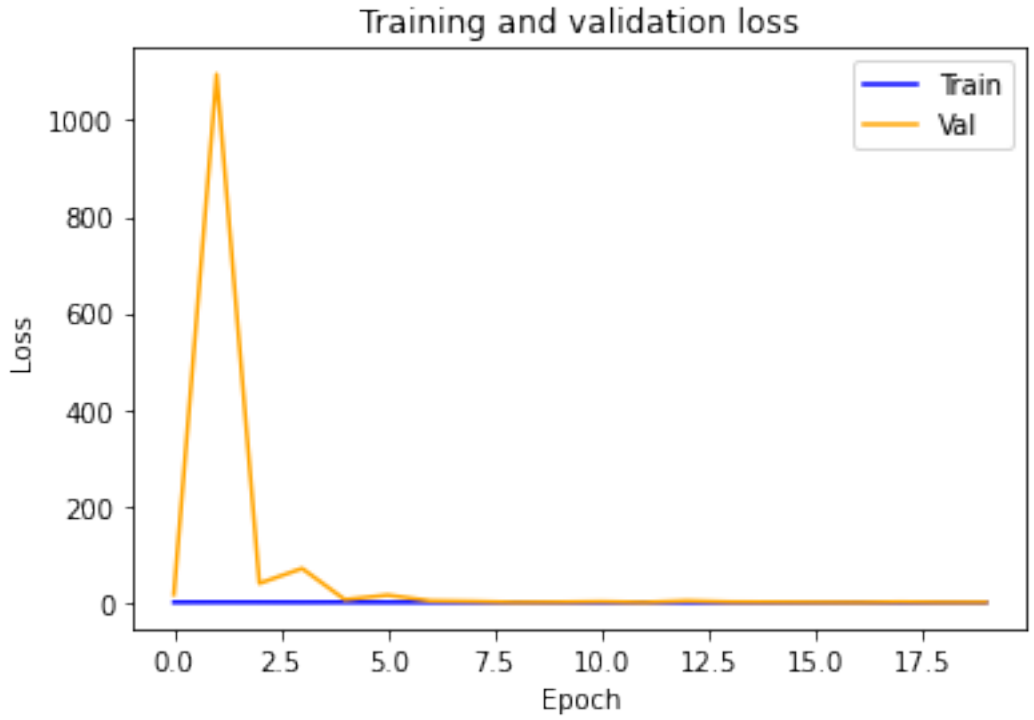
[ ]:
```
plot(hist1)
```





[38]:
```
## plotting the training and validation loss and accuracy over the epochs for␣
↪model2
plot(hist2)
```


Training and validation accuracy
Train
Val
Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Training and validation loss
Train
Val
Loss
1000
800
600
400
200
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

Testing both the models on the test dataset we created

[40]:
```
## define a function to evaluate the model using the test set
def test_evaluate(test_ds,model,max_steps=None):
  steps = 0
  for path_batch,label_batch in test_ds:
    if max_steps != None and steps == max_steps:
      break
    predictions = model.predict(path_batch)
    steps += 1
    # Record metrics after each batch
    test_loss(label_batch, predictions)
    test_accuracy(label_batch, predictions)
```

[48]:
```
test_loss = tf.keras.metrics.BinaryCrossentropy(name='test_loss')
test_accuracy = tf.keras.metrics.BinaryAccuracy(name='test_accuracy')
test_evaluate(test_ds_zoom1,model1)
```

```
1/1 [==============================] - 5s 5s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 5s 5s/step
1/1 [==============================] - 4s 4s/step
1/1 [==============================] - 3s 3s/step
```

[51]:
```
print("The test loss for model1 (Zoom level 4) is: ",test_loss.result().
↪numpy(), "\nThe test accuracy is: ",test_accuracy.result().numpy()*100,"%")
```

```
The test loss for model1 (Zoom level 4) is:  0.77775586
The test accuracy is:  78.57142686843872 %
```

[41]:
```
test_loss = tf.keras.metrics.BinaryCrossentropy(name='test_loss')
test_accuracy = tf.keras.metrics.BinaryAccuracy(name='test_accuracy')
test_evaluate(test_ds_zoom2,model2)
```

```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 44ms/step
```

```
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 3s 3s/step
```

[42]:
```
print("The test loss for model2 (Zoom level 3) is: ",test_loss.result().
↪numpy(), "\nThe test accuracy is: ",test_accuracy.result().numpy()*100,"%")
```

```
The test loss for model2 (Zoom level 3) is:  0.32616392
The test accuracy is:  91.84549450874329 %
```

Below we define a function to generate a mask heatmap which would tell us as to where the cancer is present using the above models that we trained

[43]:
```
def create_mask(filename,model,level,height,width,path_to_dir):

  slide = open_slide(path_to_dir + filename)
  tumor_mask = open_slide(path_to_dir + filename.split(".")[0] + "_mask." +␣
↪filename.split(".")[1])
  dwn_smpls = int(slide.level_downsamples[level])
  x_range = slide.level_dimensions[level][0]
  y_range = slide.level_dimensions[level][1]
  x_lt = int(x_range/width)
  y_lt = int(y_range/height)
  prediction = []
  tissue_perc = []
  count = 0
  for x in range(x_lt):
    for y in range(y_lt):

      region = read_slide(slide, x=x*width*dwn_smpls, y=y*height*dwn_smpls,␣
↪level=level, width=width, height=height)
      region_mask = read_slide(tumor_mask, x=x*width*dwn_smpls,␣
↪y=y*height*dwn_smpls, level=level, width=width, height=height)[:,:,0]

      percent_tissue = len(find_tissue_pixels(region)) *100 / float(height *␣
↪width)
      count = count + 1
      tissue_perc.append(percent_tissue)
      temp = model.predict(np.expand_dims(region, axis=0))[0][0]
      ## converting any non zero prediction value to 1
      if temp > 0:
        temp = 1
      prediction.append(temp)
  return np.array(prediction).reshape((x_lt,y_lt))
```

[56]:
```
test_file = "tumor_078.tif"
test_pred = create_mask(test_file,model1,4,299,299,data_dir)
```

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 343ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 410ms/step
1/1 [==============================] - 0s 469ms/step
1/1 [==============================] - 0s 404ms/step
1/1 [==============================] - 0s 400ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 330ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 254ms/step
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 292ms/step
1/1 [==============================] - 0s 284ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 302ms/step
1/1 [==============================] - 0s 259ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 298ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 283ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 267ms/step
```

```
1/1 [==============================] - 0s 259ms/step
1/1 [==============================] - 0s 296ms/step
1/1 [==============================] - 0s 445ms/step
1/1 [==============================] - 0s 483ms/step
1/1 [==============================] - 0s 452ms/step
1/1 [==============================] - 0s 353ms/step
1/1 [==============================] - 0s 323ms/step
1/1 [==============================] - 0s 409ms/step
1/1 [==============================] - 0s 425ms/step
1/1 [==============================] - 0s 387ms/step
1/1 [==============================] - 0s 429ms/step
1/1 [==============================] - 0s 296ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 280ms/step
1/1 [==============================] - 0s 283ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 293ms/step
1/1 [==============================] - 0s 289ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 260ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 280ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 289ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 287ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 287ms/step
1/1 [==============================] - 0s 263ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 284ms/step
```

```
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 289ms/step
1/1 [==============================] - 0s 339ms/step
1/1 [==============================] - 0s 477ms/step
1/1 [==============================] - 0s 447ms/step
1/1 [==============================] - 0s 361ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 288ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 296ms/step
1/1 [==============================] - 0s 287ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 305ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 305ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 284ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 302ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 335ms/step
1/1 [==============================] - 0s 438ms/step
1/1 [==============================] - 0s 489ms/step
1/1 [==============================] - 1s 651ms/step
1/1 [==============================] - 0s 293ms/step
1/1 [==============================] - 0s 353ms/step
1/1 [==============================] - 0s 476ms/step
1/1 [==============================] - 0s 405ms/step
1/1 [==============================] - 1s 654ms/step
1/1 [==============================] - 0s 448ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 290ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 316ms/step
```

```
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 293ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 290ms/step
1/1 [==============================] - 1s 519ms/step
1/1 [==============================] - 1s 537ms/step
1/1 [==============================] - 0s 445ms/step
1/1 [==============================] - 1s 548ms/step
1/1 [==============================] - 0s 440ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 325ms/step
1/1 [==============================] - 0s 459ms/step
1/1 [==============================] - 1s 555ms/step
1/1 [==============================] - 1s 617ms/step
1/1 [==============================] - 1s 566ms/step
1/1 [==============================] - 1s 604ms/step
1/1 [==============================] - 1s 608ms/step
1/1 [==============================] - 1s 524ms/step
1/1 [==============================] - 1s 690ms/step
1/1 [==============================] - 1s 546ms/step
1/1 [==============================] - 1s 892ms/step
1/1 [==============================] - 0s 450ms/step
1/1 [==============================] - 1s 515ms/step
1/1 [==============================] - 1s 554ms/step
1/1 [==============================] - 1s 514ms/step
1/1 [==============================] - 1s 505ms/step
1/1 [==============================] - 0s 481ms/step
1/1 [==============================] - 0s 290ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 296ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 265ms/step
```

```
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 288ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 288ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 434ms/step
1/1 [==============================] - 0s 310ms/step
1/1 [==============================] - 0s 334ms/step
1/1 [==============================] - 0s 307ms/step
1/1 [==============================] - 0s 316ms/step
1/1 [==============================] - 0s 333ms/step
1/1 [==============================] - 0s 302ms/step
1/1 [==============================] - 0s 322ms/step
1/1 [==============================] - 1s 700ms/step
1/1 [==============================] - 1s 710ms/step
1/1 [==============================] - 0s 415ms/step
1/1 [==============================] - 0s 433ms/step
1/1 [==============================] - 0s 380ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 300ms/step
1/1 [==============================] - 0s 289ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 280ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 287ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 289ms/step
1/1 [==============================] - 0s 277ms/step
```

```
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 292ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 293ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 295ms/step
1/1 [==============================] - 0s 260ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 260ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 263ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 284ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 289ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 284ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 294ms/step
1/1 [==============================] - 0s 283ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 257ms/step
1/1 [==============================] - 0s 289ms/step
1/1 [==============================] - 0s 264ms/step
```

```
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 263ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 260ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 288ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 280ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 302ms/step
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 280ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 297ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 290ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 268ms/step
```

```
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 284ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 291ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 260ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 289ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 283ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 298ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 263ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 285ms/step
```

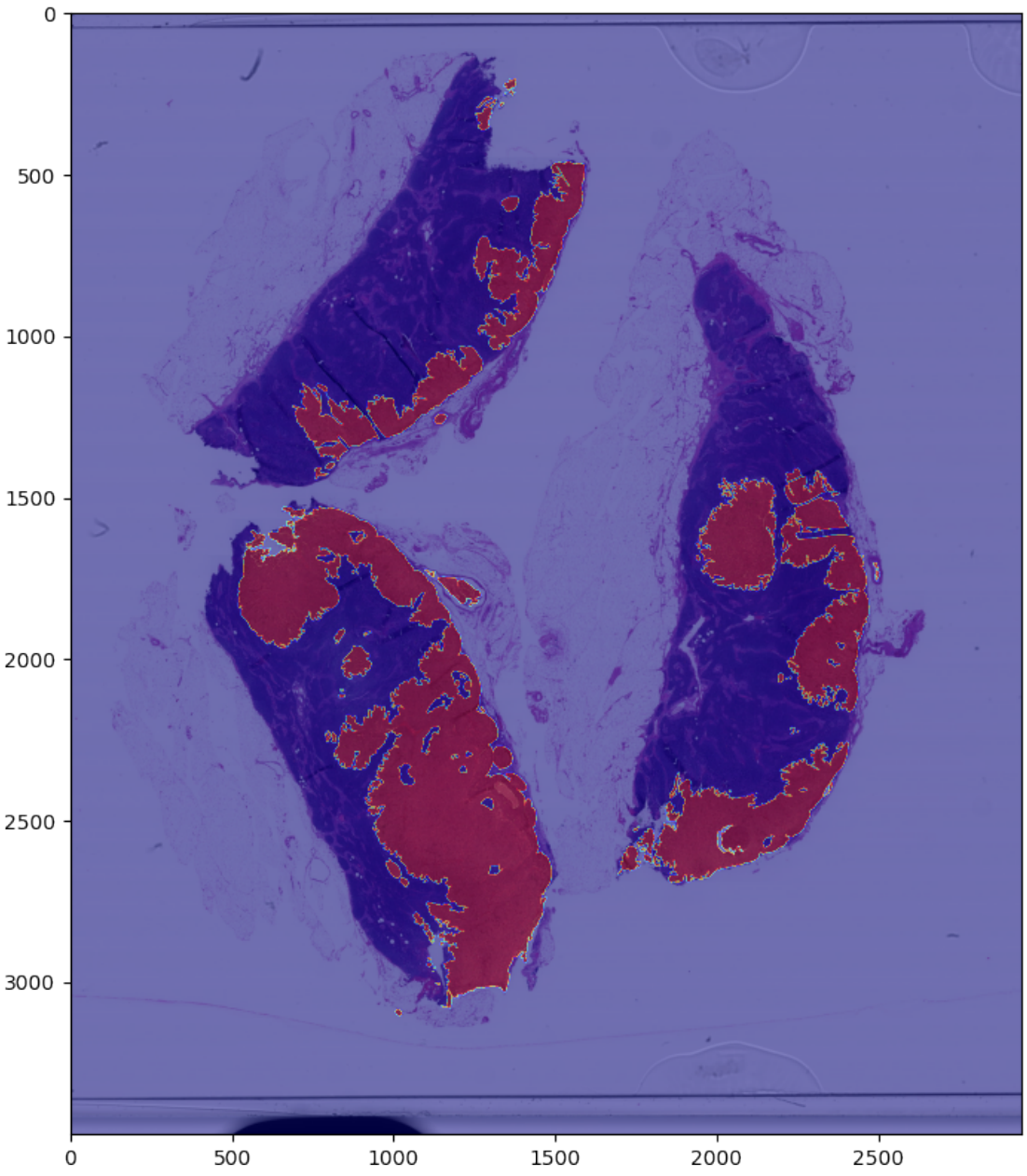
```
1/1 [==============================] - 0s 263ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 280ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 274ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 280ms/step
1/1 [==============================] - 0s 261ms/step
1/1 [==============================] - 0s 309ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 261ms/step
1/1 [==============================] - 0s 290ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 273ms/step
1/1 [==============================] - 0s 270ms/step
1/1 [==============================] - 0s 285ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 261ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 275ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 261ms/step
1/1 [==============================] - 0s 260ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 319ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 263ms/step
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 288ms/step
```

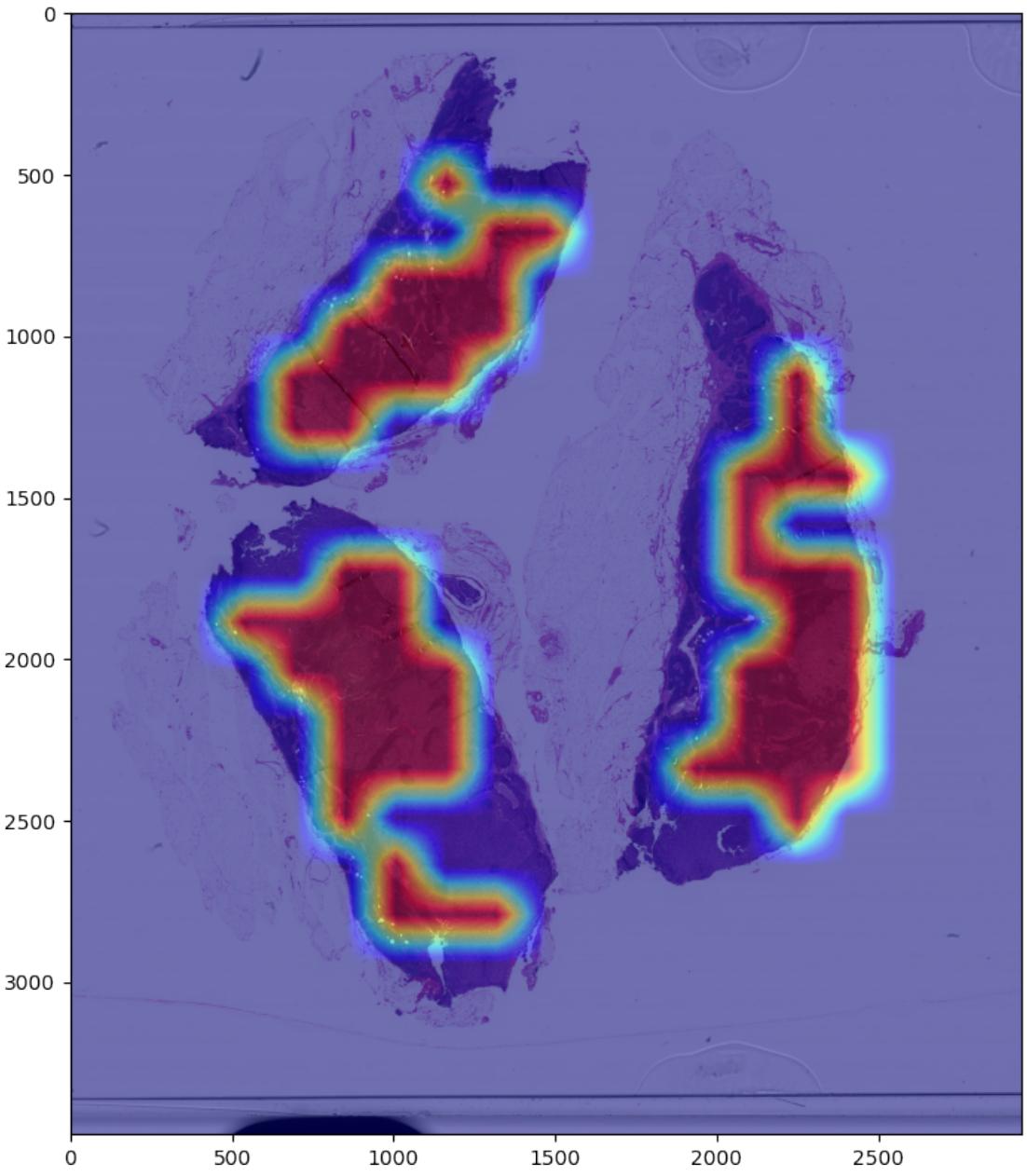
```
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 282ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 270ms/step
```

Plotting the overlap of the mask and slide by using both the original mask and the predicted heatmap mask

[62]:
```
slide_test = open_slide(data_dir + test_file)
mask_test = open_slide(data_dir + test_file.split(".")[0] + "_mask."+ test_file.
 ↪split(".")[1])

slide_image = read_slide(slide_test,
                         x=0,
                         y=0,
                         level=5,
                         width=slide_test.level_dimensions[5][0],
                         height=slide_test.level_dimensions[5][1])
mask_image = read_slide(mask_test,
                         x=0,
                         y=0,
                         level=5,
                         width=mask_test.level_dimensions[5][0],
                         height=mask_test.level_dimensions[5][1])[:,:,0]
predict_mask = cv2.resize(test_pred.T,(slide_test.
 ↪level_dimensions[5][0],slide_test.level_dimensions[5][1]))

plt.figure(figsize=(10,10), dpi=100)
plt.imshow(slide_image)
plt.imshow(mask_image, cmap='jet', alpha=0.5)

plt.figure(figsize=(10,10), dpi=100)
plt.imshow(slide_image)
plt.imshow(predict_mask, cmap='jet', alpha=0.5)
```

[62]:
```
<matplotlib.image.AxesImage at 0x7f617c4d5df0>
```


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500