Importing Packages

In [1]:
```
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader, ConcatDataset
import glob
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score
import random
import cv2
import sys
```

Reading the images

In [2]:
```
tumor = []
healthy = []
for f in glob.iglob("./Dataset/brain_tumor_dataset/yes/*.jpg"):
    img = cv2.imread(f)
    img = cv2.resize(img,(128,128))
    b, g, r = cv2.split(img)
    img = cv2.merge([r,g,b])
    tumor.append(img)

for f in glob.iglob("./Dataset/brain_tumor_dataset/no/*.jpg"):
    img = cv2.imread(f)
    img = cv2.resize(img,(128,128))
    b, g, r = cv2.split(img)
    img = cv2.merge([r,g,b])
    healthy.append(img)
```

In [3]:
```
print(len(healthy))
print(len(tumor))
```

```
1500
1500
```

In [4]:
```
healthy = np.array(healthy)
tumor = np.array(tumor)
All = np.concatenate((healthy, tumor))
```

In [5]:
```
healthy.shape
```

Out[5]: (1500, 128, 128, 3)

In [6]:
```
tumor.shape
```

Out[6]: (1500, 128, 128, 3)

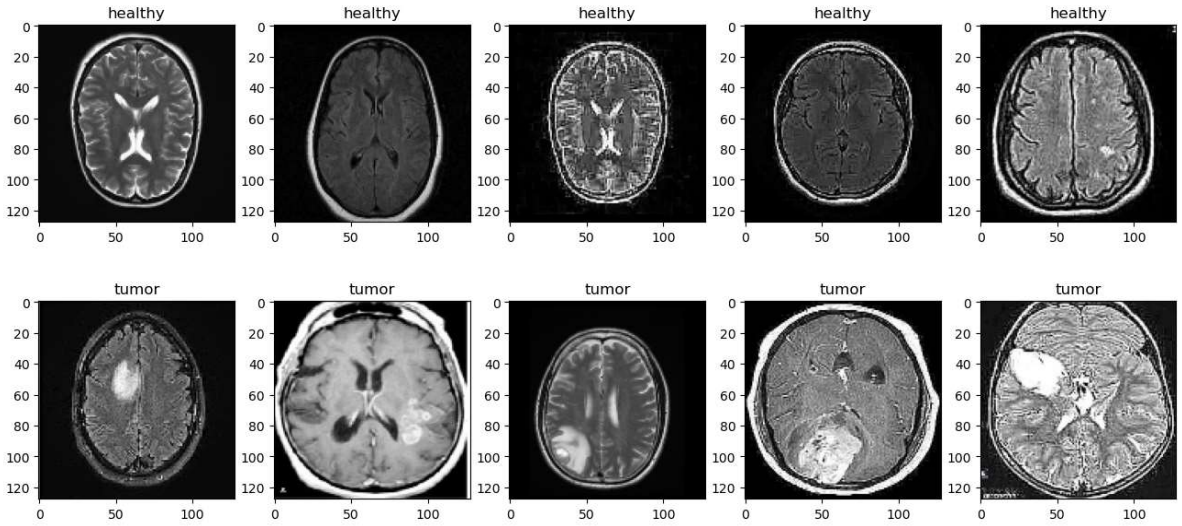
In [7]:
```
np.random.choice(10, 5, replace=False)
```

Out[7]:
```
array([7, 8, 3, 0, 1])
```

Visualizing Brain MRI Images

In [8]:
```
def plot_random(healthy, tumor, num=5):
    healthy_imgs = healthy[np.random.choice(healthy.shape[0], num, replace=Fal
    tumor_imgs = tumor[np.random.choice(tumor.shape[0], num, replace=False)]

    plt.figure(figsize=(16,9))
    for i in range(num):
        plt.subplot(1, num, i+1)
        plt.title('healthy')
        plt.imshow(healthy_imgs[i])

    plt.figure(figsize=(16,9))
    for i in range(num):
        plt.subplot(1, num, i+1)
        plt.title('tumor')
        plt.imshow(tumor_imgs[i])
```

In [9]:
```
plot_random(healthy, tumor, num=5)
```



Create Torch Dataset Class

What is Pytorch's Abstract Dataset Class

In [10]:
```
class Dataset(object):
    """An abstract class representing a Dataset.

    All other datasets should subclass it. All subclasses should override
    ``__len__``, that provides the size of the dataset, and ``__getitem__``,
    supporting integer indexing in range from 0 to len(self) exclusive.
    """

    def __getitem__(self, index):
        raise NotImplementedError

    def __len__(self):
        raise NotImplementedError

    def __add__(self, other):
        return ConcatDataset([self, other])
```

Creating MRI custom dataset class

In [11]:
```
class MRI(Dataset):
    def __init__(self):

        tumor = []
        healthy = []
        # cv2 - It reads in BGR format by default
        for f in glob.iglob("./Dataset/brain_tumor_dataset/yes/*.jpg"):
            img = cv2.imread(f)
            img = cv2.resize(img,(128,128)) # I can add this later in the boot
            b, g, r = cv2.split(img)
            img = cv2.merge([r,g,b])
            img = img.reshape((img.shape[2],img.shape[0],img.shape[1])) # othe
            tumor.append(img)

        for f in glob.iglob("./Dataset/brain_tumor_dataset/no/*.jpg"):
            img = cv2.imread(f)
            img = cv2.resize(img,(128,128))
            b, g, r = cv2.split(img)
            img = cv2.merge([r,g,b])
            img = img.reshape((img.shape[2],img.shape[0],img.shape[1]))
            healthy.append(img)

        # our images
        tumor = np.array(tumor,dtype=np.float32)
        healthy = np.array(healthy,dtype=np.float32)

        # our labels
        tumor_label = np.ones(tumor.shape[0], dtype=np.float32)
        healthy_label = np.zeros(healthy.shape[0], dtype=np.float32)

        # Concatenates
        self.images = np.concatenate((tumor, healthy), axis=0)
        self.labels = np.concatenate((tumor_label, healthy_label))

    def __len__(self):
        return self.images.shape[0]

    def __getitem__(self, index):

        sample = {'image': self.images[index], 'label':self.labels[index]}

        return sample

    def normalize(self):
        self.images = self.images/255.0
```

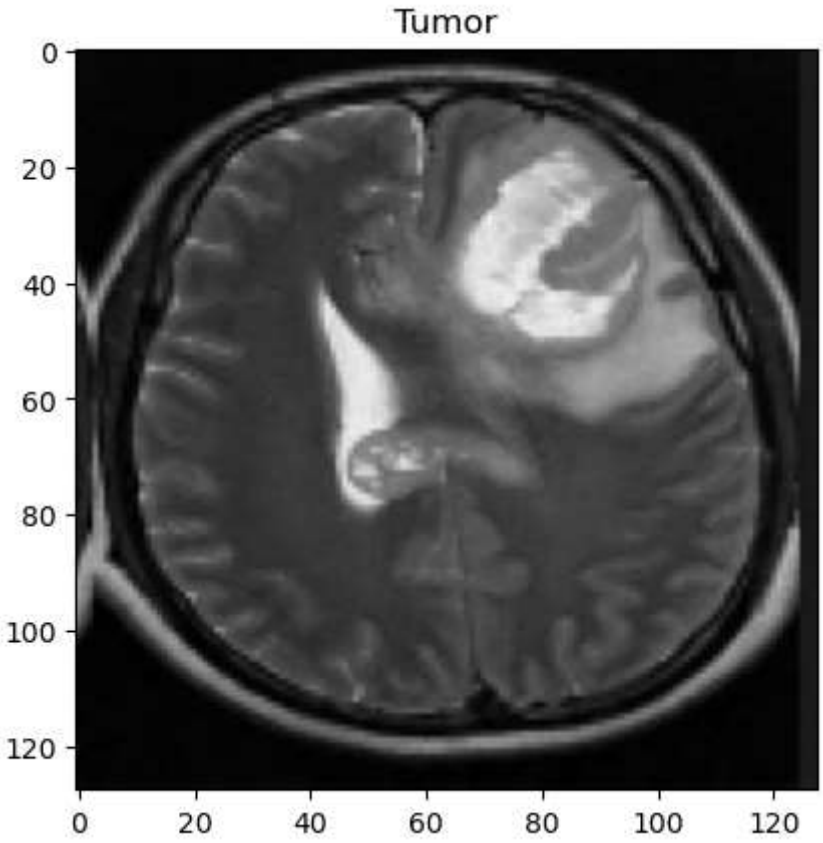
In [12]:
```
mri_dataset = MRI()
mri_dataset.normalize()
```

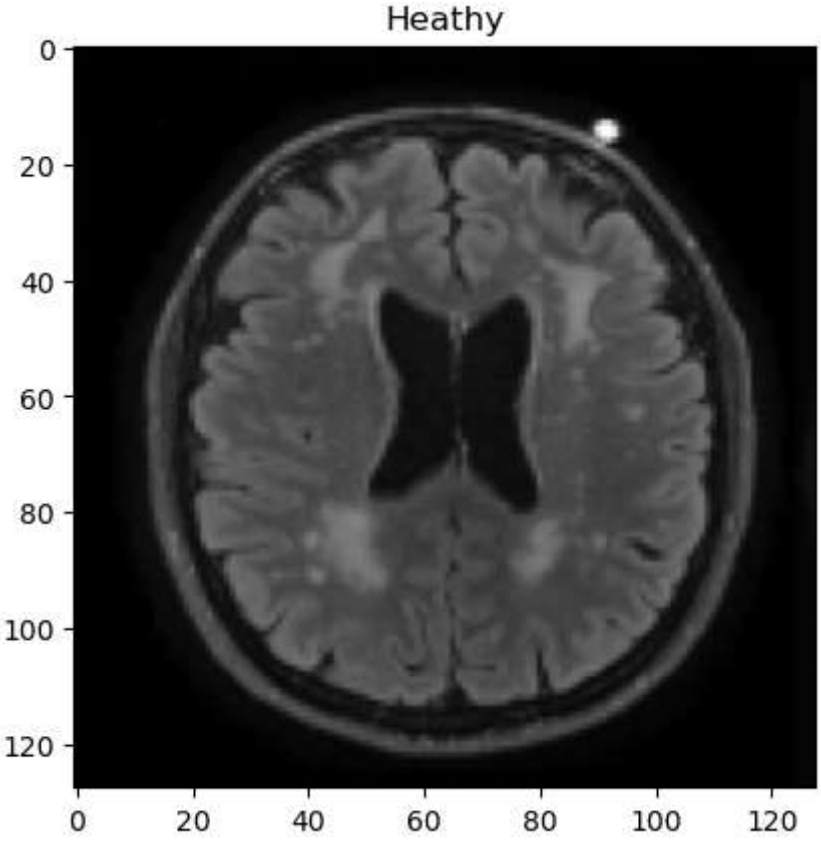
In [13]:
```
print(len(mri_dataset))
```

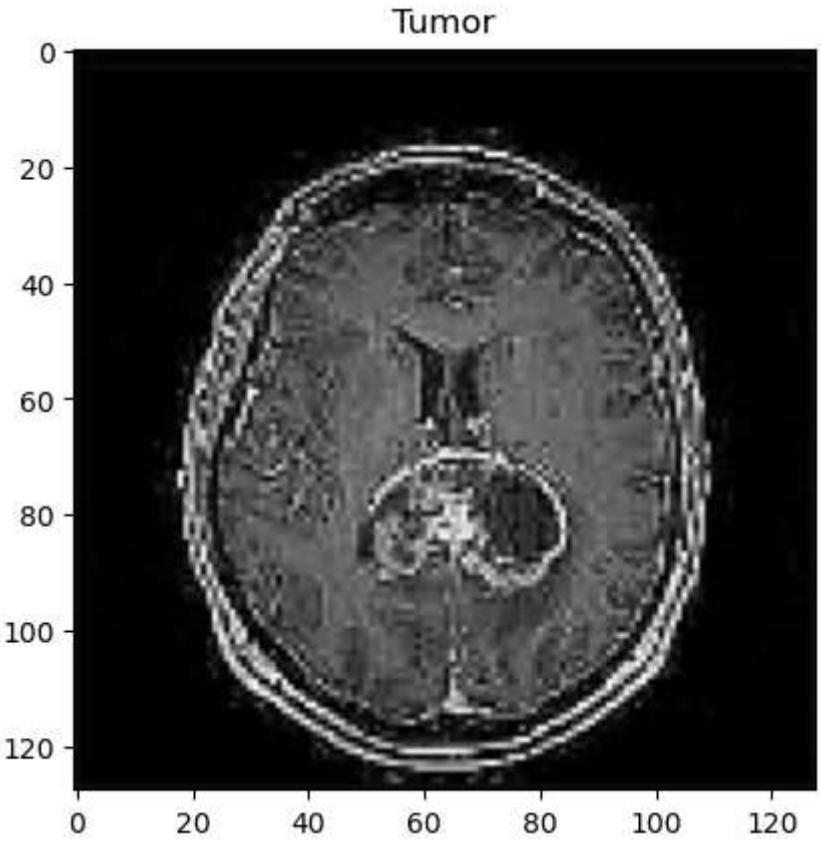
```
3000
```

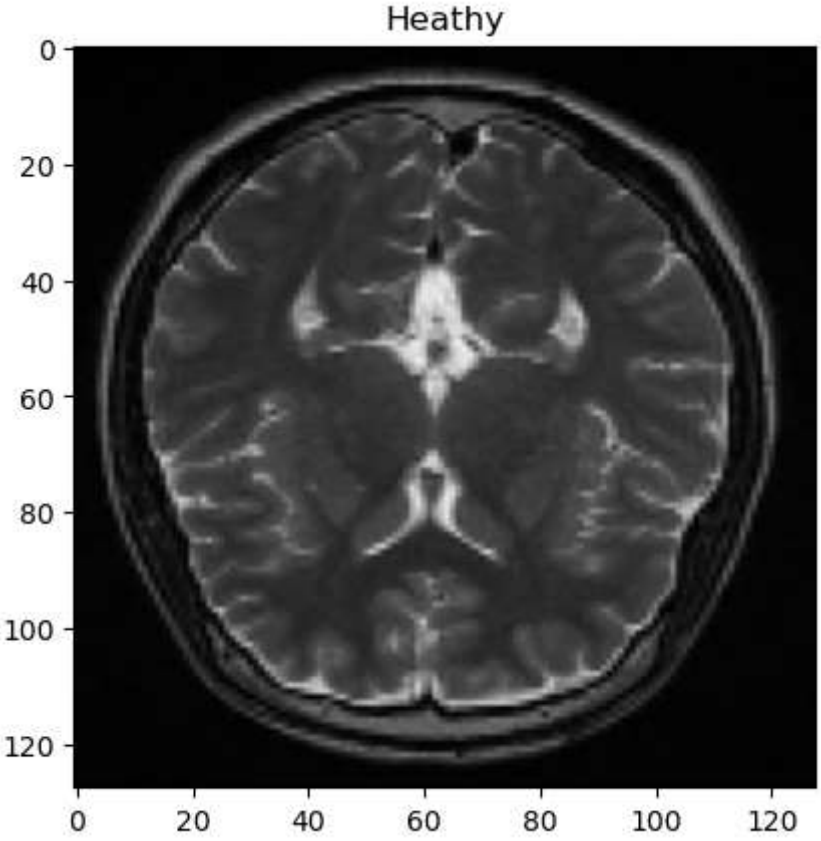
Creating a dataloader

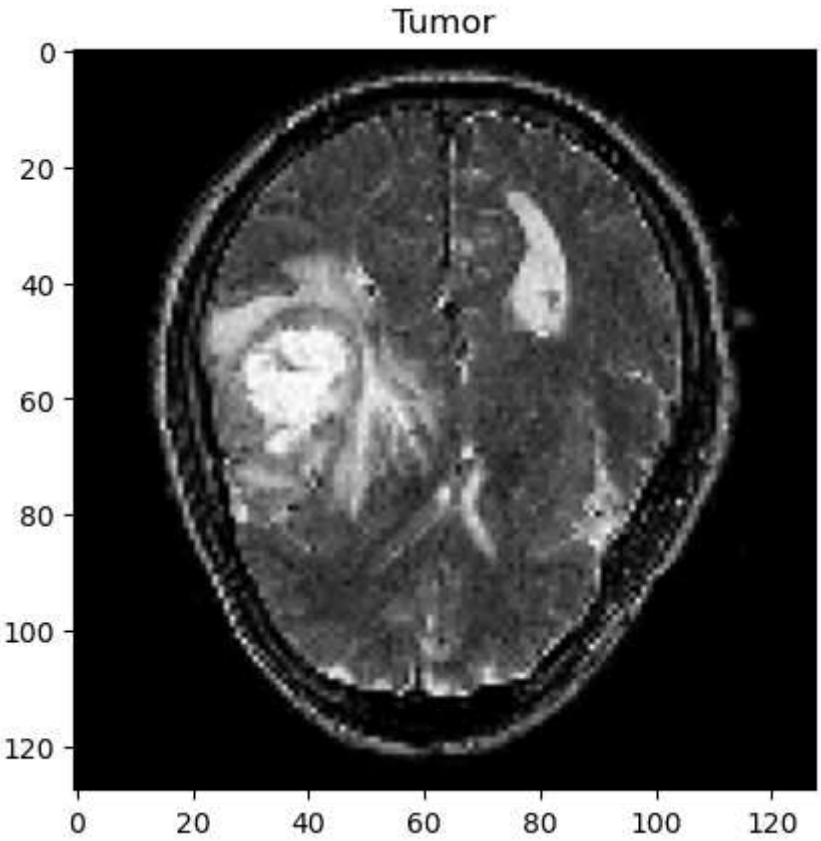
In [14]:
```
# One way of iterating
names={0:'Heathy', 1:'Tumor'}
dataloader = DataLoader(mri_dataset, shuffle=True)
for i, sample in enumerate(dataloader):
    img = sample['image'].squeeze()
    img = img.reshape((img.shape[1], img.shape[2], img.shape[0]))
    plt.title(names[sample['label'].item()])
    plt.imshow(img)
    plt.show()
    if i == 5:
        break
```

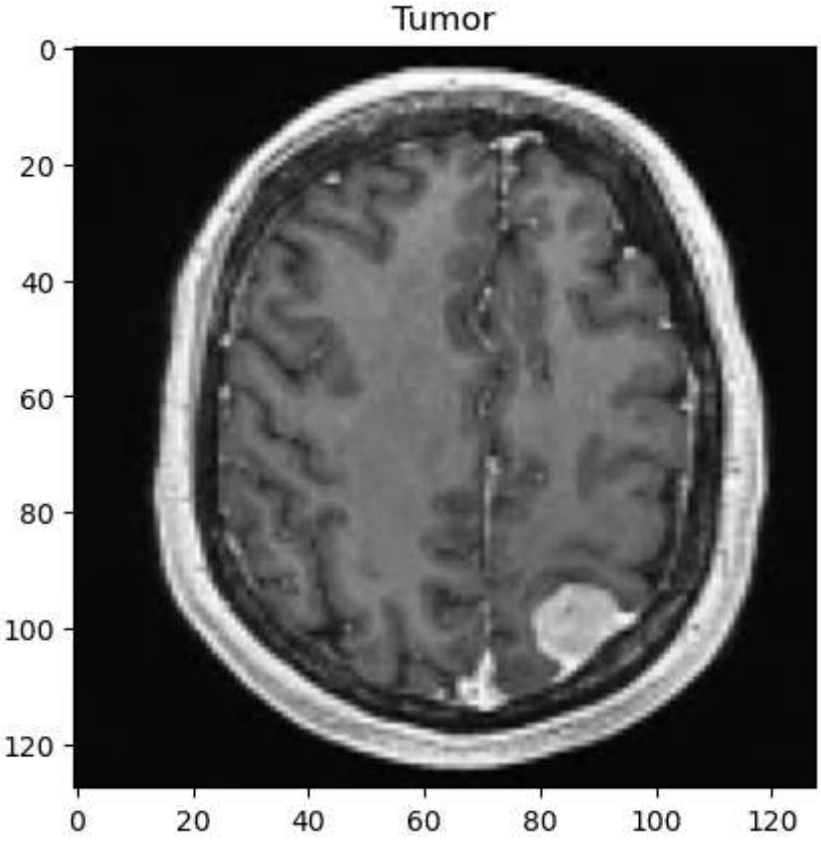



Heathy
0
20
40
60
80
100
120
0
20
40
60
80
100
120


Tumor
0
20
40
60
80
100
120
0
20
40
60
80
100
120


Heathy
0
20
40
60
80
100
120
0
20
40
60
80
100
120


Tumor
0
20
40
60
80
100
120
0
20
40
60
80
100
120



Create a Model

In [15]:
```
import torch.nn as nn
import torch.nn.functional as F

class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.cnn_model = nn.Sequential(
        nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5),
        nn.Tanh(),
        nn.AvgPool2d(kernel_size=2, stride=5),
        nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5),
        nn.Tanh(),
        nn.AvgPool2d(kernel_size=2, stride=5))

        self.fc_model = nn.Sequential(
        nn.Linear(in_features=256, out_features=120),
        nn.Tanh(),
        nn.Linear(in_features=120, out_features=84),
        nn.Tanh(),
        nn.Linear(in_features=84, out_features=1))

    def forward(self, x):
        x = self.cnn_model(x)
        x = x.view(x.size(0), -1)
        x = self.fc_model(x)
        x = F.sigmoid(x)

        return x
```

In [16]:
```
model = CNN()
```

In [17]:
```
model
```

Out[17]:
```
CNN(
  (cnn_model): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): Tanh()
    (2): AvgPool2d(kernel_size=2, stride=5, padding=0)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): Tanh()
    (5): AvgPool2d(kernel_size=2, stride=5, padding=0)
  )
  (fc_model): Sequential(
    (0): Linear(in_features=256, out_features=120, bias=True)
    (1): Tanh()
    (2): Linear(in_features=120, out_features=84, bias=True)
    (3): Tanh()
    (4): Linear(in_features=84, out_features=1, bias=True)
  )
)
```

In [18]:
```
model.cnn_model[0]
```

Out[18]:
```
Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
```

In [19]:
```
model.cnn_model[0].weight
```

Out[19]:
```
Parameter containing:
tensor([[[[ 2.9573e-02, -7.7953e-02, -7.2569e-02,  4.4003e-03, -9.6477e-02],
          [-1.0754e-01, -1.0518e-01, -1.1168e-01, -1.1511e-01, -3.1796e-02],
          [ 3.4410e-02, -2.1195e-02, -9.1147e-02,  8.9164e-02, -2.1565e-02],
          [ 5.5142e-02,  4.4064e-02, -9.9059e-02, -5.9153e-02,  2.9360e-02],
          [-1.1060e-02,  9.7618e-02, -7.1455e-02, -5.8399e-02, -3.0665e-02]],

         [[ 6.9466e-02,  1.1255e-01,  3.7784e-02, -3.9512e-03,  1.9333e-04],
          [ 3.4204e-02,  8.5983e-02, -6.8333e-02, -1.1357e-01, -4.6050e-02],
          [-9.2760e-02,  1.0067e-01, -7.5785e-02,  3.8746e-02,  1.1018e-01],
          [ 8.2843e-02,  1.1334e-01, -2.8418e-02,  8.8285e-02, -8.4272e-02],
          [-1.0325e-01,  6.8024e-02, -1.5047e-02,  5.2250e-02, -6.5932e-02]],

         [[ 2.8964e-02, -9.9149e-02,  9.7550e-02, -7.8056e-03,  8.3437e-03],
          [-1.8433e-02,  1.5478e-02,  8.0912e-02,  1.7884e-02,  5.3953e-02],
          [-5.5636e-02, -5.7376e-02, -3.8045e-02,  3.4594e-02,  1.1267e-01],
          [-5.8828e-02, -6.1325e-02,  3.8862e-02,  1.7697e-02,  2.5607e-02],
          [ 8.9775e-02, -3.2251e-02, -1.0453e-01,  3.5765e-02,  4.7769e-0
2]]],


        [[[ 4.6824e-02,  9.6761e-02,  1.1052e-01, -7.9088e-02,  5.1605e-02],
          [ 4.3547e-02, -7.8214e-02,  6.5919e-02,  6.1746e-02, -4.8187e-03],
          [ 3.4830e-02,  4.1905e-02, -7.6240e-03, -9.6970e-02,  8.5597e-02],
          [ 2.1046e-02,  1.0095e-01,  1.7831e-02,  2.2094e-02,  9.5450e-02],
          [-1.0966e-02,  1.1223e-01,  1.0846e-01, -6.7152e-02,  6.8098e-02]],

         [[-4.3959e-02,  5.2850e-02,  6.7924e-02, -8.7380e-02, -1.6340e-02],
          [-5.0203e-02,  7.2495e-02, -9.9137e-02,  5.7733e-02,  6.4979e-02],
          [ 5.8017e-02,  8.4666e-02,  3.2640e-04, -3.0949e-02, -5.2590e-03],
          [-7.9131e-03, -2.8462e-03, -1.0331e-01, -5.3214e-02,  7.4186e-02],
          [ 3.0625e-02,  2.1722e-02, -4.9065e-02, -1.0345e-02, -8.0083e-02]],

         [[ 9.7843e-02, -9.5742e-02, -3.2650e-02,  9.8257e-02,  1.0347e-01],
          [ 7.1217e-03,  1.1491e-01, -8.2454e-02, -1.7708e-03,  7.1209e-02],
          [ 1.1325e-01, -6.9386e-02,  2.5067e-02, -5.7049e-02, -7.0886e-02],
          [ 9.5073e-02, -1.1441e-01,  1.6067e-02, -9.0615e-02,  1.1318e-01],
          [ 1.0767e-01,  9.2662e-02, -1.1092e-01, -5.1462e-02, -6.9232e-0
2]]],


        [[[ 4.3675e-02,  7.0409e-02, -5.7108e-02,  9.5308e-02, -6.3752e-02],
          [-3.4416e-02, -8.2304e-02, -8.0510e-02, -1.4954e-02, -8.2848e-02],
          [ 6.7157e-02,  1.9393e-02, -4.7346e-02, -4.8752e-02, -2.2887e-02],
          [-9.5370e-03, -1.0248e-01, -8.4972e-02,  1.2317e-02, -1.0835e-01],
          [ 3.1164e-02, -4.3959e-02, -2.0227e-02, -5.4977e-02, -2.5296e-02]],

         [[ 8.6172e-02,  7.3353e-03, -8.4506e-02, -2.3526e-02, -3.3634e-02],
          [ 5.0262e-02,  5.4701e-02,  6.0132e-02, -2.3239e-02, -3.0231e-02],
          [ 2.5170e-02, -5.7150e-02, -1.0604e-01,  7.9137e-02,  3.0461e-02],
          [-7.5905e-02,  1.1517e-01,  4.5458e-02,  9.0775e-02,  1.0290e-01],
          [ 6.3468e-02,  1.0357e-01,  3.2256e-02,  9.9715e-02, -5.5617e-02]],

         [[-5.1246e-02, -1.8327e-02,  3.8643e-02, -1.1046e-01, -1.1359e-01],
          [ 1.0663e-01, -1.3192e-03,  3.3168e-02,  1.8253e-02,  2.1274e-02],
          [ 3.9974e-02,  1.3469e-02, -2.9247e-02, -8.1447e-02, -1.0268e-01],
          [ 9.9862e-02, -2.6155e-02, -8.6696e-02,  1.6818e-02,  5.4062e-02],
```

```
          [ 1.1395e-01, -8.8087e-02,  1.0843e-02, -1.0971e-01,  1.0292e-0
1]]],


       [[[ 9.6117e-02, -9.7656e-02,  5.3771e-02, -5.3250e-02,  8.5368e-02],
         [-5.7259e-02,  9.6077e-03, -1.0885e-01, -8.1661e-02, -6.4333e-02],
         [ 1.1015e-01, -6.0927e-03,  8.1008e-02,  1.1008e-01, -4.4977e-02],
         [ 9.6149e-02,  9.7024e-02, -1.1262e-01,  2.9228e-02, -7.2435e-02],
         [-6.1908e-02,  2.2144e-02,  9.0598e-02, -7.4673e-02,  9.2427e-04]],

        [[-9.2171e-02, -5.7890e-02, -9.9309e-02,  6.1022e-02,  1.0428e-01],
         [-1.6020e-02,  1.9938e-02,  7.9153e-02, -9.4814e-02,  1.0773e-01],
         [-7.2184e-02,  1.2767e-02,  8.1565e-02, -4.1497e-02,  5.9616e-03],
         [ 2.1666e-02, -9.5556e-02, -6.7259e-02, -1.0866e-01, -2.4570e-02],
         [ 1.0370e-01, -1.5935e-02,  1.8723e-02,  5.3898e-02, -9.3542e-02]],

        [[ 2.1001e-02, -2.4271e-02,  7.5056e-02, -6.4322e-02,  8.9513e-02],
         [-1.1355e-01, -4.6420e-02, -6.4884e-02,  4.4083e-02, -2.3625e-03],
         [-1.0917e-02,  3.5324e-02,  1.6030e-02, -8.0084e-02,  1.1534e-02],
         [-8.4132e-03,  9.7711e-02,  1.1488e-01, -3.3415e-02, -4.8622e-03],
         [-7.9186e-02,  5.2787e-05,  2.9857e-02,  9.1085e-02,  3.4263e-0
2]]],


       [[[-9.6539e-03, -1.9920e-02, -9.2953e-02,  1.5155e-03, -2.8950e-02],
         [ 9.2911e-02, -7.5134e-02,  8.6272e-02,  1.1350e-01, -7.3414e-02],
         [-3.7944e-04, -5.0754e-02,  1.1216e-01,  9.0224e-03,  7.8896e-02],
         [-9.2020e-02,  1.2126e-02,  8.4501e-02,  7.3692e-02,  1.2813e-02],
         [-5.8416e-02,  4.2088e-02, -9.8305e-02,  2.5183e-02,  7.2506e-02]],

        [[ 1.0057e-01,  1.1829e-02, -1.1815e-03, -9.4430e-02,  5.6940e-02],
         [ 1.0056e-01, -8.3365e-02, -4.5058e-03,  2.9671e-02, -7.3267e-02],
         [ 8.1851e-02, -1.0613e-01,  7.3324e-02,  2.2394e-02,  9.2509e-02],
         [ 1.1317e-01, -2.5566e-02, -1.9560e-03,  8.9929e-02,  7.8791e-02],
         [ 9.8753e-02,  8.8917e-02,  4.5018e-02, -5.4425e-02, -1.3913e-02]],

        [[ 3.1336e-02, -8.6038e-02, -8.6509e-02,  3.6678e-02,  1.8226e-02],
         [ 1.0059e-02,  8.9319e-02,  1.0351e-01, -6.5478e-02, -7.4004e-02],
         [-6.6322e-02, -1.0478e-01, -2.5791e-03, -5.1986e-02, -5.0744e-02],
         [-9.7492e-02, -1.0452e-01, -9.0166e-02, -1.0140e-01,  4.0922e-02],
         [ 6.9777e-02,  1.1432e-02,  1.0152e-01, -4.5663e-02,  7.1806e-0
2]]],


       [[[-6.7215e-02,  3.4098e-02, -9.1295e-02, -6.4085e-02,  1.7730e-02],
         [-8.0053e-02, -7.9467e-02, -1.0013e-01, -7.6726e-02, -5.6479e-02],
         [-5.3848e-02, -3.0305e-02, -1.4492e-02, -5.3070e-02, -1.0176e-01],
         [ 9.4795e-02, -1.0800e-01,  6.8217e-02, -3.0131e-02, -1.2711e-02],
         [ 6.7909e-02, -5.7958e-02, -1.2159e-02,  8.4160e-02, -4.4881e-02]],

        [[ 4.2924e-02, -8.3458e-02, -4.5111e-02, -4.0442e-02,  2.2698e-03],
         [ 8.9134e-02,  9.2326e-02,  7.2877e-02,  2.9795e-03, -5.3399e-02],
         [ 7.1194e-02,  2.0444e-02, -3.9675e-02, -6.4422e-02,  1.0531e-01],
         [ 9.4779e-02, -6.9203e-02,  7.7797e-02,  8.1367e-02,  1.1342e-01],
         [-1.0225e-01,  6.8785e-02,  3.8028e-02, -4.2235e-02, -6.6562e-02]],

        [[-1.0687e-01, -1.1465e-01, -1.2925e-03,  4.5073e-02, -3.1984e-03],
```

```
           [ 1.0465e-01,  2.4855e-02, -9.1761e-02,  5.1239e-03, -2.1921e-03],
           [-9.5639e-02, -5.3934e-02, -1.3828e-02, -9.7026e-02,  4.9855e-02],
           [-9.7986e-02, -5.0539e-02,  6.5128e-03,  9.4797e-02,  5.2262e-02],
           [-1.1808e-02,  7.9359e-02, -4.4185e-02, -8.5536e-02, -5.8339e-0
2]]]],
       requires_grad=True)
```

In [20]:
```
model.fc_model[0]
```

Out[20]:
```
Linear(in_features=256, out_features=120, bias=True)
```

In [21]:
```
model.fc_model[0].weight
```

Out[21]:
```
Parameter containing:
tensor([[-4.3541e-02,  5.1187e-02, -2.2779e-03,  ..., -4.8563e-02,
          3.0976e-02, -2.5181e-03],
        [ 1.0186e-02,  5.6233e-02,  5.3120e-03,  ...,  4.5479e-02,
         -2.2770e-02, -1.9618e-04],
        [ 5.7610e-03, -2.6551e-02, -4.5352e-02,  ..., -1.4953e-02,
         -4.5227e-02,  2.8636e-02],
        ...,
        [-4.8828e-02,  2.1131e-02, -1.5949e-02,  ...,  2.9193e-02,
         -3.2599e-02, -1.4949e-02],
        [-6.0958e-02, -3.6807e-02, -2.2194e-02,  ..., -4.5673e-02,
         -2.6616e-02, -3.6918e-05],
        [ 2.2497e-02,  2.5538e-02, -1.8552e-02,  ...,  3.9344e-02,
          2.5112e-02,  2.4981e-02]], requires_grad=True)
```

torch.tensor vs. torch.cuda.tensor

here key difference is just that torch.Tensor occupies CPU memory while torch.cuda.Tensor occupies GPU memory. Of course operations on a CPU Tensor are computed with CPU while operations for the GPU / CUDA Tensor are computed on GPU.

In [22]:
```
# device will be 'cuda' if a GPU is available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# creating a CPU tensor
cpu_tensor = torch.rand(10).to(device)
# moving same tensor to GPU
gpu_tensor = cpu_tensor.to(device)

print(cpu_tensor, cpu_tensor.dtype, type(cpu_tensor), cpu_tensor.type())
print(gpu_tensor, gpu_tensor.dtype, type(gpu_tensor), gpu_tensor.type())

print(cpu_tensor*gpu_tensor)
```

```
tensor([0.3812, 0.1533, 0.7614, 0.0437, 0.6789, 0.6571, 0.1940, 0.1251, 0.209
7,
        0.1060]) torch.float32 <class 'torch.Tensor'> torch.FloatTensor
tensor([0.3812, 0.1533, 0.7614, 0.0437, 0.6789, 0.6571, 0.1940, 0.1251, 0.209
7,
        0.1060]) torch.float32 <class 'torch.Tensor'> torch.FloatTensor
tensor([0.1453, 0.0235, 0.5797, 0.0019, 0.4609, 0.4317, 0.0376, 0.0157, 0.044
0,
        0.0112])
```

Evaluate a New-Born Neural Network!

In [23]:
```
mri_dataset = MRI()
mri_dataset.normalize()
device = torch.device('cpu:0')
model = CNN().to(device)
```

In [24]:
```
dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=False)
```

In [25]:
```
model.eval()
outputs = []
y_true = []
with torch.no_grad():
    for D in dataloader:
        image = D['image'].to(device)
        label = D['label'].to(device)

        y_hat = model(image)

        outputs.append(y_hat.cpu().detach().numpy())
        y_true.append(label.cpu().detach().numpy())
```

In [26]:
```
outputs = np.concatenate( outputs, axis=0 ).squeeze()
y_true = np.concatenate( y_true, axis=0 ).squeeze()
```

In [27]:
```
def threshold(scores,threshold=0.50, minimum=0, maximum = 1.0):
    x = np.array(list(scores))
    x[x >= threshold] = maximum
    x[x < threshold] = minimum
    return x
```

In [28]:
```
accuracy_score(y_true, threshold(outputs))
```

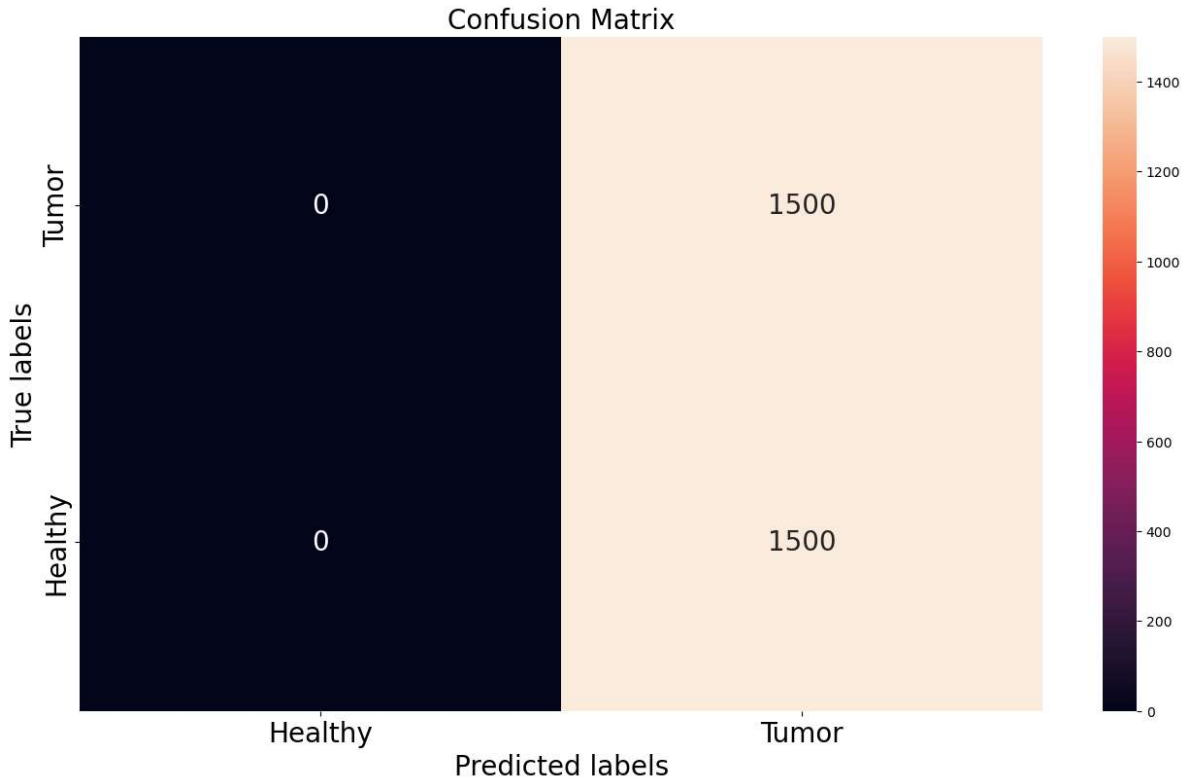
Out[28]:
```
0.5
```

In [29]:
```
from sklearn.metrics import classification_report
print (classification_report(y_true, threshold(outputs)))
```

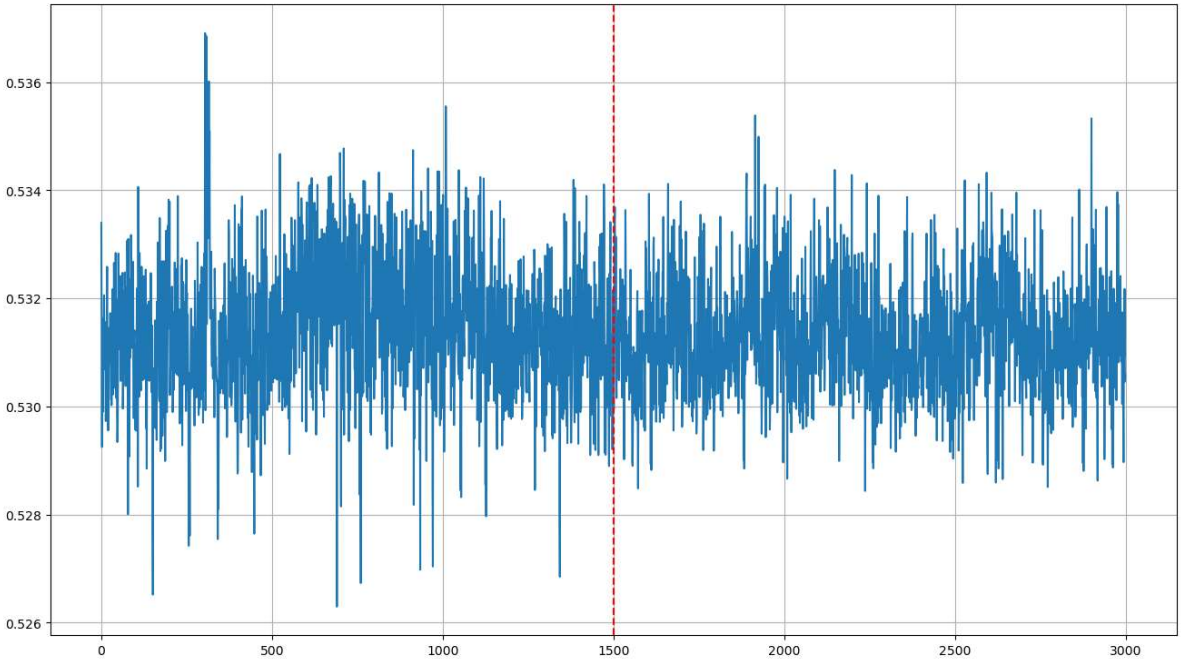
```
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00      1500
         1.0       0.50      1.00      0.67      1500

    accuracy                           0.50      3000
   macro avg       0.25      0.50      0.33      3000
weighted avg       0.25      0.50      0.33      3000


C:\Users\mshiv\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\mshiv\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\mshiv\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [30]:
```
# a better confusion matrix
import seaborn as sns

plt.figure(figsize=(16,9))
cm = confusion_matrix(y_true, threshold(outputs))
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax, annot_kws={"size": 20})

# labels, title and ticks
ax.set_xlabel('Predicted labels', fontsize=20)
ax.set_ylabel('True labels', fontsize=20)
ax.set_title('Confusion Matrix', fontsize=20)
ax.xaxis.set_ticklabels(['Healthy','Tumor'], fontsize=20)
ax.yaxis.set_ticklabels(['Tumor','Healthy'], fontsize=20)



print('True Positives(TP) = ', cm[0,0])
print('True Negatives(TN) = ', cm[1,1])
print('False Positives(FP) = ', cm[0,1])
print('False Negatives(FN) = ', cm[1,0])
```

```
True Positives(TP) =  0
True Negatives(TN) =  1500
False Positives(FP) =  1500
False Negatives(FN) =  0
```



In [31]:
```
plt.figure(figsize=(16,9))
plt.plot(outputs)
plt.axvline(x=len(tumor), color='r', linestyle='--')
plt.grid()
```



Train the dumb model

In [32]:
```
eta = 0.0001
EPOCH = 400
optimizer = torch.optim.Adam(model.parameters(), lr=eta)
dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=True)
model.train()
```

Out[32]:
```
CNN(
  (cnn_model): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): Tanh()
    (2): AvgPool2d(kernel_size=2, stride=5, padding=0)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): Tanh()
    (5): AvgPool2d(kernel_size=2, stride=5, padding=0)
  )
  (fc_model): Sequential(
    (0): Linear(in_features=256, out_features=120, bias=True)
    (1): Tanh()
    (2): Linear(in_features=120, out_features=84, bias=True)
    (3): Tanh()
    (4): Linear(in_features=84, out_features=1, bias=True)
  )
)
```

In [38]:
```
import torch
import torch.nn as nn
import numpy as np

# Assuming you have val_dataset and val_dataloader defined similarly to train

best_loss = float('inf')
early_stop_patience = 5
no_improvement_counter = 0

for epoch in range(1, EPOCH + 1):
    losses = []
    for D in dataloader:
        optimizer.zero_grad()
        data = D['image'].to(device)
        label = D['label'].to(device)
        y_hat = model(data)
        # define loss function
        error = nn.BCELoss()
        loss = torch.sum(error(y_hat.squeeze(), label))
        loss.backward()
        optimizer.step()
        losses.append(loss.item())

    mean_loss = np.mean(losses)
    print('Train Epoch: {}\tLoss: {:.6f}'.format(epoch, mean_loss))

    # Validation
    model.eval()
    with torch.no_grad():
        val_losses = []
        for val_D in dataloader:
            val_data = val_D['image'].to(device)
            val_label = val_D['label'].to(device)
            val_y_hat = model(val_data)
            val_loss = torch.sum(error(val_y_hat.squeeze(), val_label))
            val_losses.append(val_loss.item())

        mean_val_loss = np.mean(val_losses)
        print('Validation Loss: {:.6f}'.format(mean_val_loss))

        if mean_val_loss < best_loss:
            best_loss = mean_val_loss
            no_improvement_counter = 0
        else:
            no_improvement_counter += 1

        if no_improvement_counter >= early_stop_patience:
            print("Early stopping triggered!")
            break


    model.train()  # set the model back to training mode
```

```
Train Epoch: 1  Loss: 0.371314
Validation Loss: 0.364506
Train Epoch: 2  Loss: 0.365287
Validation Loss: 0.366877
Train Epoch: 3  Loss: 0.364973
Validation Loss: 0.362095
Train Epoch: 4  Loss: 0.356691
Validation Loss: 0.351124
Train Epoch: 5  Loss: 0.351531
Validation Loss: 0.343149
Train Epoch: 6  Loss: 0.346325
Validation Loss: 0.347253
Train Epoch: 7  Loss: 0.342701
Validation Loss: 0.331864
Train Epoch: 8  Loss: 0.339631
Validation Loss: 0.331229
Train Epoch: 9  Loss: 0.329936
Validation Loss: 0.325602
Train Epoch: 10 Loss: 0.325069
Validation Loss: 0.315700
Train Epoch: 11 Loss: 0.320846
Validation Loss: 0.309658
Train Epoch: 12 Loss: 0.317288
Validation Loss: 0.308287
Train Epoch: 13 Loss: 0.310510
Validation Loss: 0.298363
Train Epoch: 14 Loss: 0.301068
Validation Loss: 0.294377
Train Epoch: 15 Loss: 0.297829
Validation Loss: 0.287165
Train Epoch: 16 Loss: 0.293207
Validation Loss: 0.281544
Train Epoch: 17 Loss: 0.285281
Validation Loss: 0.280968
Train Epoch: 18 Loss: 0.279405
Validation Loss: 0.267547
Train Epoch: 19 Loss: 0.273755
Validation Loss: 0.273176
Train Epoch: 20 Loss: 0.266458
Validation Loss: 0.258568
Train Epoch: 21 Loss: 0.263317
Validation Loss: 0.252046
Train Epoch: 22 Loss: 0.252888
Validation Loss: 0.245003
Train Epoch: 23 Loss: 0.252852
Validation Loss: 0.239368
Train Epoch: 24 Loss: 0.246544
Validation Loss: 0.234446
Train Epoch: 25 Loss: 0.237092
Validation Loss: 0.231387
Train Epoch: 26 Loss: 0.236868
Validation Loss: 0.223174
Train Epoch: 27 Loss: 0.229557
Validation Loss: 0.217223
Train Epoch: 28 Loss: 0.228204
Validation Loss: 0.212161
Train Epoch: 29 Loss: 0.219970
```

```
Validation Loss: 0.253930
Train Epoch: 30 Loss: 0.219885
Validation Loss: 0.203411
Train Epoch: 31 Loss: 0.218115
Validation Loss: 0.214889
Train Epoch: 32 Loss: 0.206241
Validation Loss: 0.196950
Train Epoch: 33 Loss: 0.204409
Validation Loss: 0.192167
Train Epoch: 34 Loss: 0.199575
Validation Loss: 0.188962
Train Epoch: 35 Loss: 0.195151
Validation Loss: 0.187890
Train Epoch: 36 Loss: 0.187662
Validation Loss: 0.179525
Train Epoch: 37 Loss: 0.182352
Validation Loss: 0.173479
Train Epoch: 38 Loss: 0.183126
Validation Loss: 0.182675
Train Epoch: 39 Loss: 0.178712
Validation Loss: 0.169690
Train Epoch: 40 Loss: 0.175609
Validation Loss: 0.163268
Train Epoch: 41 Loss: 0.170221
Validation Loss: 0.158306
Train Epoch: 42 Loss: 0.166270
Validation Loss: 0.162532
Train Epoch: 43 Loss: 0.167795
Validation Loss: 0.160746
Train Epoch: 44 Loss: 0.159544
Validation Loss: 0.147212
Train Epoch: 45 Loss: 0.155980
Validation Loss: 0.149898
Train Epoch: 46 Loss: 0.155839
Validation Loss: 0.144049
Train Epoch: 47 Loss: 0.146655
Validation Loss: 0.138533
Train Epoch: 48 Loss: 0.145749
Validation Loss: 0.143828
Train Epoch: 49 Loss: 0.140844
Validation Loss: 0.131588
Train Epoch: 50 Loss: 0.139097
Validation Loss: 0.159000
Train Epoch: 51 Loss: 0.136981
Validation Loss: 0.157538
Train Epoch: 52 Loss: 0.138373
Validation Loss: 0.126641
Train Epoch: 53 Loss: 0.129575
Validation Loss: 0.124795
Train Epoch: 54 Loss: 0.129902
Validation Loss: 0.116210
Train Epoch: 55 Loss: 0.122326
Validation Loss: 0.114162
Train Epoch: 56 Loss: 0.118222
Validation Loss: 0.109578
Train Epoch: 57 Loss: 0.115492
Validation Loss: 0.120504
```

```
Train Epoch: 58 Loss: 0.122434
Validation Loss: 0.104768
Train Epoch: 59 Loss: 0.111364
Validation Loss: 0.102303
Train Epoch: 60 Loss: 0.108339
Validation Loss: 0.106927
Train Epoch: 61 Loss: 0.107134
Validation Loss: 0.097431
Train Epoch: 62 Loss: 0.104543
Validation Loss: 0.102378
Train Epoch: 63 Loss: 0.101562
Validation Loss: 0.091891
Train Epoch: 64 Loss: 0.097987
Validation Loss: 0.087914
Train Epoch: 65 Loss: 0.096588
Validation Loss: 0.086160
Train Epoch: 66 Loss: 0.093823
Validation Loss: 0.084309
Train Epoch: 67 Loss: 0.095861
Validation Loss: 0.098321
Train Epoch: 68 Loss: 0.086863
Validation Loss: 0.080523
Train Epoch: 69 Loss: 0.087345
Validation Loss: 0.077219
Train Epoch: 70 Loss: 0.084509
Validation Loss: 0.076069
Train Epoch: 71 Loss: 0.081470
Validation Loss: 0.088101
Train Epoch: 72 Loss: 0.083440
Validation Loss: 0.072480
Train Epoch: 73 Loss: 0.078224
Validation Loss: 0.069079
Train Epoch: 74 Loss: 0.077807
Validation Loss: 0.068162
Train Epoch: 75 Loss: 0.075728
Validation Loss: 0.064388
Train Epoch: 76 Loss: 0.071005
Validation Loss: 0.071907
Train Epoch: 77 Loss: 0.072976
Validation Loss: 0.071945
Train Epoch: 78 Loss: 0.067153
Validation Loss: 0.061677
Train Epoch: 79 Loss: 0.069135
Validation Loss: 0.056639
Train Epoch: 80 Loss: 0.065645
Validation Loss: 0.056219
Train Epoch: 81 Loss: 0.067576
Validation Loss: 0.068280
Train Epoch: 82 Loss: 0.064515
Validation Loss: 0.051278
Train Epoch: 83 Loss: 0.059782
Validation Loss: 0.056726
Train Epoch: 84 Loss: 0.057663
Validation Loss: 0.048674
Train Epoch: 85 Loss: 0.056946
Validation Loss: 0.049557
Train Epoch: 86 Loss: 0.055082
```

```
Validation Loss: 0.046508
Train Epoch: 87 Loss: 0.054972
Validation Loss: 0.046813
Train Epoch: 88 Loss: 0.052482
Validation Loss: 0.044881
Train Epoch: 89 Loss: 0.053435
Validation Loss: 0.059498
Train Epoch: 90 Loss: 0.048453
Validation Loss: 0.039260
Train Epoch: 91 Loss: 0.047066
Validation Loss: 0.037618
Train Epoch: 92 Loss: 0.046569
Validation Loss: 0.052037
Train Epoch: 93 Loss: 0.044069
Validation Loss: 0.039141
Train Epoch: 94 Loss: 0.043071
Validation Loss: 0.053592
Train Epoch: 95 Loss: 0.044885
Validation Loss: 0.034707
Train Epoch: 96 Loss: 0.047369
Validation Loss: 0.034276
Train Epoch: 97 Loss: 0.037684
Validation Loss: 0.030038
Train Epoch: 98 Loss: 0.036696
Validation Loss: 0.029229
Train Epoch: 99 Loss: 0.033572
Validation Loss: 0.027590
Train Epoch: 100        Loss: 0.033154
Validation Loss: 0.029520
Train Epoch: 101        Loss: 0.039133
Validation Loss: 0.041218
Train Epoch: 102        Loss: 0.036582
Validation Loss: 0.025757
Train Epoch: 103        Loss: 0.030897
Validation Loss: 0.026881
Train Epoch: 104        Loss: 0.028555
Validation Loss: 0.032076
Train Epoch: 105        Loss: 0.026060
Validation Loss: 0.022705
Train Epoch: 106        Loss: 0.027163
Validation Loss: 0.029610
Train Epoch: 107        Loss: 0.033486
Validation Loss: 0.021603
Train Epoch: 108        Loss: 0.027964
Validation Loss: 0.018943
Train Epoch: 109        Loss: 0.023833
Validation Loss: 0.017828
Train Epoch: 110        Loss: 0.027444
Validation Loss: 0.018300
Train Epoch: 111        Loss: 0.020969
Validation Loss: 0.016448
Train Epoch: 112        Loss: 0.020088
Validation Loss: 0.017753
Train Epoch: 113        Loss: 0.021915
Validation Loss: 0.029799
Train Epoch: 114        Loss: 0.026758
Validation Loss: 0.016892
```

```
Train Epoch: 115        Loss: 0.028019
Validation Loss: 0.015154
Train Epoch: 116        Loss: 0.018574
Validation Loss: 0.014675
Train Epoch: 117        Loss: 0.021379
Validation Loss: 0.013411
Train Epoch: 118        Loss: 0.017928
Validation Loss: 0.022600
Train Epoch: 119        Loss: 0.015064
Validation Loss: 0.014302
Train Epoch: 120        Loss: 0.016906
Validation Loss: 0.011747
Train Epoch: 121        Loss: 0.013566
Validation Loss: 0.010526
Train Epoch: 122        Loss: 0.011601
Validation Loss: 0.015587
Train Epoch: 123        Loss: 0.013163
Validation Loss: 0.013012
Train Epoch: 124        Loss: 0.011048
Validation Loss: 0.008890
Train Epoch: 125        Loss: 0.010528
Validation Loss: 0.007300
Train Epoch: 126        Loss: 0.010581
Validation Loss: 0.009653
Train Epoch: 127        Loss: 0.011005
Validation Loss: 0.006845
Train Epoch: 128        Loss: 0.008591
Validation Loss: 0.006369
Train Epoch: 129        Loss: 0.009729
Validation Loss: 0.012960
Train Epoch: 130        Loss: 0.007509
Validation Loss: 0.005546
Train Epoch: 131        Loss: 0.006990
Validation Loss: 0.006508
Train Epoch: 132        Loss: 0.006349
Validation Loss: 0.004228
Train Epoch: 133        Loss: 0.005083
Validation Loss: 0.004952
Train Epoch: 134        Loss: 0.005313
Validation Loss: 0.004851
Train Epoch: 135        Loss: 0.007551
Validation Loss: 0.007625
Train Epoch: 136        Loss: 0.006532
Validation Loss: 0.008231
Train Epoch: 137        Loss: 0.007731
Validation Loss: 0.004482
Early stopping triggered!
```

Evaluate a smart model

In [39]:
```
model.eval()
dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=False)
outputs=[]
y_true = []
with torch.no_grad():
    for D in dataloader:
        image =  D['image'].to(device)
        label = D['label'].to(device)

        y_hat = model(image)

        outputs.append(y_hat.cpu().detach().numpy())
        y_true.append(label.cpu().detach().numpy())

outputs = np.concatenate( outputs, axis=0 )
y_true = np.concatenate( y_true, axis=0 )
```

In [40]:
```
accuracy_score(y_true, threshold(outputs))
```

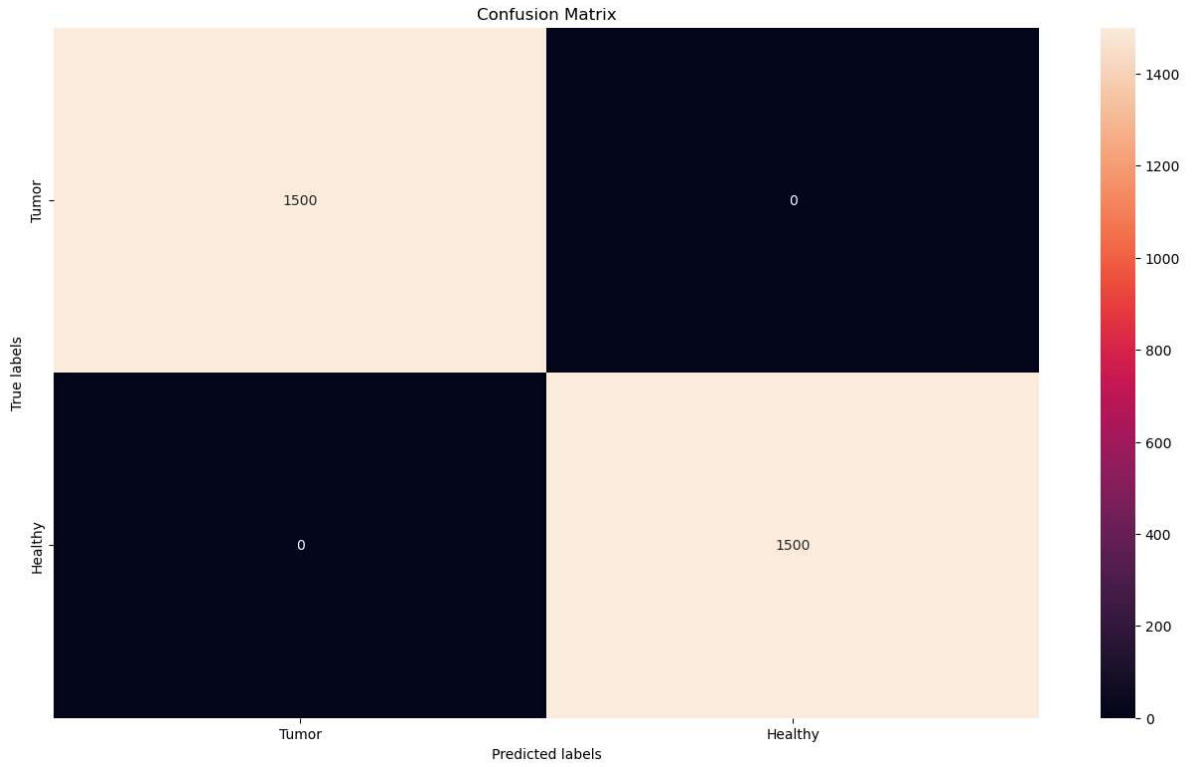
Out[40]:
```
1.0
```

In [41]:
```
print (classification_report(y_true, threshold(outputs)))
```

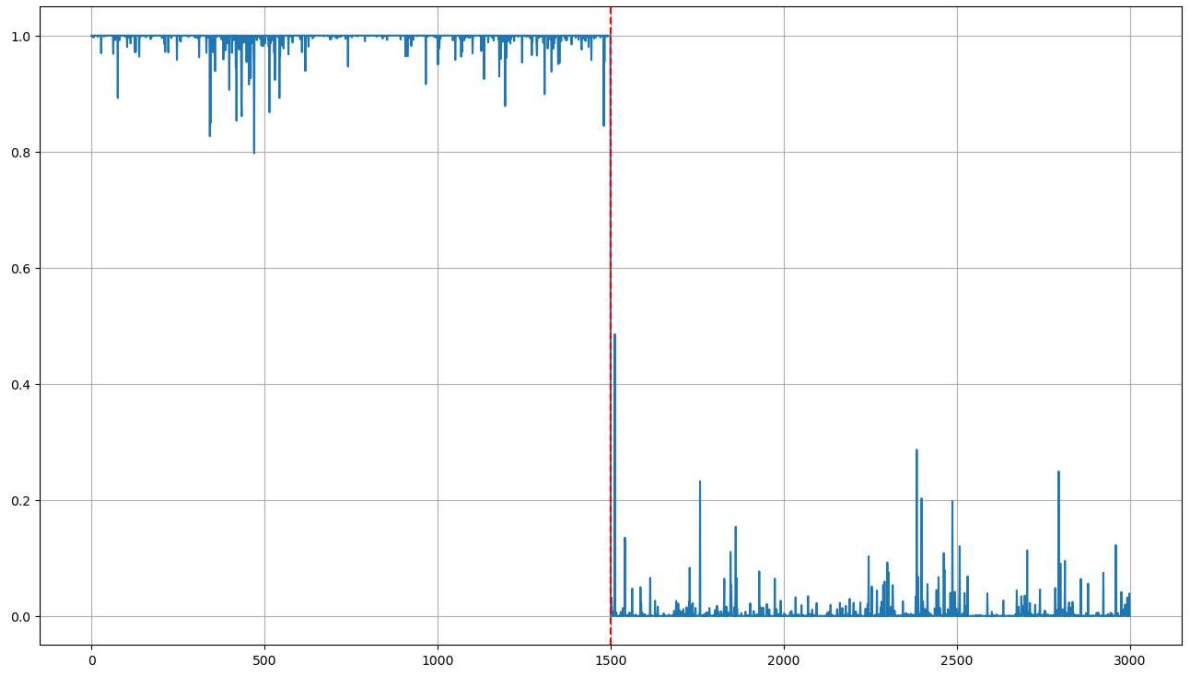
```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1500
         1.0       1.00      1.00      1.00      1500

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

In [42]:
```
cm = confusion_matrix(y_true, threshold(outputs))
plt.figure(figsize=(16,9))

ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax);  #annot=True to annotate cells, f

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Tumor','Healthy'])
ax.yaxis.set_ticklabels(['Tumor','Healthy'])


print('True Positives(TP) = ', cm[0,0])
print('True Negatives(TN) = ', cm[1,1])
print('False Positives(FP) = ', cm[0,1])
print('False Negatives(FN) = ', cm[1,0])
```

```
True Positives(TP) =  1500
True Negatives(TN) =  1500
False Positives(FP) =  0
False Negatives(FN) =  0
```



In [43]:
```
plt.figure(figsize=(16,9))
plt.plot(outputs)
plt.axvline(x=len(tumor), color='r', linestyle='--')
plt.grid()
```



Visualising the Feature Maps of the Convolutional Filters

In [44]:
```
model
```

Out[44]:
```
CNN(
  (cnn_model): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): Tanh()
    (2): AvgPool2d(kernel_size=2, stride=5, padding=0)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): Tanh()
    (5): AvgPool2d(kernel_size=2, stride=5, padding=0)
  )
  (fc_model): Sequential(
    (0): Linear(in_features=256, out_features=120, bias=True)
    (1): Tanh()
    (2): Linear(in_features=120, out_features=84, bias=True)
    (3): Tanh()
    (4): Linear(in_features=84, out_features=1, bias=True)
  )
)
```

In [45]:
```
no_of_layers = 0
conv_layers = []
```

In [46]:
```
model_children = list(model.children())
model_children
```

Out[46]:
```
[Sequential(
   (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
   (1): Tanh()
   (2): AvgPool2d(kernel_size=2, stride=5, padding=0)
   (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
   (4): Tanh()
   (5): AvgPool2d(kernel_size=2, stride=5, padding=0)
 ),
 Sequential(
   (0): Linear(in_features=256, out_features=120, bias=True)
   (1): Tanh()
   (2): Linear(in_features=120, out_features=84, bias=True)
   (3): Tanh()
   (4): Linear(in_features=84, out_features=1, bias=True)
 )]
```

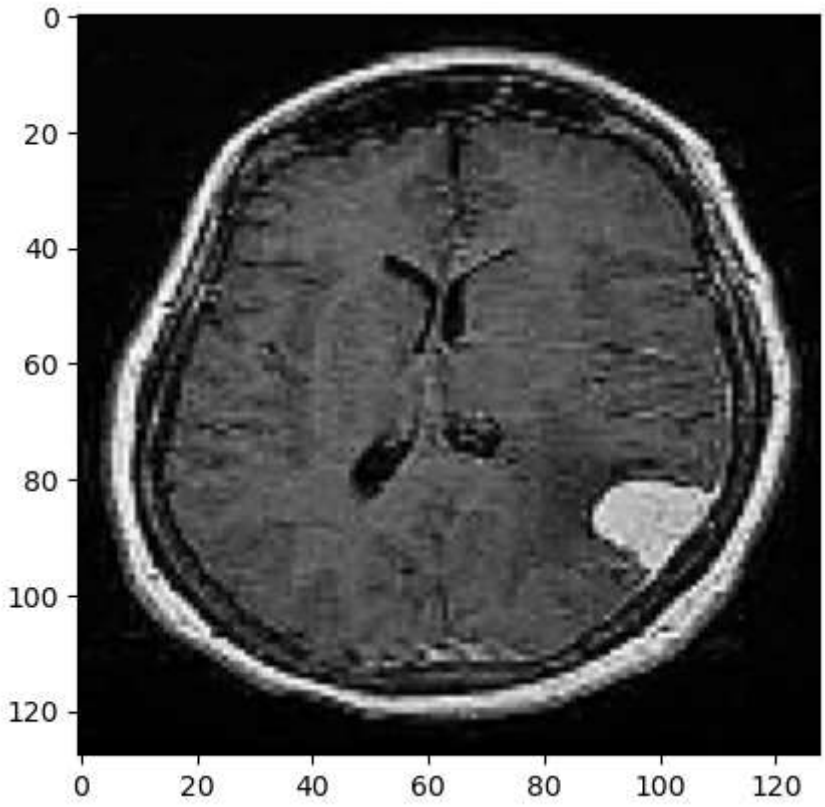
In [47]:
```
for child in model_children:
    if type(child) == nn.Sequential:
        for layer in child.children():
            if type(layer) == nn.Conv2d:
                no_of_layers += 1
                conv_layers.append(layer)
```

In [48]:
```
conv_layers
```

Out[48]:
```
[Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1)),
 Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))]
```

In [49]:
```
img = mri_dataset[100]['image']
plt.imshow(img.reshape(128,128,3))
```

Out[49]: <matplotlib.image.AxesImage at 0x1dfae851250>



In [50]:
```
img = torch.from_numpy(img).to(device)
```

In [51]:
```
img.shape
```

Out[51]: torch.Size([3, 128, 128])

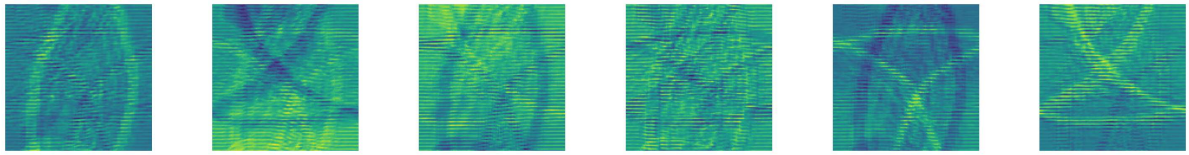
In [52]:
```
img = img.unsqueeze(0)
img.shape
```

Out[52]: torch.Size([1, 3, 128, 128])

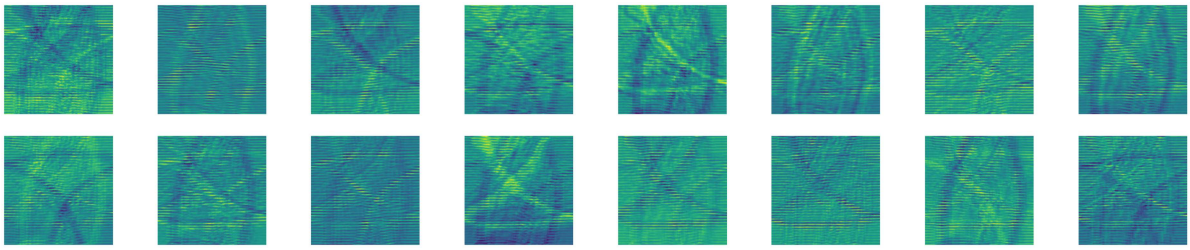
In [53]:
```
results = [conv_layers[0](img)]
for i in range(1, len(conv_layers)):
    results.append(conv_layers[i](results[-1]))
outputs = results
```

In [54]:
```
for num_layer in range(len(outputs)):
    plt.figure(figsize=(50, 10))
    layer_viz = outputs[num_layer].squeeze()
    print("Layer ",num_layer+1)
    for i, f in enumerate(layer_viz):
        plt.subplot(2, 8, i + 1)
        plt.imshow(f.detach().cpu().numpy())
        plt.axis("off")
    plt.show()
    plt.close()
```

```
Layer  1
```



```
Layer  2
```



Are We Over-fitting?

Preparing a validation set: We need to change the MRI dataset slightly!

In [55]:
```
# Import train/test split function from sklearn
from sklearn.model_selection import train_test_split
```

In [56]:
```
class MRI(Dataset):

    def __init__(self):

        # Variables to hold the Training data and Validation data
        self.X_train, self.y_train, self.X_val, self.y_val = None, None, None,

        # A variable to determine if we are interested in retrieving the train
        self.mode = 'train'

        tumor = []
        healthy = []
        # cv2 - It reads in BGR format by default
        for f in glob.iglob("./Dataset/brain_tumor_dataset/yes/*.jpg"):
            img = cv2.imread(f)
            img = cv2.resize(img,(128,128)) # I can add this later in the boot
            b, g, r = cv2.split(img)
            img = cv2.merge([r,g,b])
            img = img.reshape((img.shape[2],img.shape[0],img.shape[1])) # othe
            tumor.append(img)

        for f in glob.iglob("./Dataset/brain_tumor_dataset/no/*.jpg"):
            img = cv2.imread(f)
            img = cv2.resize(img,(128,128))
            b, g, r = cv2.split(img)
            img = cv2.merge([r,g,b])
            img = img.reshape((img.shape[2],img.shape[0],img.shape[1]))
            healthy.append(img)

        # our images
        tumor = np.array(tumor,dtype=np.float32)
        healthy = np.array(healthy,dtype=np.float32)

        # our labels
        tumor_label = np.ones(tumor.shape[0], dtype=np.float32)
        healthy_label = np.zeros(healthy.shape[0], dtype=np.float32)

        # Concatenates
        self.images = np.concatenate((tumor, healthy), axis=0)
        self.labels = np.concatenate((tumor_label, healthy_label))

    # Define a function that would separate the data into Training and Validat
    def train_val_split(self):
        self.X_train, self.X_val, self.y_train, self.y_val = \
        train_test_split(self.images, self.labels, test_size=0.20, random_stat

    def __len__(self):
        # Use self.mode to deetrmine whether train or val data is of interest
        if self.mode == 'train':
            return self.X_train.shape[0]
        elif self.mode == 'val':
            return self.X_val.shape[0]

    def __getitem__(self, idx):
        # Use self.mode to deetrmine whether train or val data is of interest
        if self.mode== 'train':
            sample = {'image': self.X_train[idx], 'label': self.y_train[idx]}
```

```
        elif self.mode== 'val':
            sample = {'image': self.X_val[idx], 'label': self.y_val[idx]}

        return sample

    def normalize(self):
        self.images = self.images/255.0
```

Are we overfitting?

In [57]:
```
mri_dataset = MRI()
mri_dataset.normalize()
mri_dataset.train_val_split()
```

In [58]:
```
train_dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=True)
val_dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=False)
```

In [59]:
```
device = torch.device("cpu:0")
model = CNN().to(device)
```

In [60]:
```
eta=0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=eta)
```

In [61]:
```
# keep track of epoch losses
epoch_train_loss = []
epoch_val_loss = []
```

In [66]:
```
import numpy as np


best_val_loss = float('inf')
patience = 20
counter = 0

for epoch in range(1, 600):
    train_losses = []
    # train for the current epoch
    model.train()
    mri_dataset.mode = 'train'
    for D in train_dataloader:
        # Train the model
        optimizer.zero_grad()
        data = D['image'].to(device)
        label = D['label'].to(device)

        y_hat = model(data)
        error = nn.BCELoss()
        loss = torch.sum(error(y_hat.squeeze(), label))
        loss.backward()
        optimizer.step()
        train_losses.append(loss.item())

    epoch_train_loss.append(np.mean(train_losses))

    val_losses = []
    model.eval()

    mri_dataset.mode = 'val'

    with torch.no_grad():
        for D in val_dataloader:
            data = D['image'].to(device)
            label = D['label'].to(device)
            y_hat = model(data)
            error = nn.BCELoss()
            loss = torch.sum(error(y_hat.squeeze(), label))
            val_losses.append(loss.item())

    epoch_val_loss.append(np.mean(val_losses))

    # Check for early stopping
    if np.mean(val_losses) < best_val_loss:
        best_val_loss = np.mean(val_losses)
        counter = 0
    else:
        counter += 1
        if counter >= patience:
            print(f'Early stopping at epoch {epoch}.')
            break

    print('Train Epoch: {}\tTrain Loss: {:.6f}\tVal Loss: {:.6f}'.format(epoch
                                                                         np.m
```

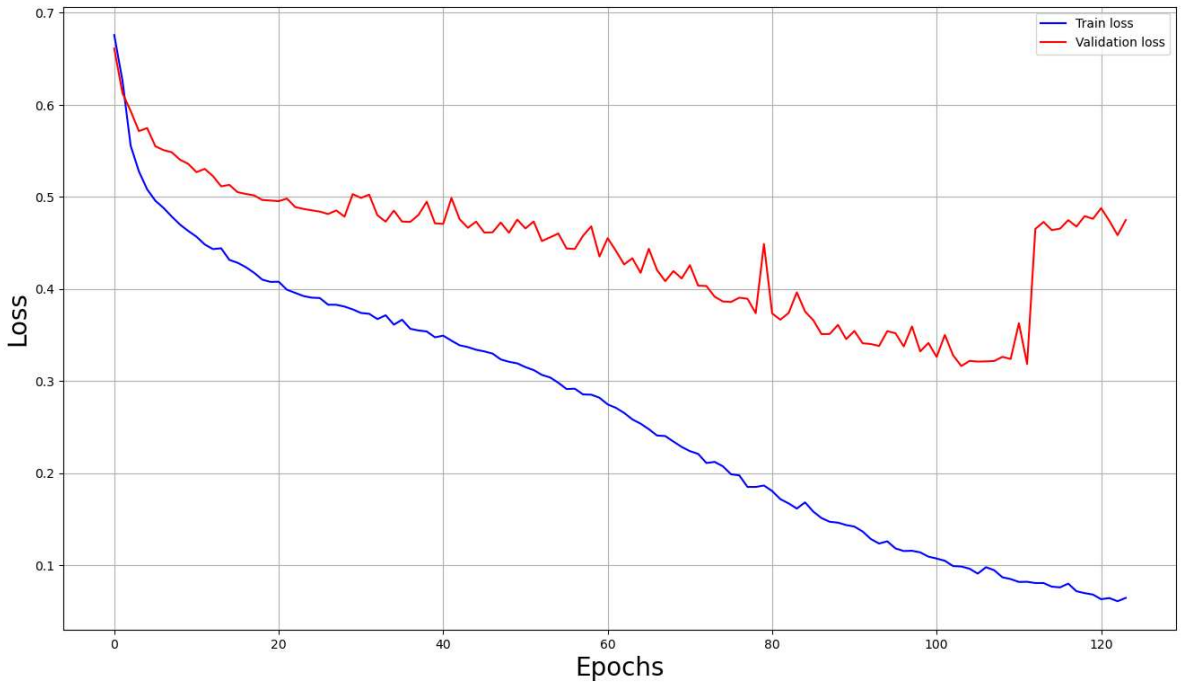
```
Train Epoch: 1  Train Loss: 0.675808    Val Loss: 0.661193
Train Epoch: 2  Train Loss: 0.626307    Val Loss: 0.612743
Train Epoch: 3  Train Loss: 0.555480    Val Loss: 0.593258
Train Epoch: 4  Train Loss: 0.527638    Val Loss: 0.571386
Train Epoch: 5  Train Loss: 0.508236    Val Loss: 0.574826
Train Epoch: 6  Train Loss: 0.495802    Val Loss: 0.555015
Train Epoch: 7  Train Loss: 0.487918    Val Loss: 0.550788
Train Epoch: 8  Train Loss: 0.478564    Val Loss: 0.548457
Train Epoch: 9  Train Loss: 0.469868    Val Loss: 0.540351
Train Epoch: 10 Train Loss: 0.462951    Val Loss: 0.535843
Train Epoch: 11 Train Loss: 0.456752    Val Loss: 0.526847
Train Epoch: 12 Train Loss: 0.448349    Val Loss: 0.530440
Train Epoch: 13 Train Loss: 0.443273    Val Loss: 0.522592
Train Epoch: 14 Train Loss: 0.444146    Val Loss: 0.511429
Train Epoch: 15 Train Loss: 0.431520    Val Loss: 0.513051
Train Epoch: 16 Train Loss: 0.428425    Val Loss: 0.505120
Train Epoch: 17 Train Loss: 0.423651    Val Loss: 0.503188
Train Epoch: 18 Train Loss: 0.417583    Val Loss: 0.501620
Train Epoch: 19 Train Loss: 0.410116    Val Loss: 0.496649
Train Epoch: 20 Train Loss: 0.407653    Val Loss: 0.496059
Train Epoch: 21 Train Loss: 0.407787    Val Loss: 0.495361
Train Epoch: 22 Train Loss: 0.399115    Val Loss: 0.498189
Train Epoch: 23 Train Loss: 0.395735    Val Loss: 0.488947
Train Epoch: 24 Train Loss: 0.392278    Val Loss: 0.486942
Train Epoch: 25 Train Loss: 0.390549    Val Loss: 0.485411
Train Epoch: 26 Train Loss: 0.390113    Val Loss: 0.483968
Train Epoch: 27 Train Loss: 0.382872    Val Loss: 0.481374
Train Epoch: 28 Train Loss: 0.382793    Val Loss: 0.485281
Train Epoch: 29 Train Loss: 0.380833    Val Loss: 0.478529
Train Epoch: 30 Train Loss: 0.377763    Val Loss: 0.502960
Train Epoch: 31 Train Loss: 0.373901    Val Loss: 0.498863
Train Epoch: 32 Train Loss: 0.372975    Val Loss: 0.502416
Train Epoch: 33 Train Loss: 0.367236    Val Loss: 0.480082
Train Epoch: 34 Train Loss: 0.371391    Val Loss: 0.473066
Train Epoch: 35 Train Loss: 0.361222    Val Loss: 0.485079
Train Epoch: 36 Train Loss: 0.366511    Val Loss: 0.473135
Train Epoch: 37 Train Loss: 0.356676    Val Loss: 0.472837
Train Epoch: 38 Train Loss: 0.354868    Val Loss: 0.480450
Train Epoch: 39 Train Loss: 0.353789    Val Loss: 0.494815
Train Epoch: 40 Train Loss: 0.347430    Val Loss: 0.471287
Train Epoch: 41 Train Loss: 0.349249    Val Loss: 0.470666
Train Epoch: 42 Train Loss: 0.343809    Val Loss: 0.498741
Train Epoch: 43 Train Loss: 0.338767    Val Loss: 0.475700
Train Epoch: 44 Train Loss: 0.336802    Val Loss: 0.466440
Train Epoch: 45 Train Loss: 0.333937    Val Loss: 0.473149
Train Epoch: 46 Train Loss: 0.332141    Val Loss: 0.461217
Train Epoch: 47 Train Loss: 0.329775    Val Loss: 0.461492
Train Epoch: 48 Train Loss: 0.323428    Val Loss: 0.472117
Train Epoch: 49 Train Loss: 0.320842    Val Loss: 0.461018
Train Epoch: 50 Train Loss: 0.319147    Val Loss: 0.475339
Train Epoch: 51 Train Loss: 0.315072    Val Loss: 0.465738
Train Epoch: 52 Train Loss: 0.311889    Val Loss: 0.473259
Train Epoch: 53 Train Loss: 0.306595    Val Loss: 0.452020
Train Epoch: 54 Train Loss: 0.303839    Val Loss: 0.456041
Train Epoch: 55 Train Loss: 0.298043    Val Loss: 0.460230
Train Epoch: 56 Train Loss: 0.291249    Val Loss: 0.443890
Train Epoch: 57 Train Loss: 0.291618    Val Loss: 0.443400
```

```
Train Epoch: 58 Train Loss: 0.285416    Val Loss: 0.457725
Train Epoch: 59 Train Loss: 0.285127    Val Loss: 0.468004
Train Epoch: 60 Train Loss: 0.281810    Val Loss: 0.435159
Train Epoch: 61 Train Loss: 0.274555    Val Loss: 0.455156
Train Epoch: 62 Train Loss: 0.270817    Val Loss: 0.441403
Train Epoch: 63 Train Loss: 0.265315    Val Loss: 0.426598
Train Epoch: 64 Train Loss: 0.258294    Val Loss: 0.433321
Train Epoch: 65 Train Loss: 0.253629    Val Loss: 0.417493
Train Epoch: 66 Train Loss: 0.247662    Val Loss: 0.443626
Train Epoch: 67 Train Loss: 0.240738    Val Loss: 0.420548
Train Epoch: 68 Train Loss: 0.240179    Val Loss: 0.408459
Train Epoch: 69 Train Loss: 0.234166    Val Loss: 0.419378
Train Epoch: 70 Train Loss: 0.228359    Val Loss: 0.411378
Train Epoch: 71 Train Loss: 0.223786    Val Loss: 0.425800
Train Epoch: 72 Train Loss: 0.220813    Val Loss: 0.403546
Train Epoch: 73 Train Loss: 0.210901    Val Loss: 0.403198
Train Epoch: 74 Train Loss: 0.212079    Val Loss: 0.391631
Train Epoch: 75 Train Loss: 0.207381    Val Loss: 0.386400
Train Epoch: 76 Train Loss: 0.198689    Val Loss: 0.385834
Train Epoch: 77 Train Loss: 0.197495    Val Loss: 0.390515
Train Epoch: 78 Train Loss: 0.184913    Val Loss: 0.389316
Train Epoch: 79 Train Loss: 0.184893    Val Loss: 0.373643
Train Epoch: 80 Train Loss: 0.186408    Val Loss: 0.448886
Train Epoch: 81 Train Loss: 0.180422    Val Loss: 0.373317
Train Epoch: 82 Train Loss: 0.171613    Val Loss: 0.366561
Train Epoch: 83 Train Loss: 0.167042    Val Loss: 0.373892
Train Epoch: 84 Train Loss: 0.161406    Val Loss: 0.396248
Train Epoch: 85 Train Loss: 0.168119    Val Loss: 0.375460
Train Epoch: 86 Train Loss: 0.158093    Val Loss: 0.366045
Train Epoch: 87 Train Loss: 0.151075    Val Loss: 0.350936
Train Epoch: 88 Train Loss: 0.147052    Val Loss: 0.351092
Train Epoch: 89 Train Loss: 0.146080    Val Loss: 0.360917
Train Epoch: 90 Train Loss: 0.143458    Val Loss: 0.345498
Train Epoch: 91 Train Loss: 0.141832    Val Loss: 0.354355
Train Epoch: 92 Train Loss: 0.136468    Val Loss: 0.340979
Train Epoch: 93 Train Loss: 0.128290    Val Loss: 0.340169
Train Epoch: 94 Train Loss: 0.123391    Val Loss: 0.338001
Train Epoch: 95 Train Loss: 0.125813    Val Loss: 0.354171
Train Epoch: 96 Train Loss: 0.118067    Val Loss: 0.351794
Train Epoch: 97 Train Loss: 0.115263    Val Loss: 0.337551
Train Epoch: 98 Train Loss: 0.115548    Val Loss: 0.359267
Train Epoch: 99 Train Loss: 0.113816    Val Loss: 0.332159
Train Epoch: 100        Train Loss: 0.109188    Val Loss: 0.341275
Train Epoch: 101        Train Loss: 0.107077    Val Loss: 0.326156
Train Epoch: 102        Train Loss: 0.104665    Val Loss: 0.350057
Train Epoch: 103        Train Loss: 0.098965    Val Loss: 0.327892
Train Epoch: 104        Train Loss: 0.098433    Val Loss: 0.316212
Train Epoch: 105        Train Loss: 0.096057    Val Loss: 0.321811
Train Epoch: 106        Train Loss: 0.090757    Val Loss: 0.321066
Train Epoch: 107        Train Loss: 0.097689    Val Loss: 0.321282
Train Epoch: 108        Train Loss: 0.094384    Val Loss: 0.321705
Train Epoch: 109        Train Loss: 0.086730    Val Loss: 0.326242
Train Epoch: 110        Train Loss: 0.084809    Val Loss: 0.323878
Train Epoch: 111        Train Loss: 0.081688    Val Loss: 0.362790
Train Epoch: 112        Train Loss: 0.081872    Val Loss: 0.318432
Train Epoch: 113        Train Loss: 0.080487    Val Loss: 0.465179
Train Epoch: 114        Train Loss: 0.080488    Val Loss: 0.472751
```

```
Train Epoch: 115        Train Loss: 0.076560    Val Loss: 0.463891
Train Epoch: 116        Train Loss: 0.075795    Val Loss: 0.465500
Train Epoch: 117        Train Loss: 0.079839    Val Loss: 0.474723
Train Epoch: 118        Train Loss: 0.071602    Val Loss: 0.467803
Train Epoch: 119        Train Loss: 0.069528    Val Loss: 0.479115
Train Epoch: 120        Train Loss: 0.067979    Val Loss: 0.476181
Train Epoch: 121        Train Loss: 0.062943    Val Loss: 0.487732
Train Epoch: 122        Train Loss: 0.064157    Val Loss: 0.473955
Train Epoch: 123        Train Loss: 0.060732    Val Loss: 0.458447
Early stopping at epoch 124.
```

In [67]:
```
plt.figure(figsize=(16,9))
plt.plot(epoch_train_loss, c='b', label='Train loss')
plt.plot(epoch_val_loss, c='r', label = 'Validation loss')
plt.legend()
plt.grid()
plt.xlabel('Epochs', fontsize=20)
plt.ylabel('Loss', fontsize=20)
```

Out[67]:
```
Text(0, 0.5, 'Loss')
```



In [ ]:
```
```