Finding the MRI brain tumor detection dataset

Let's find the dataset in this link: https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection (https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection)

Import packages

In [20]:
```
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader, ConcatDataset
import glob2
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score
import random
import cv2
import sys
```

Reading the Images

In [21]:
```
tumor = []
healthy = []
for f in glob2.iglob("./data/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 glob2.iglob("./data/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 [4]:
```
healthy = np.array(healthy)
tumor = np.array(tumor)
```

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

Out[5]:
```
(91, 128, 128, 3)
```

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

Out[6]:
```
(154, 128, 128, 3)
```

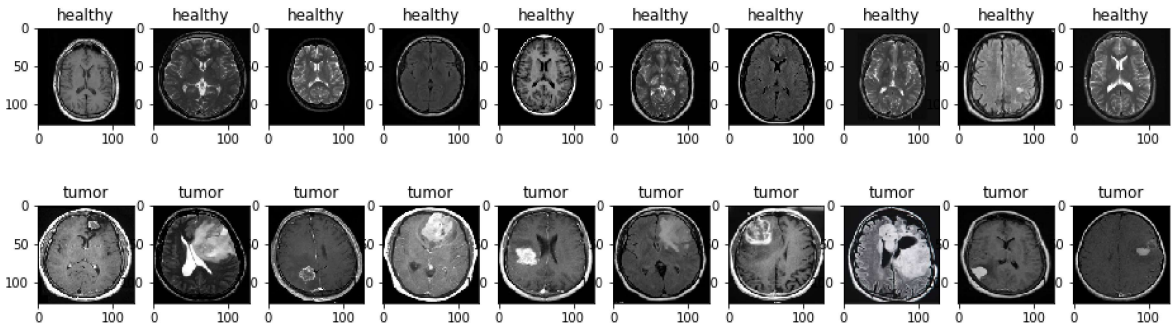
In [7]:
```
np.random.choice(10, 5, replace=False)
```

Out[7]:
```
array([5, 6, 7, 9, 8])
```

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=10)
```



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 cutom dataset class

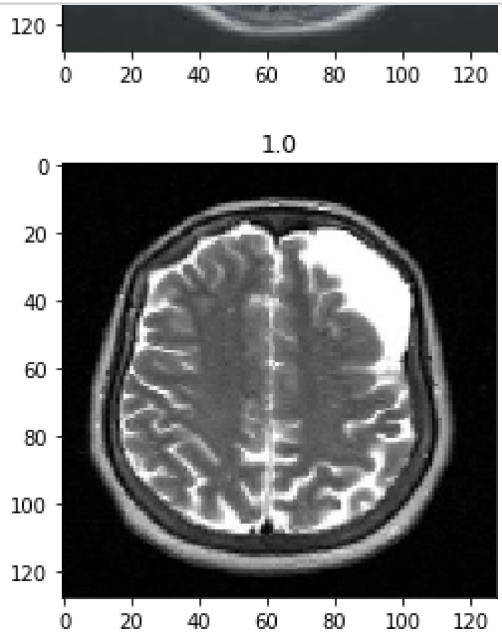
In [22]:
```
class MRI(Dataset):
    def __init__(self):

        tumor = []
        healthy = []
        # cv2 - It reads in BGR format by default
        for f in glob2.iglob("./data/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 glob2.iglob("./data/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 [17]:
```
mri_dataset = MRI()
mri_dataset.normalize()
```

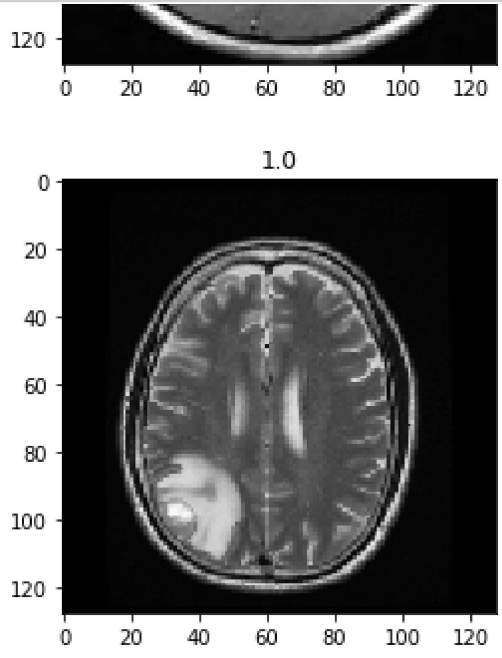
Creating a dataloader

The World Without dataloaders

In [23]:
```
index = list(range(len(mri)))
random.shuffle(index)
for idx in index:
    sample = mri[idx]
    img = sample['image']
    label = sample['label']
    img = img.reshape(img.shape[1], img.shape[2], img.shape[0])
    plt.title(label)
    plt.imshow(img)
    plt.show()
```



In [29]:
```
it = iter(mri)
for i in range(10):
    sample = next(it)
    img = sample['image']
    label = sample['label']
    img = img.reshape(img.shape[1], img.shape[2], img.shape[0])
    plt.title(label)
    plt.imshow(img)
    plt.show()
```



The World with data loaders

In [30]:
```
dataloader = DataLoader (mri_dataset, batch_size=10, shuffle=True)
```

In [32]:
```
for sample in dataloader:
    img = sample['image']
    print(img.shape)
    sys.exit()
    #img img.reshape(img.shape[1], img.shape[2], img.shape[0])
    #plt.imshow(img)
    #plt.show()
```

```
torch.Size([10, 3, 128, 128])

An exception has occurred, use %tb to see the full traceback.

SystemExit



C:\Users\srira\anaconda3\envs\MRI\lib\site-packages\IPython\core\interactives
hell.py:3304: UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

Create a model

In [33]:
```
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
```

The logic behind the numbers

Here is the grand formula

$$n_{out} = ceil[\frac{n_{in} + 2p - f}{s} + 1]$$

Look into the parameters of the model

In [34]:
```
model = CNN()
```

In [35]:
```
model
```

Out[35]:
```
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]:
```
model.cnn_model
```

Out[38]:
```
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)
)
```

In [39]:
```
model.cnn_model[0]
```

Out[39]:
```
Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
```

In [40]:
```
model.cnn_model[0].weight
```

```
Parameter containing:
tensor([[[ 2.8615e-02,  1.4409e-04, -1.9788e-03,  4.6759e-02,  7.1333e-02],
         [-1.1318e-01, -6.9383e-02,  5.9661e-02,  5.0969e-02,  9.3040e-02],
         [-1.0679e-01,  1.0443e-01,  1.0835e-02, -2.4515e-02, -5.2875e-03],
         [-1.0584e-01, -1.2717e-02, -8.8371e-02, -5.1271e-02, -7.8316e-02],
         [-1.0457e-01, -7.9666e-02,  9.5497e-02, -1.5778e-02, -6.5364e-02]],

        [[ 8.0895e-02,  8.7384e-02,  7.9559e-02, -3.6089e-02, -9.6216e-02],
         [ 8.4533e-02,  2.3745e-03, -1.0332e-01, -1.1450e-01,  1.0005e-01],
         [-4.6434e-02,  7.2655e-02, -1.1040e-01,  1.3803e-02,  1.3546e-02],
         [ 9.8149e-02, -1.1156e-01,  5.7886e-03,  6.9954e-02, -8.1929e-03],
         [ 9.2074e-02,  8.9424e-02, -4.8969e-02, -2.2216e-02, -9.7844e-0
```

In [42]:
```
model.cnn_model[0].weight[0].shape
```

Out[42]:
```
torch.Size([3, 5, 5])
```

In [43]:
```
model.cnn_model[0].weight[0][1]
```

Out[43]:
```
tensor([[ 0.0286,  0.0001, -0.0020,  0.0468,  0.0713],
        [-0.1132, -0.0694,  0.0597,  0.0510,  0.0930],
        [-0.1068,  0.1044,  0.0108, -0.0245, -0.0053],
        [-0.1058, -0.0127, -0.0884, -0.0513, -0.0783],
        [-0.1046, -0.0797,  0.0955, -0.0158, -0.0654]],
       grad_fn=<SelectBackward0>)
```

Linear layer

In [44]:
```
model.fc_model
```

Out[44]:
```
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]:
```
model.fc_model[0]
```

Out[45]:
```
Linear(in_features=256, out_features=120, bias=True)
```

In [46]:
```
model.fc_model[0].weight.shape
```

Out[46]:
```
torch.Size([120, 256])
```

Understanding x.view(x.size(0),-1)

In [48]:
```
x = torch.tensor ([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16])
```

In [49]:
```
x = x.reshape((2,2,2,2))
```

In [51]:
```
x
```

Out[51]:
```
tensor([[[[ 1,  2],
          [ 3,  4]],

         [[ 5,  6],
          [ 7,  8]]],


        [[[ 9, 10],
          [11, 12]],

         [[13, 14],
          [15, 16]]]])
```

In [52]:
```
x.size(0)
```

Out[52]:
```
2
```

In [53]:
```
x.view(-1)
```

Out[53]:
```
tensor([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])
```

In [54]:
```
x.view(x.size(0), -1).shape
```

Out[54]:
```
torch.Size([2, 8])
```

Some Basics of Training and Evaluation in Pytorch

model.eval()

- Used particularly for inference **NOTHING to DO with gradients!!!**
- changes the forward() behaviour of the module it is called up on eg, it disables dropout and has batch norm use the entire population statistics. This is necessary for inference

model.train()

- Brings drop out and batch norm to action (i.e., train mode).
- Gradients are computed

numpy array vs tensor

The difference between a NumPy array and a tensor is that the tensors are backed by the accelerator memory like GPU and they are immutable, unlike NumPy arrays. You can never update a tensor but create a new one. If you are into machine learning or going to be into it, A Tensor is a suitable choice if you are going to use GPU. A tensor can reside in accelerator's memory.

- The numpy arrays are the core functionality of the numpy package designed to support faster mathematical operations. Unlike python's inbuilt list data structure, they can only hold elements of a single data type. Library like pandas which is used for data preprocessing is built around the numpy array. **Pytorch tensors are similar to numpy arrays, but can also be operated on CUDA-capable Nvidia GPU.**
- Numpy arrays are mainly used in typical machine learning algorithms (such as k-means or Decision Tree in scikit-learn) whereas pytorch tensors are mainly used in deep learning which requires heavy matrix computation.
- Unlike numpy arrays, while creating pytorch tensor, it also accepts two other arguments called the device_type (whether the computation happens on CPU or GPU) and the requires_grad (which is used to compute the derivatives).

torch.tensor vs. torch.cuda.tensor

he 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 [55]:
```
# 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([0.5118, 0.3718, 0.2437, 0.9759, 0.9754, 0.9759, 0.4781, 0.8415, 0.285
0,
        0.6792]) torch.float32 <class 'torch.Tensor'> torch.FloatTensor
tensor([0.5118, 0.3718, 0.2437, 0.9759, 0.9754, 0.9759, 0.4781, 0.8415, 0.285
0,
        0.6792]) torch.float32 <class 'torch.Tensor'> torch.FloatTensor
tensor([0.2619, 0.1383, 0.0594, 0.9523, 0.9514, 0.9524, 0.2286, 0.7081, 0.081
2,
        0.4613])
```

As the underlying hardware interface is completely different, CPU Tensors are just compatible with CPU Tensor and vice versa GPU Tensors are just compatible to GPU Tensors.

In which scenario is torch.cuda.Tensor() necessary?

When you want to use GPU acceleration (which is much faster in most cases) for your program, you need to use torch.cuda.Tensor, but you have to make sure that ALL tensors you are using are CUDA Tensors, mixing is not possible here.

tensor.cpu().detach().numpy(): Convert Pytorch tensor to Numpy array

As mentioned before, np.ndarray object does not have this extra "computational graph" layer and therefore, when converting a torch.tensor to np.ndarray you must explicitly remove the computational graph of the tensor using the detach() command. .cpu() returns a copy of this object in CPU memory.

Evaluate a New-Born Neural Network!

In [59]:
```
mri_dataset = MRI()
mri_dataset.normalize()
device = torch.device('cpu')
model = CNN().to(device)
```

In [60]:
```
dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=False)
```

In [61]:
```
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())
```

```
C:\Users\srira\anaconda3\envs\MRI\lib\site-packages\torch\nn\functional.py:1806: UserWarning: nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
```

In [62]:
```
outputs = np.concatenate( outputs, axis=0 ).squeeze()
y_true = np.concatenate( y_true, axis=0 ).squeeze()
```

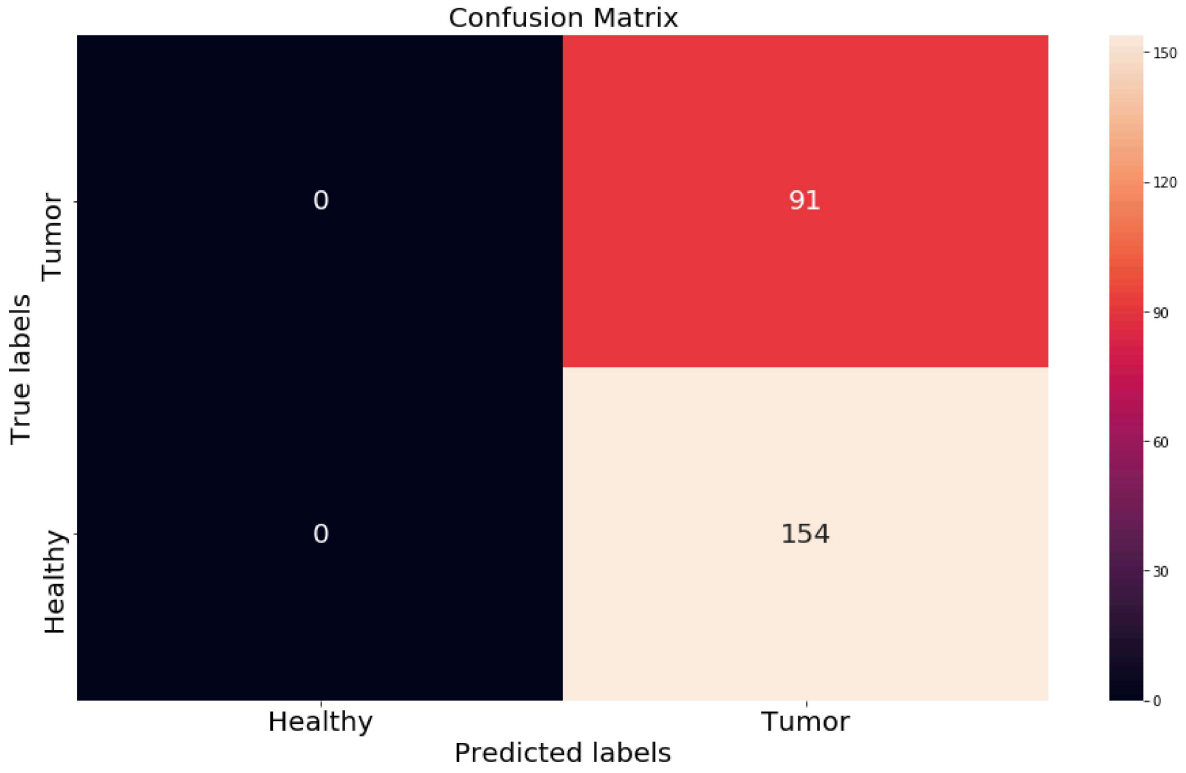
In [64]:
```
def threshold(scores,threshold=0.50, minimum=0, maximum = 1.0):
    x = np.array(list(scores))
    x[x >= threshold] = maximum
    x[x < threshold] = minimum
    return x
```

In [65]:
```
accuracy_score(y_true, threshold(outputs))
```

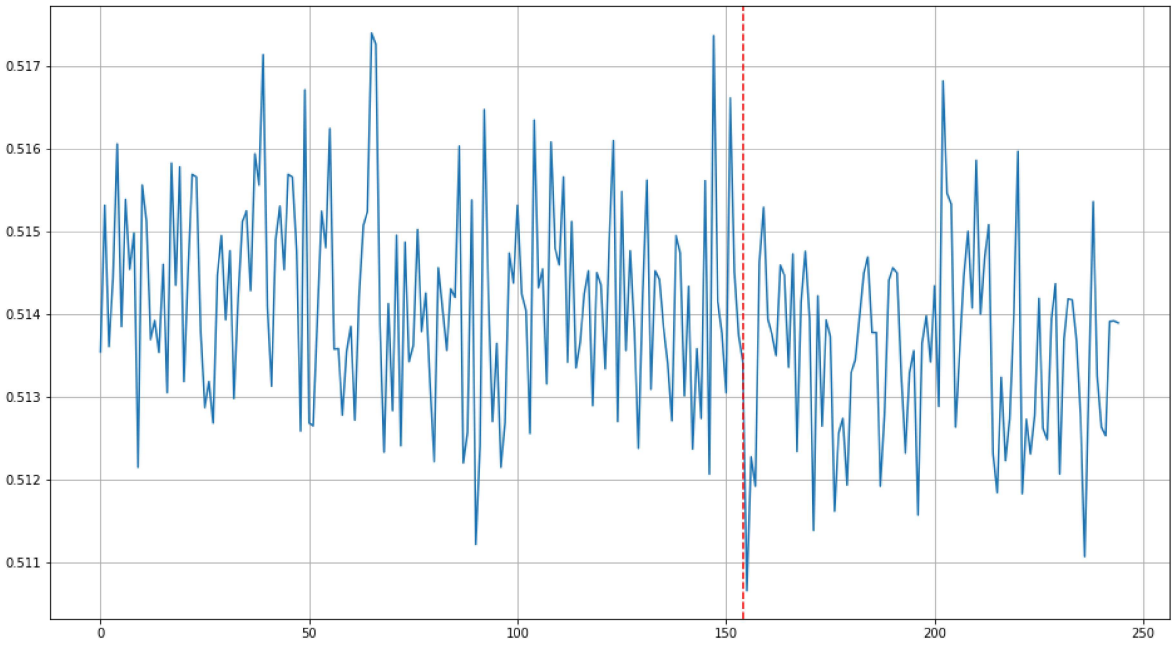
Out[65]:
```
0.6285714285714286
```

In [66]:
```
# 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)
```

Out[66]:
```
[Text(0, 0.5, 'Tumor'), Text(0, 1.5, 'Healthy')]
```



In [67]:
```
plt.figure(figsize=(16,9))
plt.plot(outputs)
plt.axvline(x=len(tumor), color='r', linestyle='--')
plt.grid()
```



Train the dumb model

In [68]:
```
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[68]:
```
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 [69]:
```
for epoch in range(1, EPOCH):
    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())
    if (epoch+1) % 10 == 0:
        print('Train Epoch: {}\tLoss: {:.6f}'.format(epoch+1, np.mean(losses))
```

```
C:\Users\srira\anaconda3\envs\MRI\lib\site-packages\torch\nn\functional.py:18
06: UserWarning: nn.functional.sigmoid is deprecated. Use torch.sigmoid inste
ad.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid inste
ad.")
```

```
Train Epoch: 10 Loss: 0.613238
Train Epoch: 20 Loss: 0.559900
Train Epoch: 30 Loss: 0.536614
Train Epoch: 40 Loss: 0.522238
Train Epoch: 50 Loss: 0.511623
Train Epoch: 60 Loss: 0.501433
Train Epoch: 70 Loss: 0.484001
Train Epoch: 80 Loss: 0.478290
Train Epoch: 90 Loss: 0.451469
Train Epoch: 100        Loss: 0.430972
Train Epoch: 110        Loss: 0.416857
Train Epoch: 120        Loss: 0.397001
Train Epoch: 130        Loss: 0.379315
Train Epoch: 140        Loss: 0.357959
Train Epoch: 150        Loss: 0.341558
Train Epoch: 160        Loss: 0.308392
Train Epoch: 170        Loss: 0.289783
Train Epoch: 180        Loss: 0.265693
Train Epoch: 190        Loss: 0.243074
Train Epoch: 200        Loss: 0.227914
Train Epoch: 210        Loss: 0.199677
Train Epoch: 220        Loss: 0.189192
Train Epoch: 230        Loss: 0.162135
Train Epoch: 240        Loss: 0.137883
Train Epoch: 250        Loss: 0.120065
Train Epoch: 260        Loss: 0.105881
Train Epoch: 270        Loss: 0.086956
Train Epoch: 280        Loss: 0.071348
Train Epoch: 290        Loss: 0.061062
Train Epoch: 300        Loss: 0.045453
Train Epoch: 310        Loss: 0.037165
Train Epoch: 320        Loss: 0.031246
Train Epoch: 330        Loss: 0.025958
Train Epoch: 340        Loss: 0.021502
Train Epoch: 350        Loss: 0.017400
Train Epoch: 360        Loss: 0.014252
Train Epoch: 370        Loss: 0.012353
Train Epoch: 380        Loss: 0.010776
Train Epoch: 390        Loss: 0.009125
Train Epoch: 400        Loss: 0.007856
```

Evaluate a smart model

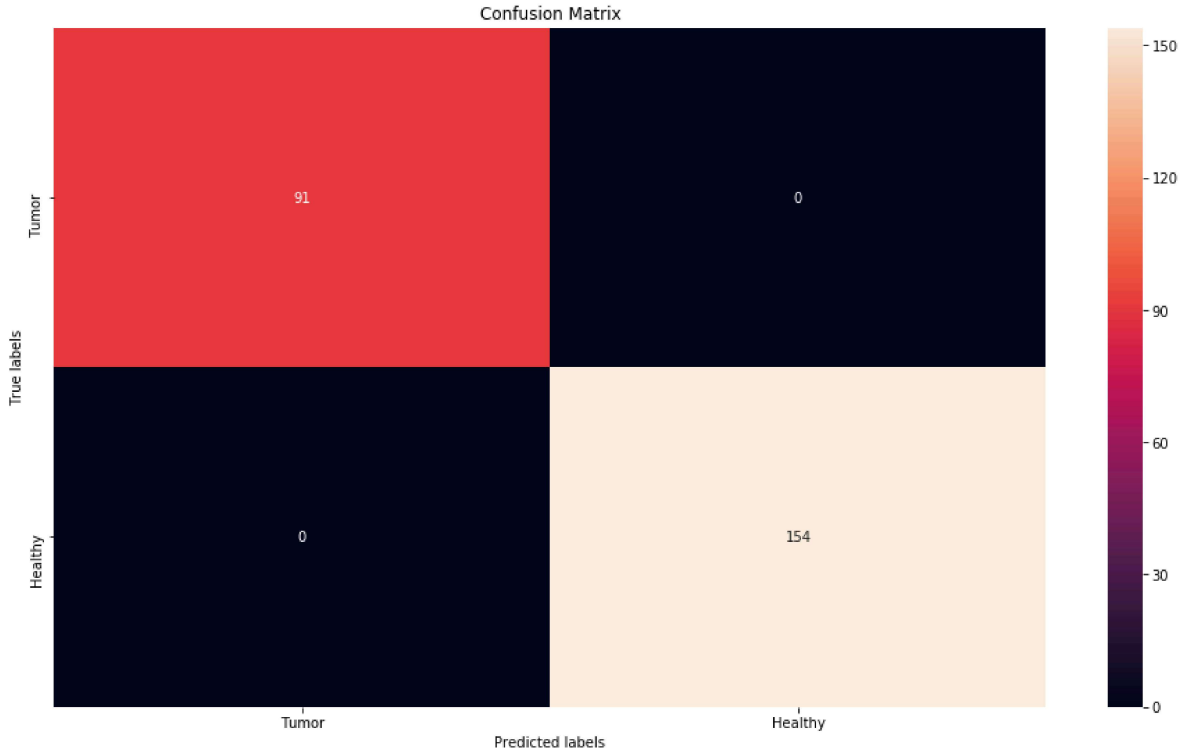
In [70]:
```
model.eval()
dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=False)
outputs=[]
y_true = []
with torch.no_grad():
    for D in dataloader:
        image =  D['image'].to(device)
        label = D['label'].to(device)

        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 [71]:
```
accuracy_score(y_true, threshold(outputs))
```

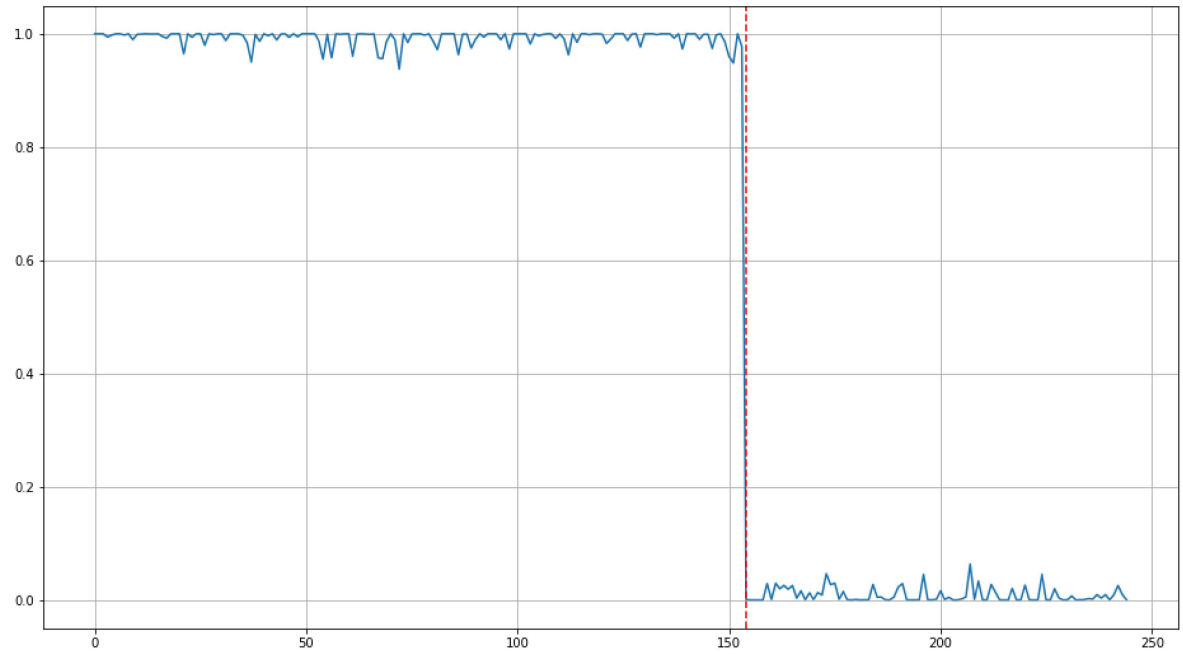
Out[71]: 1.0

In [72]:
```
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'])
```

Out[72]: [Text(0, 0.5, 'Tumor'), Text(0, 1.5, 'Healthy')]



In [73]:
```
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 [74]:
```
model
```

Out[74]:
```
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 [75]:
```
no_of_layers = 0
conv_layers = []
```

In [76]:
```
model_children = list(model.children())
model_children
```

Out[76]:
```
[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)
 )]
```

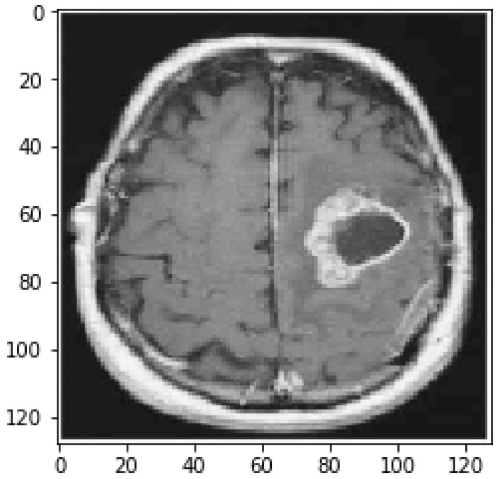
In [77]:
```
for child in model_children:
    if type(child) == nn.Sequential:
        for layer in child.children():
            if type(layer) == nn.Conv2d:
                no_of_layers += 1
                conv_layers.append(layer)
```

In [78]:
```
conv_layers
```

Out[78]:
```
[Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1)),
 Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))]
```

In [79]:
```
img = mri_dataset[100]['image']
plt.imshow(img.reshape(128,128,3))
```

Out[79]:
```
<matplotlib.image.AxesImage at 0x1d6cd612320>
```



In [80]:
```
img = torch.from_numpy(img).to(device)
```

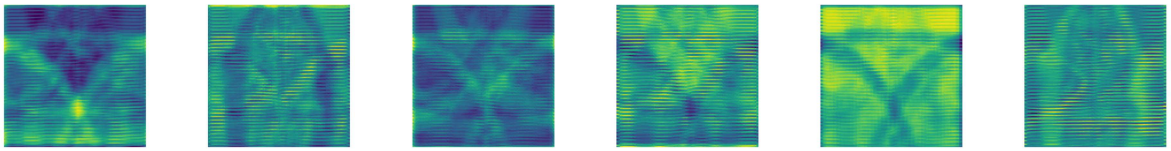
In [81]:
```
img.shape
```

Out[81]:
```
torch.Size([3, 128, 128])
```

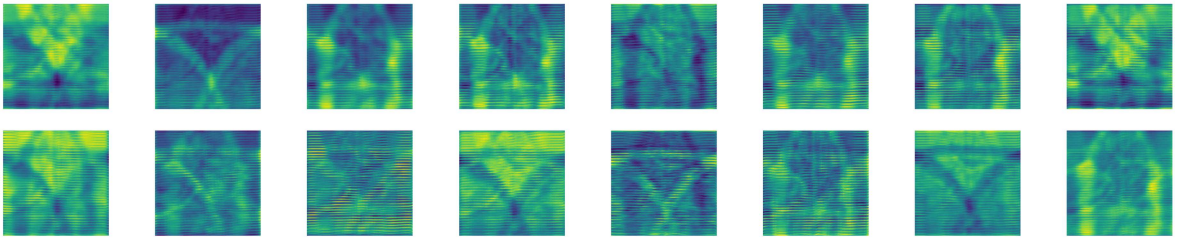
In [83]:
```
img = img.unsqueeze(0)  # Add batch dimension
results = [conv_layers[0](img)]
for i in range(1, len(conv_layers)):
    results.append(conv_layers[i](results[-1]))
outputs = results
```

In [84]:
```
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!

We will need to make changes to our **MRI dataset class**:

- Define a function to divide the data into train and validation sets
- Define a variable called **mode** to determine whether we are interested in the training OR validation data
- Change **len()** and **getitem**() functions and conditioned over the variable **mode**

In [86]:
```
# Import train/test split function from sklearn
from sklearn.model_selection import train_test_split
```

In [87]:
```
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("./data/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("./data/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 [88]:
```
mri_dataset = MRI()
mri_dataset.normalize()
mri_dataset.train_val_split()
```

In [89]:
```
train_dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=True)
val_dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=False)
```

In [91]:
```
device = torch.device("cpu")
model = CNN().to(device)
```

In [92]:
```
eta=0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=eta)
```

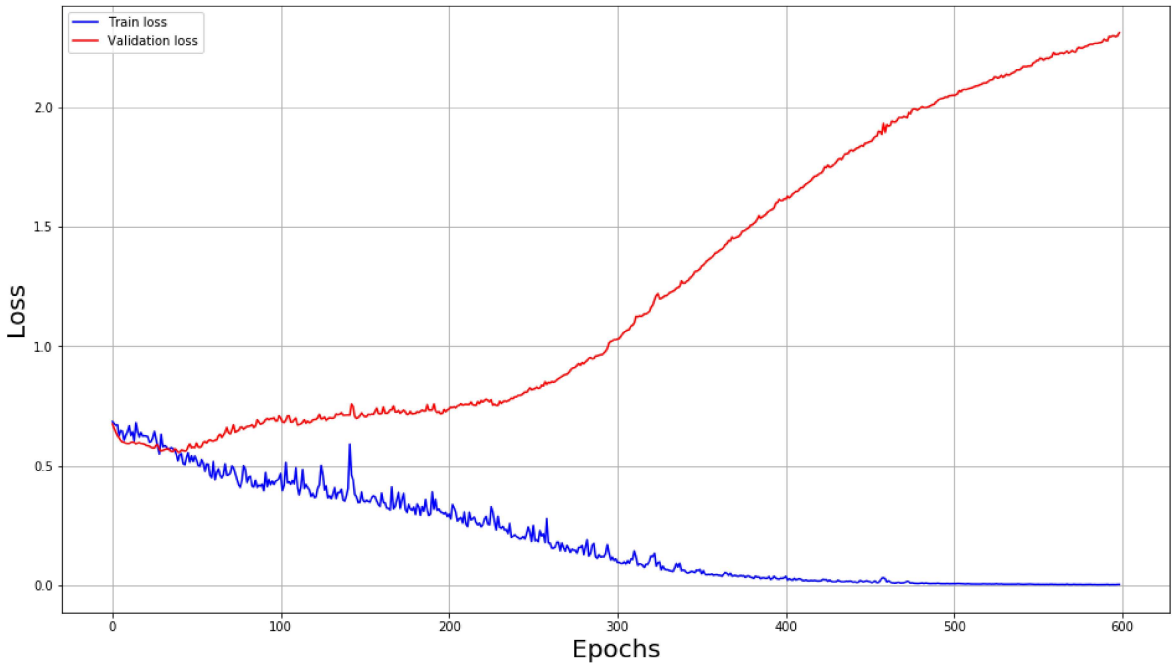
In [93]:
```
# keep track of epoch losses
epoch_train_loss = []
epoch_val_loss = []
```

In [94]:
```
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))

    # validate for the current epoch
    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))

    if (epoch+1) % 10 == 0:
        print('Train Epoch: {}\tTrain Loss: {:.6f}\tVal Loss: {:.6f}'.format(e
```

```
Train Epoch: 10 Train Loss: 0.626618    Val Loss: 0.594669
Train Epoch: 20 Train Loss: 0.624329    Val Loss: 0.593146
Train Epoch: 30 Train Loss: 0.549134    Val Loss: 0.570088
Train Epoch: 40 Train Loss: 0.547733    Val Loss: 0.568139
Train Epoch: 50 Train Loss: 0.518123    Val Loss: 0.578808
Train Epoch: 60 Train Loss: 0.460436    Val Loss: 0.608882
Train Epoch: 70 Train Loss: 0.460993    Val Loss: 0.660111
Train Epoch: 80 Train Loss: 0.500268    Val Loss: 0.653505
Train Epoch: 90 Train Loss: 0.415764    Val Loss: 0.673874
Train Epoch: 100        Train Loss: 0.440570    Val Loss: 0.688613
Train Epoch: 110        Train Loss: 0.425302    Val Loss: 0.693222
Train Epoch: 120        Train Loss: 0.372940    Val Loss: 0.683721
Train Epoch: 130        Train Loss: 0.366797    Val Loss: 0.696388
Train Epoch: 140        Train Loss: 0.352535    Val Loss: 0.711878
Train Epoch: 150        Train Loss: 0.390249    Val Loss: 0.721253
Train Epoch: 160        Train Loss: 0.338560    Val Loss: 0.718994
Train Epoch: 170        Train Loss: 0.331603    Val Loss: 0.725447
Train Epoch: 180        Train Loss: 0.313739    Val Loss: 0.716998
Train Epoch: 190        Train Loss: 0.292859    Val Loss: 0.733707
Train Epoch: 200        Train Loss: 0.305369    Val Loss: 0.726395
Train Epoch: 210        Train Loss: 0.261251    Val Loss: 0.754153
Train Epoch: 220        Train Loss: 0.260625    Val Loss: 0.760360
Train Epoch: 230        Train Loss: 0.232124    Val Loss: 0.752555
Train Epoch: 240        Train Loss: 0.205698    Val Loss: 0.780609
Train Epoch: 250        Train Loss: 0.224908    Val Loss: 0.825945
Train Epoch: 260        Train Loss: 0.279764    Val Loss: 0.840991
Train Epoch: 270        Train Loss: 0.157038    Val Loss: 0.879134
Train Epoch: 280        Train Loss: 0.158401    Val Loss: 0.921043
Train Epoch: 290        Train Loss: 0.113376    Val Loss: 0.959788
Train Epoch: 300        Train Loss: 0.105093    Val Loss: 1.027755
Train Epoch: 310        Train Loss: 0.110376    Val Loss: 1.084217
Train Epoch: 320        Train Loss: 0.088628    Val Loss: 1.139928
Train Epoch: 330        Train Loss: 0.068093    Val Loss: 1.210473
Train Epoch: 340        Train Loss: 0.062575    Val Loss: 1.273066
Train Epoch: 350        Train Loss: 0.062688    Val Loss: 1.317345
Train Epoch: 360        Train Loss: 0.044383    Val Loss: 1.388746
Train Epoch: 370        Train Loss: 0.039089    Val Loss: 1.457390
Train Epoch: 380        Train Loss: 0.039524    Val Loss: 1.495853
Train Epoch: 390        Train Loss: 0.029998    Val Loss: 1.549515
Train Epoch: 400        Train Loss: 0.029658    Val Loss: 1.608582
Train Epoch: 410        Train Loss: 0.028185    Val Loss: 1.655849
Train Epoch: 420        Train Loss: 0.019675    Val Loss: 1.710487
Train Epoch: 430        Train Loss: 0.015302    Val Loss: 1.757073
Train Epoch: 440        Train Loss: 0.018502    Val Loss: 1.817910
Train Epoch: 450        Train Loss: 0.019577    Val Loss: 1.850106
Train Epoch: 460        Train Loss: 0.032678    Val Loss: 1.931545
Train Epoch: 470        Train Loss: 0.010090    Val Loss: 1.956541
Train Epoch: 480        Train Loss: 0.007997    Val Loss: 1.986927
Train Epoch: 490        Train Loss: 0.007958    Val Loss: 2.011108
Train Epoch: 500        Train Loss: 0.006886    Val Loss: 2.047710
Train Epoch: 510        Train Loss: 0.006984    Val Loss: 2.075544
Train Epoch: 520        Train Loss: 0.006132    Val Loss: 2.100352
Train Epoch: 530        Train Loss: 0.006213    Val Loss: 2.132497
Train Epoch: 540        Train Loss: 0.006059    Val Loss: 2.155070
Train Epoch: 550        Train Loss: 0.004082    Val Loss: 2.187640
Train Epoch: 560        Train Loss: 0.004882    Val Loss: 2.210636
Train Epoch: 570        Train Loss: 0.003554    Val Loss: 2.225332
Train Epoch: 580        Train Loss: 0.003674    Val Loss: 2.253994
Train Epoch: 590        Train Loss: 0.003566    Val Loss: 2.273471
Train Epoch: 600        Train Loss: 0.003930    Val Loss: 2.310722
```

In [95]:
```
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[95]:
```
Text(0, 0.5, 'Loss')
```



In [ ]:
```
```