CIS 410 Progress Report
Nick Titzler

1. Updated Statement of project goals:

   I'd like to create a classification algorithm for brain tumors, and a localization function to highlight the tumors. I updated the project to include localization to add a layer of complexity to the project and give myself practice with object detection. Initially, I am testing simply no tumor vs tumor. Subsequently I will attempt to build the classifier which distinguishes tumor type.

2. Current member roles and collaboration strategy

   I am the only member of the group.

3. Proposed approach

   For the classifier, I am using RES net, as I have had the most success in the previous assignment. I have not experimented yet with the algorithm best suited for object detection, but my plan now is to try YOLO first.

   Pseudocode and approach:

   1. Load dataset
      - Dataset will be stored in a custom tumor class ensuring that I can transform data easily.
      - Include function to transfer image to tensor
      - As a reference I will be using this guide: https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
      - Class example:

```
def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.tumor_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = os.path.join(self.root_dir,
                                self.tumor_frame.iloc[idx, 0])
        image = io.imread(img_name)
        tumor = self.tumor_frame.iloc[idx, 1:]
        tumor = np.array([tumor])
        tumor = tumor.astype('float').reshape(-1, 2)
        sample = {'image': image, 'tumor': tumor}

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

        return sample
```

2. Crop and resize dataset to all have same pixel lengths
    a. Resizing will be preformed with transform.resize()
    b. I will be augmenting this step this attribute after the model is written to see if it has an effect on accuracy.

```
def resize(self, sample):
    image, tumor = sample['image'], sample['tumor']

    h, w = image.shape[:2]
    if isinstance(self.output_size, int):
        if h > w:
            new_h, new_w = self.output_size * h / w, self.output_size
        else:
            new_h, new_w = self.output_size, self.output_size * w / h
    else:
        new_h, new_w = self.output_size

    new_h, new_w = int(new_h), int(new_w)

    img = transform.resize(image, (new_h, new_w))

    # h and w are swapped for tumor because for images,
    # x and y axes are axis 1 and 0 respectively
    tumor = tumor * [new_w / w, new_h / h]

    return {'image': img, 'tumor': tumor}
```

3. Apply res net

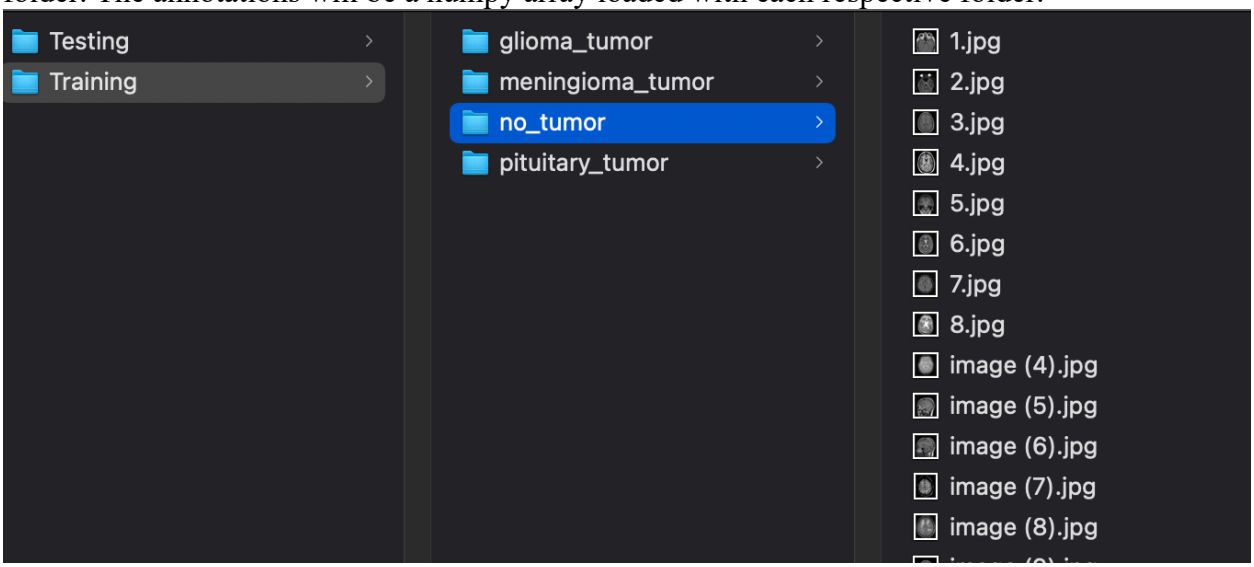
```python
# ResNet
class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes=10):
        super(ResNet, self).__init__()
        self.in_channels = 16
        self.conv = conv3x3(3, 16)
        self.bn = nn.BatchNorm2d(16)
        self.relu = nn.ReLU(inplace=True)
        self.layer1 = self.make_layer(block, 16, layers[0])
        self.layer2 = self.make_layer(block, 32, layers[1], 2)
        self.layer3 = self.make_layer(block, 64, layers[2], 2)
        self.avg_pool = nn.AvgPool2d(8)
        self.fc = nn.Linear(64, num_classes)

    def make_layer(self, block, out_channels, blocks, stride=1):
        downsample = None
        if (stride != 1) or (self.in_channels != out_channels):
            downsample = nn.Sequential(
                conv3x3(self.in_channels, out_channels, stride=stride),
                nn.BatchNorm2d(out_channels))
        layers = []
        layers.append(block(self.in_channels, out_channels, stride, downsample))
        self.in_channels = out_channels
        for i in range(1, blocks):
            layers.append(block(out_channels, out_channels))
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv(x)
        out = self.bn(out)
        out = self.relu(out)
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out
```

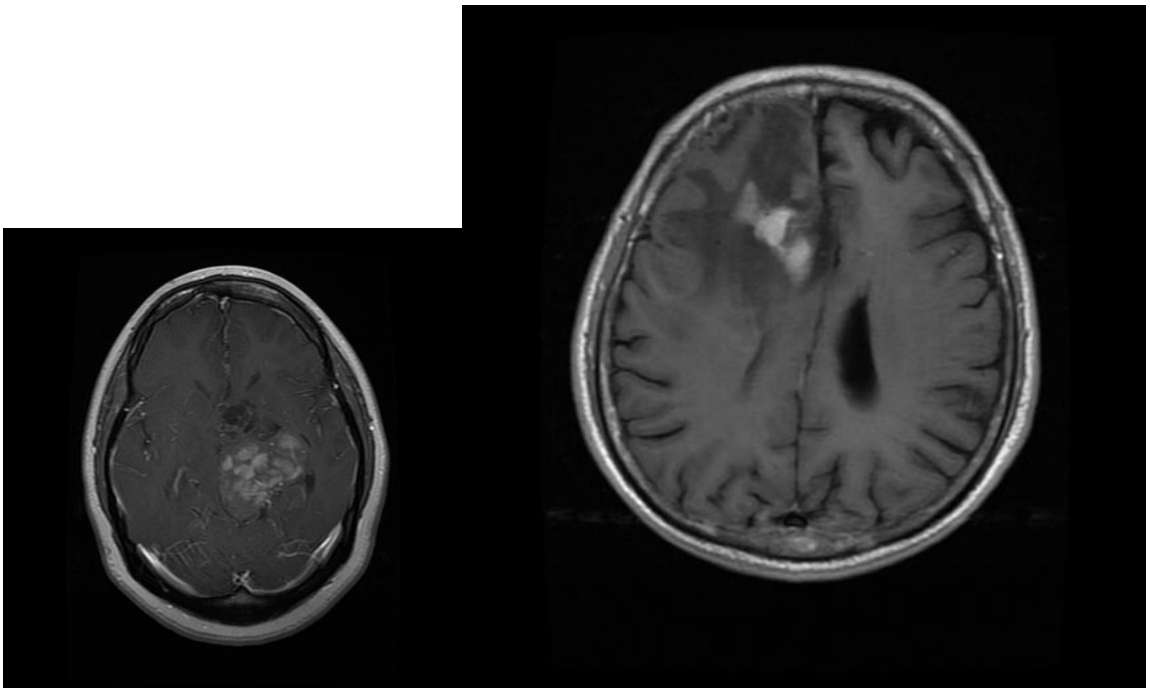
4. Apply localization algorithm

   Unsure of implementation at this time.

4. Data

I've organized my dataset into the four different tumor types. Annotations are divided by folder. The annotations will be a numpy array loaded with each respective folder.





5. Initial results

I've had trouble so far getting the dataset loaded into google collab and transferred into tensors. I've tried several methods unsuccessfully, and only recently implemented the method I showed above in step 3. I believe I am having problems with my resize function as the training functions are erroring with matrix size failures, which leads me to believe something is off with the tensor sizing.

6. Current reservations and questions

I think ill be able to classify the images with a high degree of accuracy, but as of now I am unsure of my ability to perfect the localization part of the algorithm. I believe this will be more challenging.