Brain Tumor Image Classifier

Alex Lai

Problem Statement

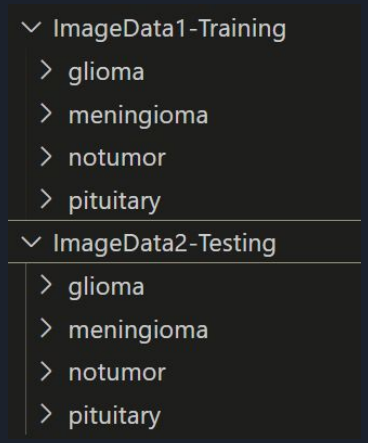
- Problem
  - Too many unorganized brain MRIs that need to be sorted as having cancers or not.
  - Need a train eye.
  - The amount of images is too much for doctors
- Solution
  - To make an image-processing brain tumor predictive model to automate this problem on scale.
  - CNN model on pytorch.
- Stakeholders
  - Hospital
  - Doctors
- Consider as pass
  - Validation loss under 0.1, no signs of overfitting, minimum of 95% accuracy.

Description of Dataset

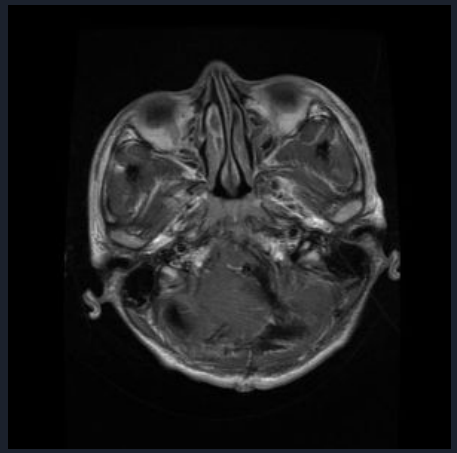
This data was provided by the IEEE Data Port:
https://ieee-dataport.org/documents/brain-tumor-mri-dataset

And is available on Kaggle:
https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset

Data folder structure



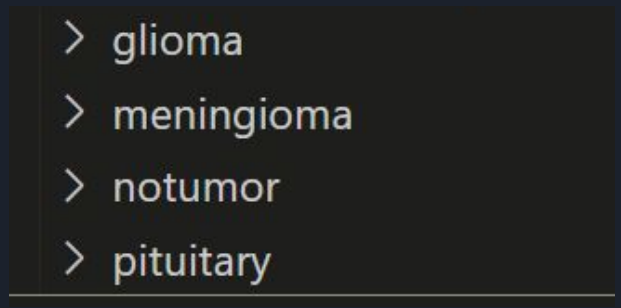
Grayscale images



Just a sample of all the data

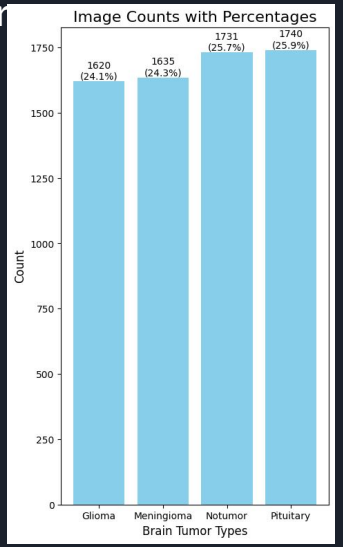
Brain MRI info

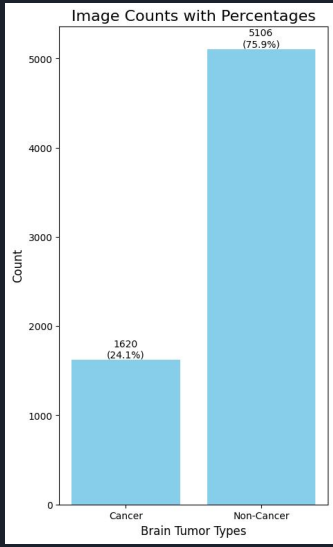
- Glioma - A type of cancer arising from glial cells in the brain or spinal cord.
  - Actual Cancer
- Meningioma - Usually a benign tumor from the meninges; rarely malignant.
- Notumor - Means no tumor was found.
- Pituitary - Refers to the gland; tumors (mostly benign) can form, rarely cancerous.



The Plan

- End goal is to make a model where you input image data of Brain MRI and will classify either Cancer or No Cancer
- We will be experimenting with different data set ideas.



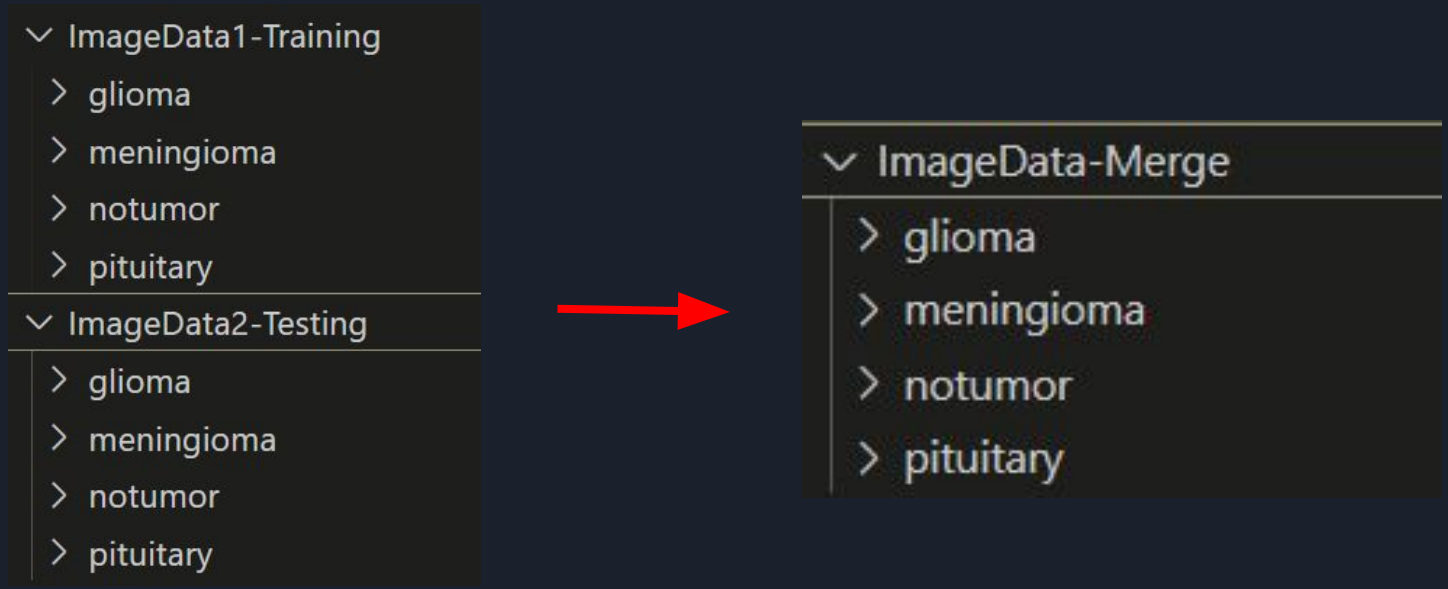


- Idea 1 is keep it as 4 classes
  - Glioma, meningioma, notumor, pituitary
- Idea 2 is to make it 2 classes.
  - Glioma as ‘cancer’
  - Meningioma, notumor, pituitar all as ‘no cancer’.
  - Data will be imbalanced

	4 classes	2 classes
Non-Augmented 512x512 6K data	Dataset 1 4 na 512	Dataset 2 2 na 512

Merge Data for Cross Validation

- Cross validation will need K folds from one sample source
- Original data was split, will merge them together as then easier to create folds for Cross Validation.



Cleaning Data Methods

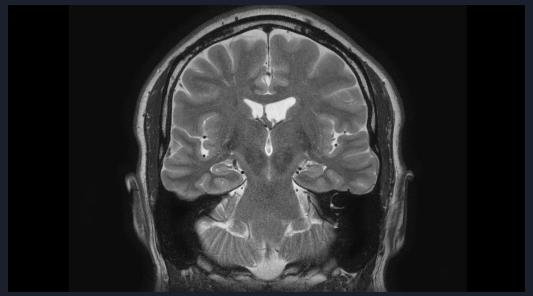
- Scan for corrupted images and delete those.
- Scan for duplicate images and delete those.
- Scan for Images if varying files sizes
  - Some outliers but checked and not that big of an issue
- Standardized the images by scaling the pixel values between 0 and 1.

Resize Image Data

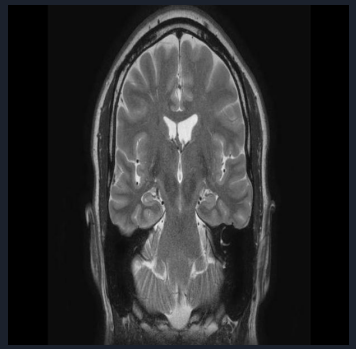
Alot of images are of different size.

1st Attempt

Original

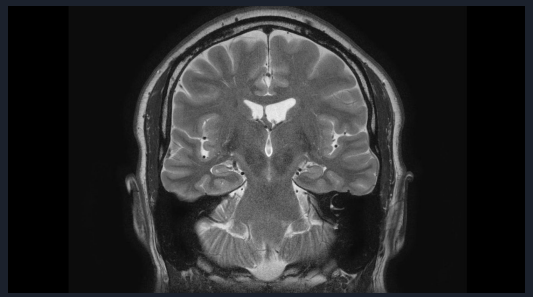


Resized (stretched/squashed):

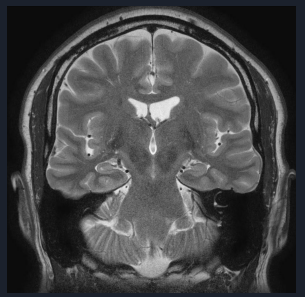


2nd Attempt

Original

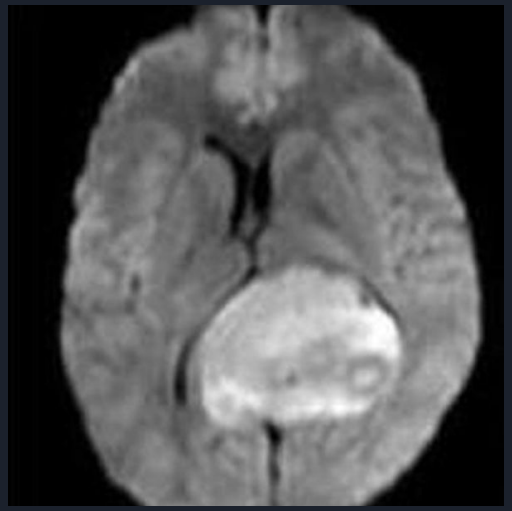


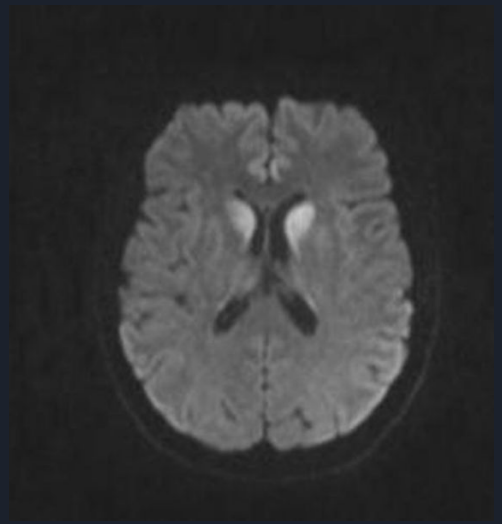
Crop then Resize:



Check for Blurry Images after resizing

- Laplacian variance method
- Threshold set to 10
- Deleted the images





Convert to grayscale

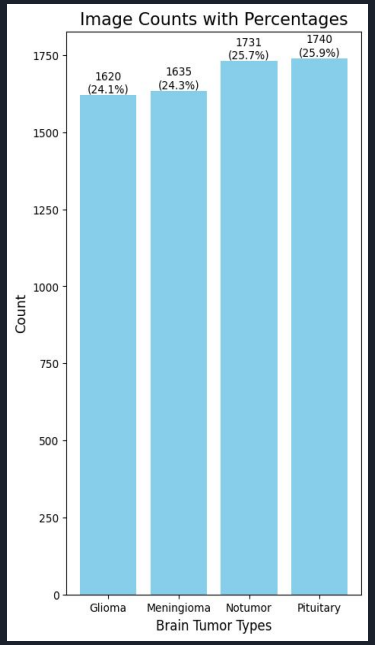
```
return transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.Resize(resize),
    transforms.ToTensor(),
])
```

Data Augmentation

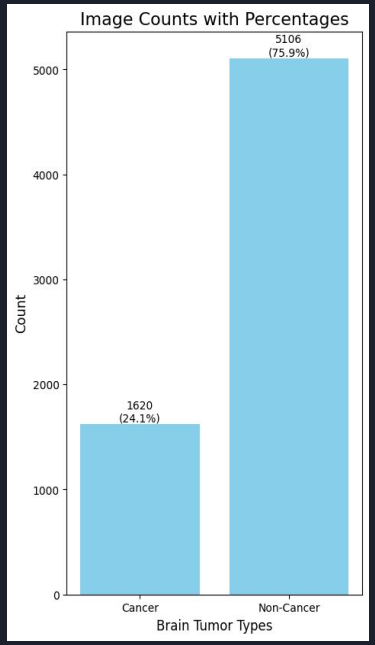
```
return transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.Resize(resize),
    transforms.ToTensor(),
])
```

Data Imbalance

4 class data balance



2 class data imbalance



Make minority class weigh more in training

```python
def calculate_class_weights(dataset):
    """
    Calculates class weights for handling class imbalance.

    Args:
        dataset (ImageFolder): The dataset object.

    Returns:
        torch.Tensor: Class weights.
    """
    class_counts = [0] * len(dataset.classes)
    for _, label in dataset:
        class_counts[label] += 1

    class_weights = 1.0 / torch.tensor(class_counts, dtype=torch.float)
    return class_weights
```

Batch Size

Things to consider:

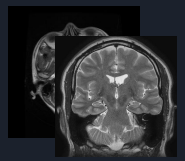
- image dimension - 512x512 images
- Amount of data - 6K amount of data
- Batch size - 64 Batch size
- VRAM memory - Max 24GB VRAM memory
- Time - Within time of 8 hours

- batch size 256 with 512 x 512 images took longer than 30 min to run for only 1 fold, 97% GPU utilization
- batch size 128 with 512 x 512 images took longer than 30 min to run for only 1 fold, 97% GPU utilization
- batch size 64 with 512 x 512 images took 3:50 min for 1 fold, 14 GB VRAM, 40% GPU utilization

CNN model design

```python
class FlexibleCNN(nn.Module):
    def __init__(self, num_classes):
        super(FlexibleCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, stride=1, padding=1)  # in_channels=1 for grayscale images
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Fully connected layers
        self.fc1 = None  # Placeholder; initialized later based on input size
        self.fc2 = nn.Linear(128, num_classes)  # Final layer depends only on num_classes

    def forward(self, x):
        # Pass through convolutional and pooling layers
        x = F.relu(self.conv1(x))  # Conv1 + ReLU
        x = self.pool(x)           # Pool1
        x = F.relu(self.conv2(x))  # Conv2 + ReLU
        x = self.pool(x)           # Pool2

        # Initialize fc1 dynamically based on the flattened size
        if self.fc1 is None:
            flattened_size = x.view(x.size(0), -1).size(1)
            self.fc1 = nn.Linear(flattened_size, 128).to(x.device)

        x = x.view(x.size(0), -1)  # Flatten feature map
        x = F.relu(self.fc1(x))    # Fully connected layer 1
        x = self.fc2(x)            # Fully connected layer 2 (output)
        return x
```

512x512 grayscale images batch size 64



Conv layer 32 filters

Conv layer 64 filters

Pooling layer

Output layer to number of classes

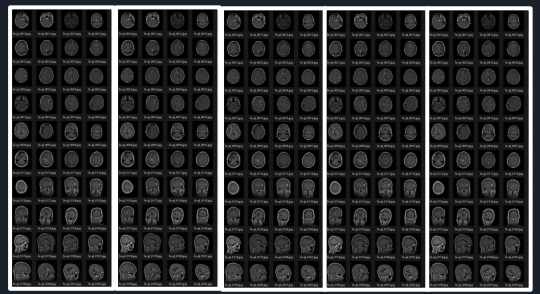
Model 1 is 4 clases

Model 2 is 2 classes

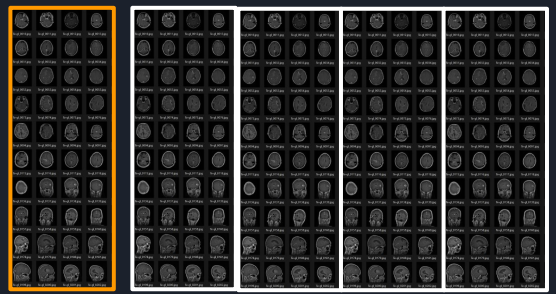
Cross validation & K folds

All data sets

Divide up into 5 folds



Each time run a folds alternates which fold is used for validation



Each model will run each fold 10 epochs

Modeling Results

Model 1 with 4 classes

4_na			
Fold	training loss	validation loss	accuracy
1	0.08880937505	0.2313201197	0.93
2	0.08777307522	0.2673308484	0.91
3	0.1160536457	0.2007067729	0.93
4	0.07483411169	0.3527899463	0.92
5	0.09756438988	0.2078608416	0.92
avg	0.0930069195	0.2520017058	0.922

Model 2 with 2 classes

2_na			
Fold	training loss	validation loss	accuracy
1	0.07948613391	0.119994813	0.96
2	0.09081313622	0.1057662009	0.96
3	0.05795129627	0.1426240807	0.95
4	0.06212242695	0.1836265536	0.95
5	0.07305600591	0.1262279286	0.95
avg	0.07268579985	0.1356479154	0.954

winner

Modeling Results

Model 1 with 4 classes

4_na			
Fold	training loss	validation loss	accuracy
1	0.08880937505	0.2313201197	0.93
2	0.08777307522	0.2673308484	0.91
3	0.1160536457	0.2007067729	0.93
4	0.07483411169	0.3527899463	0.92
5	0.09756438988	0.2078608416	0.92
avg	0.0930069195	0.2520017058	0.922

Model 2 with 2 classes

2_na			
Fold	training loss	validation loss	accuracy
1	0.07948613391	0.119994813	0.96
2	0.09081313622	0.1057662009	0.96
3	0.05795129627	0.1426240807	0.95
4	0.06212242695	0.1836265536	0.95
5	0.07305600591	0.1262279286	0.95
avg	0.07268579985	0.1356479154	0.954

Slight overfitting?

Modeling Results

Model 1 with 4 classes

4_na			
Fold	training loss	validation loss	accuracy
1	0.08880937505	0.2313201197	0.93
2	0.08777307522	0.2673308484	0.91
3	0.1160536457	0.2007067729	0.93
4	0.07483411169	0.3527899463	0.92
5	0.09756438988	0.2078608416	0.92
avg	0.0930069195	0.2520017058	0.922

Model 2 with 2 classes

2_na			
Fold	training loss	validation loss	accuracy
1	0.07948613391	0.119994813	0.96
2	0.09081313622	0.1057662009	0.96
3	0.05795129627	0.1426240807	0.95
4	0.06212242695	0.1836265536	0.95
5	0.07305600591	0.1262279286	0.95
avg	0.07268579985	0.1356479154	0.954

winner

Conclusion, Future Work, & Improving the model

Conclusion:

- Chose Model 2 with the dataset of merging the data into 2 classes.

Possible issues:

- Possible mislabeling

Future Works:

- Experiment with augmenting the data (rot and flip) which will increase the data set from 6K to 87K
- Optimize the parallel computing to process more data faster
- Experiment different types of CNN models structures than only the simple one. Chose small as dataset was small.

Thank You

Questions?