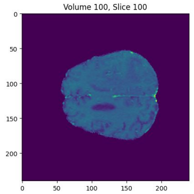
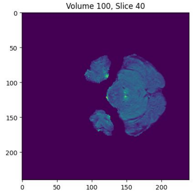
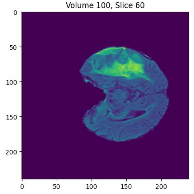
Dataset Description

369 Volumes

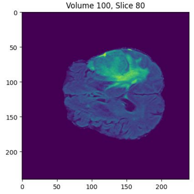
155 Slices per Volume



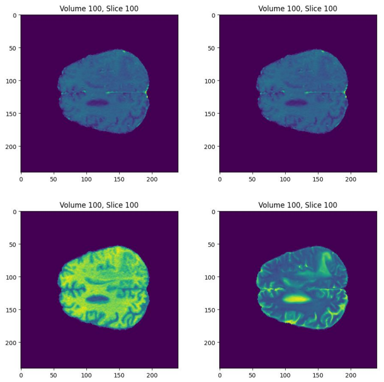


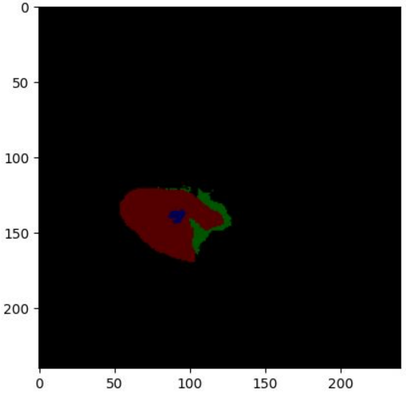


⋮



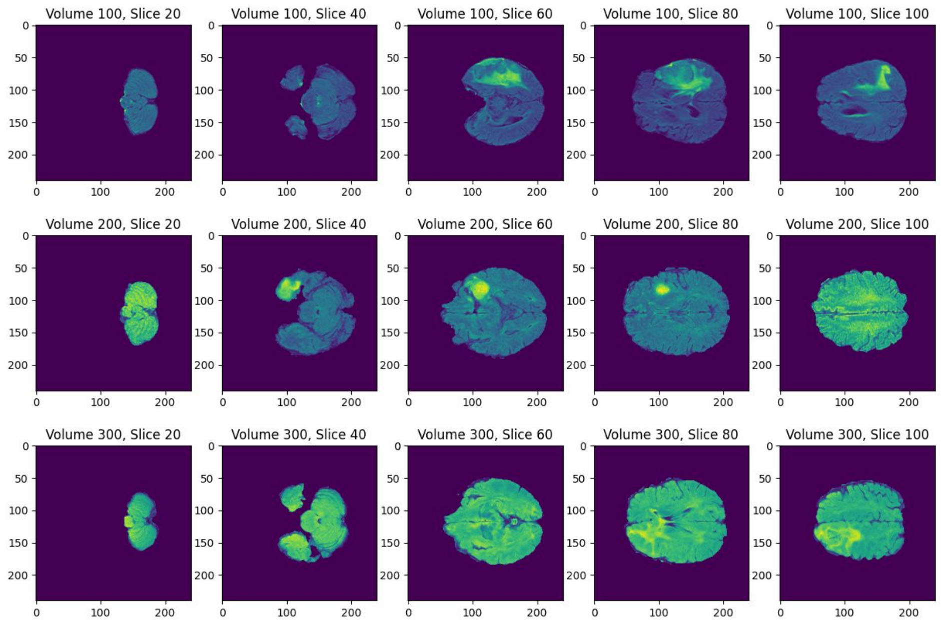
4 Image variants per slice (Native, T1, T2, T2-Flair)
+
1 image mask per slice



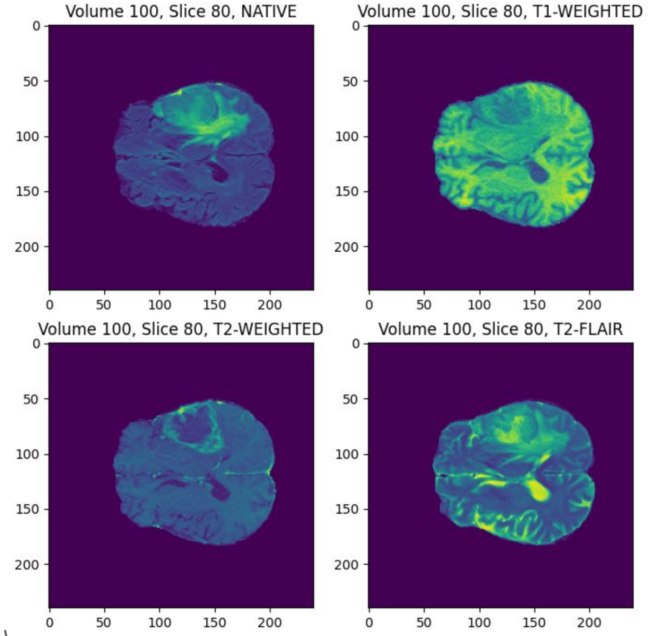


```
Total number of volumes: 369

Total number of slices: 57195

Volumes with no positive slices: 0 (0.0)

Volumes with at least one positive slice: 369 (1.0)

Negative slices: 32773 (0.5730046332721391)

Positive slices: 24422 (0.4269953667278608)
```

Sample Images

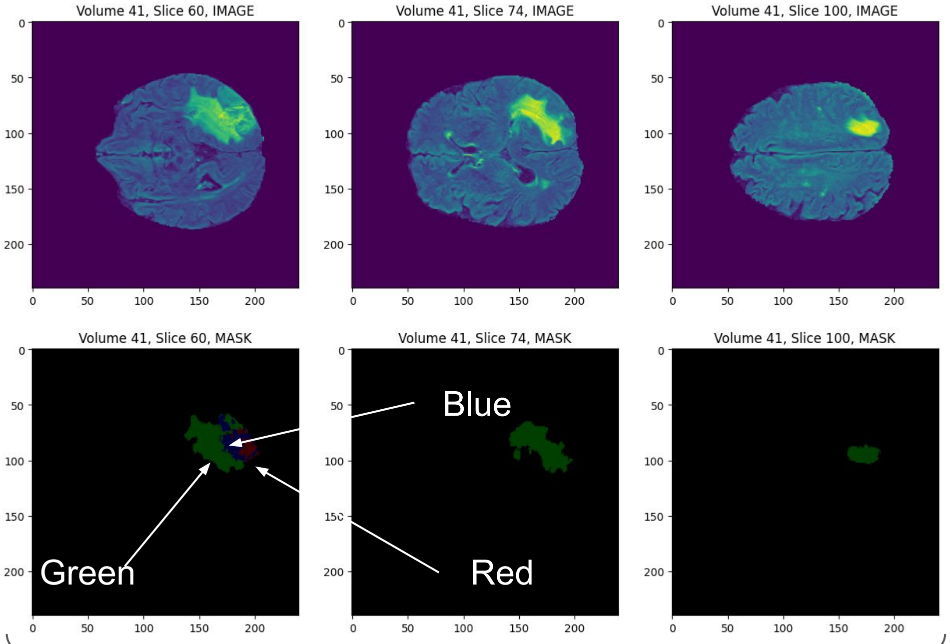


Slices of similar number correspond to same parts of the brain



Different variants highlight different structures for the same slice

Sample Masks

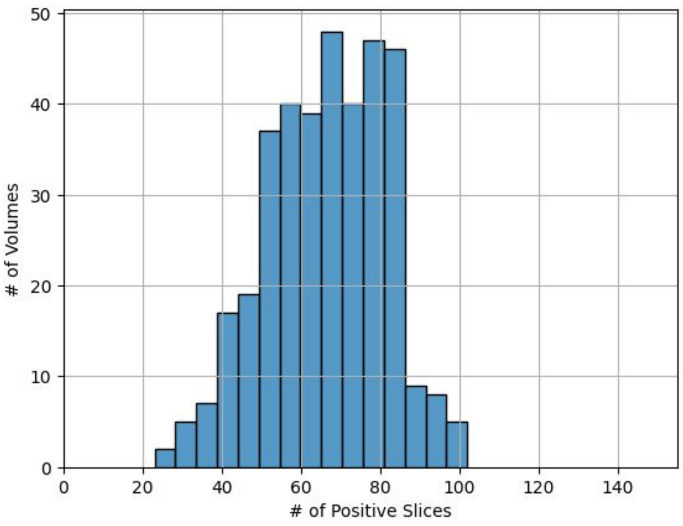


Mask labels have each pixel as 4 possible values (identified by green, red, blue and black) above, which indicate (ET) GD-enhancing tumor, (ED) peritumoral edema, (NCR) necrotic and non-enhancing tumor core, and normal

Data Exploration

Q: How many positive slices does each volume have?

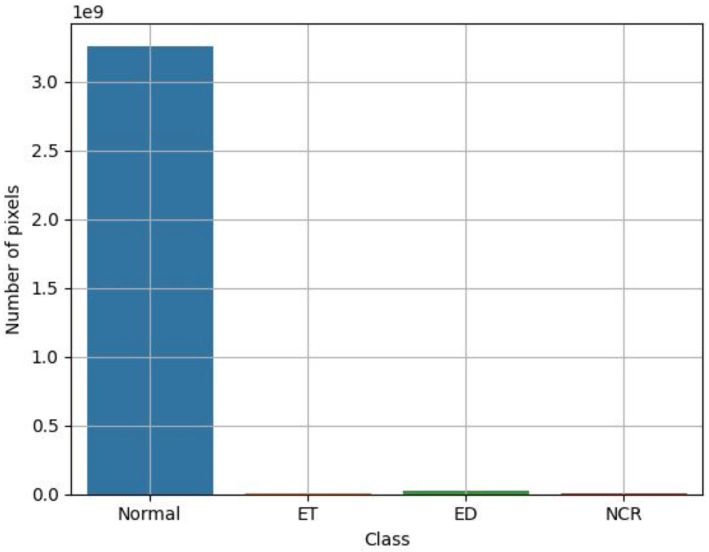
A: Each volume has between ~20 and ~100 positive slices. Most volumes have around ~60 to ~80 positive slices



Data Exploration

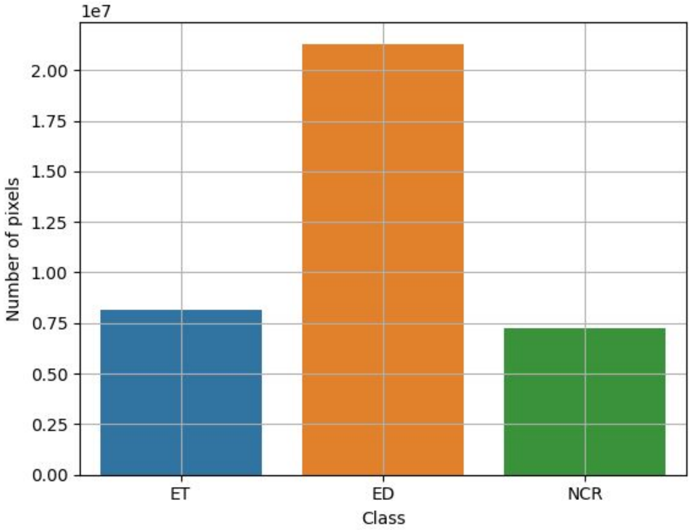
Q: Is the dataset balanced with respect to the different mask classes?

A: No. The vast majority of pixels is "Normal", probably techniques are needed to deal with this.



Q: Ignoring the "Normal" class, is the dataset balanced?

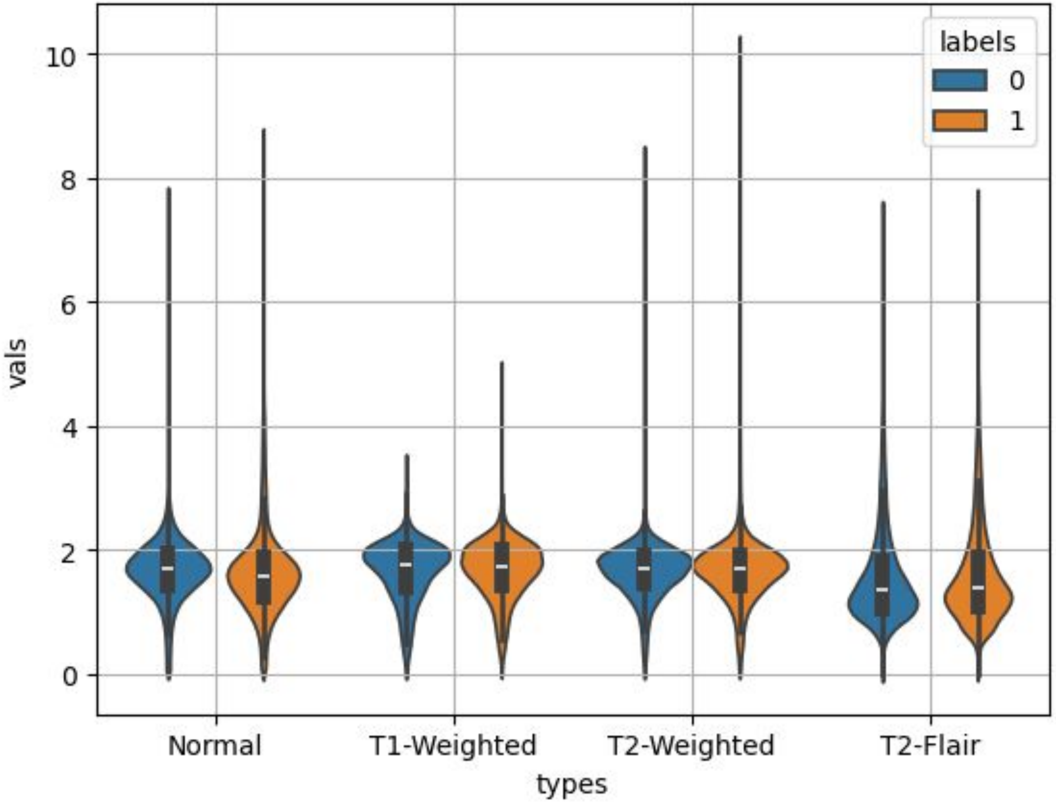
A: Not perfectly balanced, but the number of pixels of each class is comfortably within the same order of magnitude.



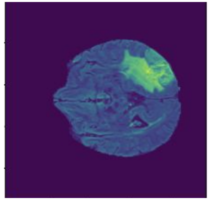
Data Exploration

Q: Is the distribution of image pixel values measurably different between positive vs negative slices?

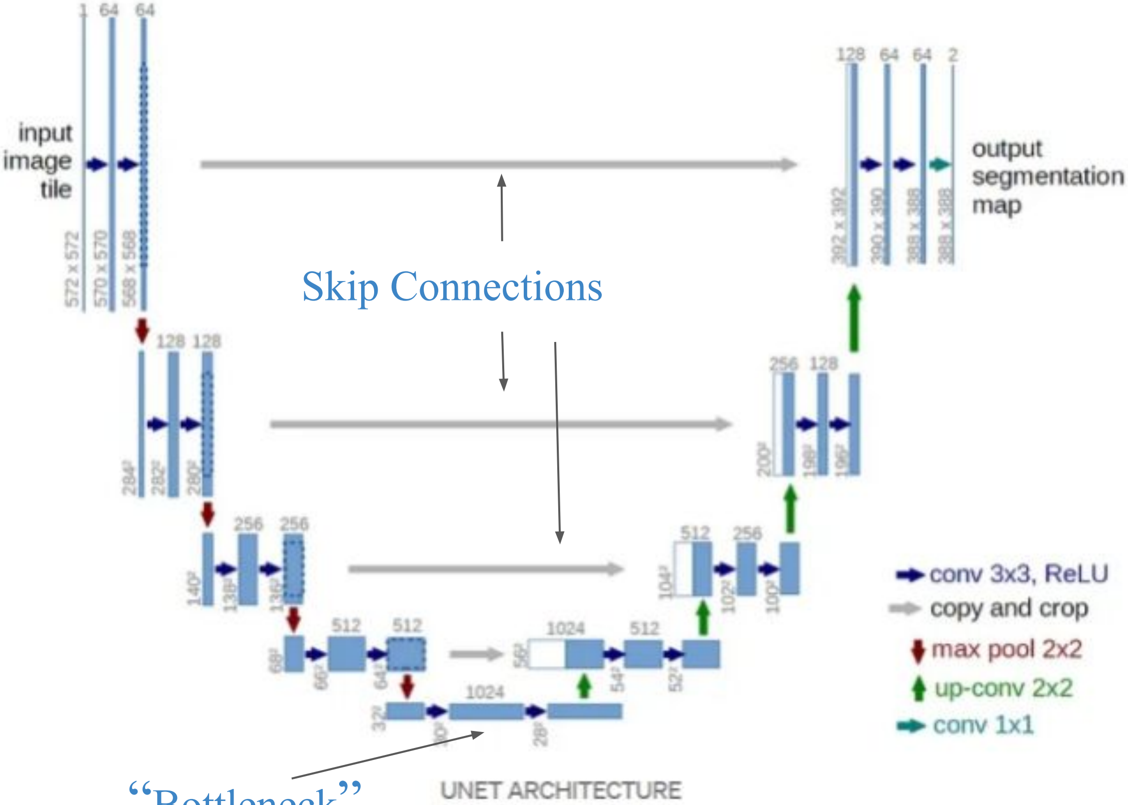
A: Distribution of pixel values is quite different between the different image variants, but the difference between positive and negative images is extremely sutil at best. In all 4 variants though, the positive images seem to have higher pixel brightness upper bounds.



ML technique: U-Net Architecture

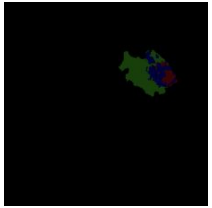


Encoder
(Downsampling)



Skip Connections

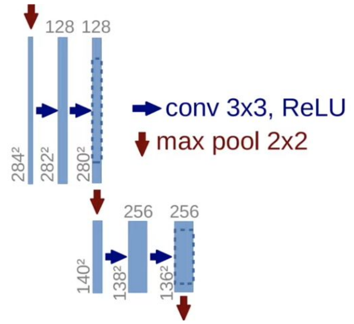
"Bottleneck"



Decoder
(Upsampling)

U-Net Architecture

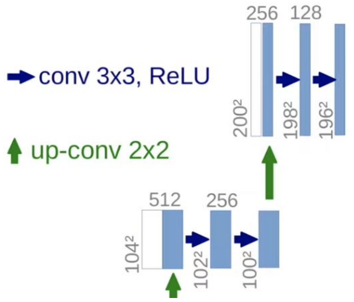
Encoder



Downsampling:

- **Convolutional Layers:** extract and learn hierarchical features from the 3D data
- **Active Functions:** ReLU
- **Pooling Layers:** put between convolutional layers, reduce the dimensions of the volume and downsample the data
- **Increasing Feature Channels:** number of feature channels doubles after max pooling

Decoder

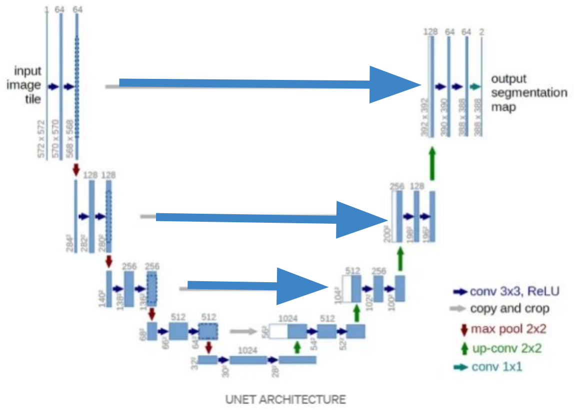


Upsampling:

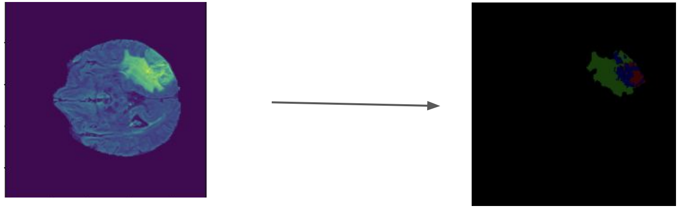
- **Upsampling Layers:** increase the spatial dimensions of the feature maps
- **Convolutional Layers:** process the merged feature maps; refine the upsampled features
- **Active Functions:** ReLU
- **Decreasing Feature Channels:** number of feature channels halves after each upsampling step; transitioning from a deep feature representation to the final segmented output

U-Net Architecture

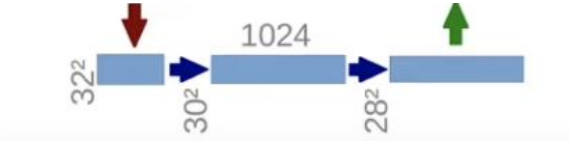
Skip Connections



- Concatenation between symmetrical stages of encoder/ decoder
- Retain high-resolution features and spatial information



Bottleneck

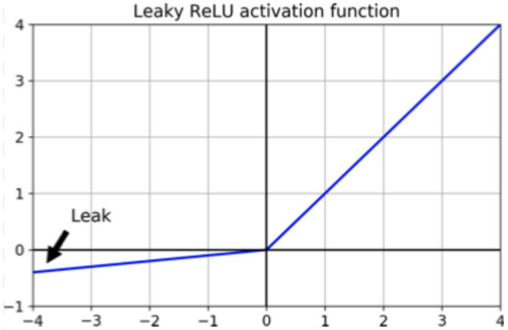


- A narrow and deep layer that connects the encoder and the decoder
- Spatial information from the input image is compressed into a more abstract feature representation.

U-Net Architecture

Activation Function: rectified linear unit ReLU

- Introduces non-linearity into the model that helps the NN to learn more complex data.
- Simple and computationally efficient.
- Sets all negative values in the output to zero, while leaving the positive values as-is.



Loss Function: pixel-wise binary cross-entropy

- Calculates the cross-entropy between the predicted probability distribution and the true probability distribution for each pixel in the image.
- Penalizes misclassification of foreground pixels, more heavily than background pixels.

$$E = \sum_{\mathbf{x} \in \Omega} w(\mathbf{x}) \log(p_{\ell(\mathbf{x})}(\mathbf{x}))$$

Potential Training Strategy

- Stochastic gradient descent with momentum and weight decay.
- Momentum adds a fraction of the previous update to the current update in order to speed up the convergence of the optimization algorithm.
- Weight decay adds a penalty term to the loss function to have the model use smaller weights.