Brain Tumor Detection and Segmentation

An analysis and evaluation of the performance on U-net convolutional networks

Boston University College of Arts and Sciences

Background

Medical field has become one of the most advanced fields in which machine learning and deep learning are applied.

Doctors need higher precision of identification to detect and then address the patient's’ medical problem.

e.g. Tumor Identification

Dataset

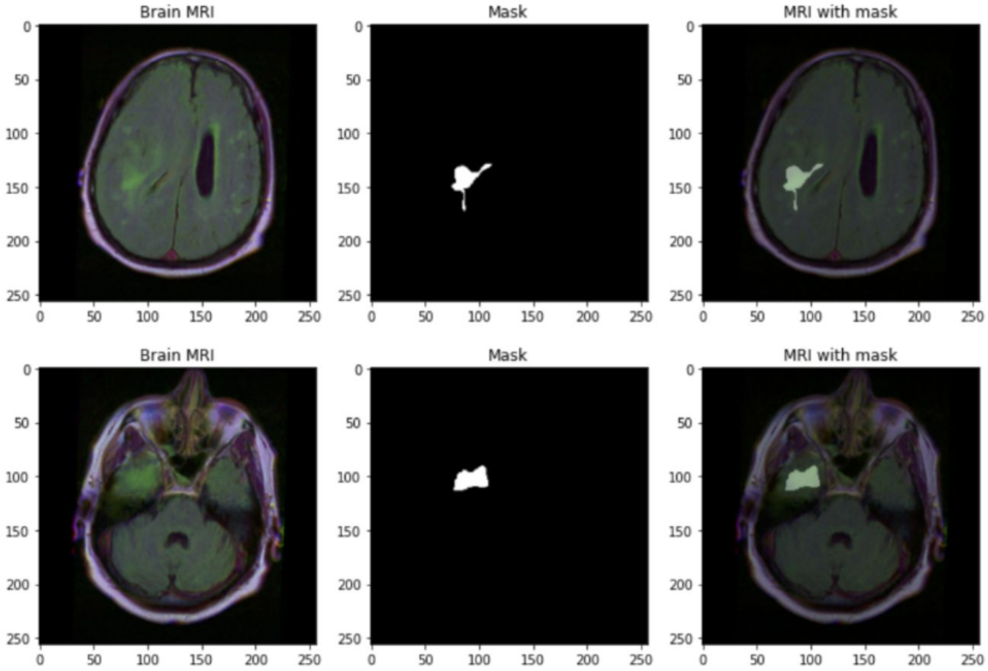
Brain MRI segmentation

https://www.kaggle.com/mateuszbuda/lgg-mri-segmentation

The data-set contains brain MR images together with manual FLAIR abnormality segmentation masks. The images were obtained from The Cancer Imaging Archive (TCIA). They correspond to 110 patients.

The data-set is organized into 110 folders named after case ID that contains information about source institution.

Dataset


Brain MRI
Mask
MRI with mask
0
50
100
150
200
250
Brain MRI
Mask
MRI with mask

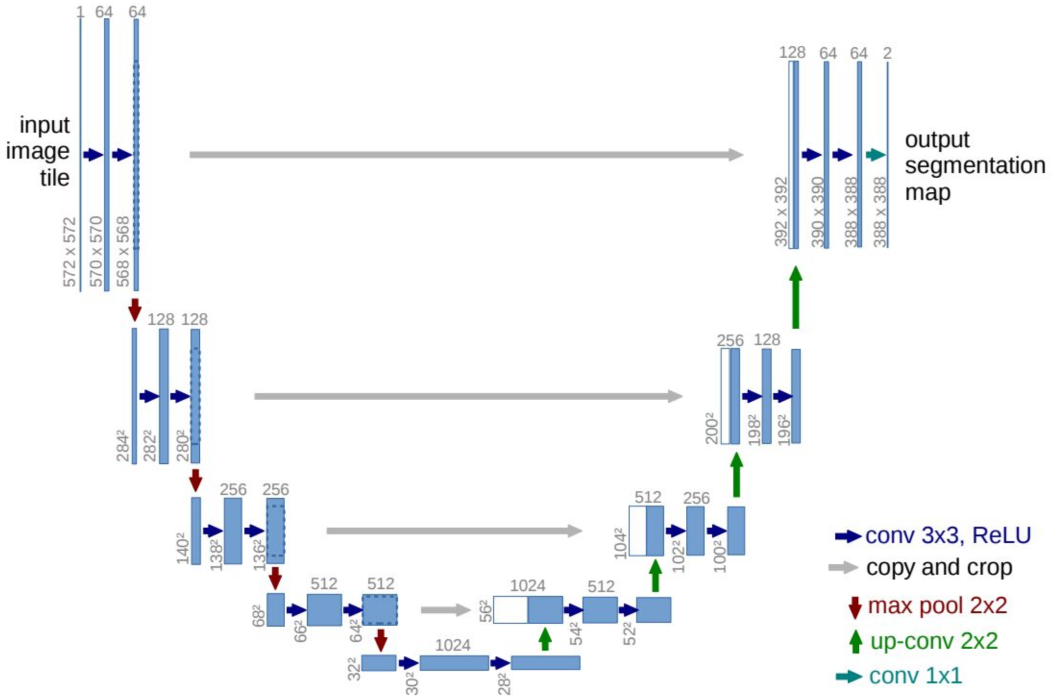
Data Processing

- A train-test split of 80% train and 10% validation, 10% test
- Each image is loaded into a custom dataset and data loader processed with PyTorch

Workstation details

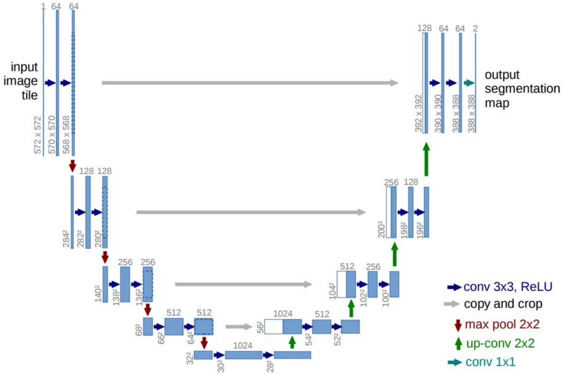
- Hardware
  - BU Shared Compute Cluster
    - 1x Tesla V100
    - 4x CPU Cores
    - 5 hours session
- Software
  - PyTorch 1.10
  - Python 3.8.6
  - Jupyterlab

U-net



input image tile
output segmentation map
1
64
64
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512
256
104²
102²
100²
256
128
200²
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Implementation




```python
# First define the double convolution layer for u-net down
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.conv(x)
```

```python
class UNET(nn.Module):
    def __init__(
            self,
            in_channels=3,
            out_channels=1,
            features=[64, 128, 256, 512],
    ):
        super(UNET, self).__init__()
        self.ups = nn.ModuleList()
        self.downs = nn.ModuleList()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Down part of UNET
        for feature in features:
            self.downs.append(DoubleConv(in_channels, feature))
            in_channels = feature

        # Up part of UNET
        for feature in reversed(features):
            self.ups.append(
                nn.ConvTranspose2d(
                    feature*2, feature, kernel_size=2, stride=2,
                )
            )
            self.ups.append(DoubleConv(feature*2, feature))

        # Lowest layer connecting up and down processes
        self.bottom_u = DoubleConv(features[-1], features[-1]*2)

        # Final convolution layer to output
        self.final_conv = nn.Conv2d(features[0], out_channels, kernel_size=1)

    def forward(self, x):
        skip_connections = []

        for down in self.downs:
            x = down(x)
            skip_connections.append(x)
            x = self.pool(x)

        x = self.bottom_u(x)
        skip_connections = skip_connections[::-1]

        for idx in range(0, len(self.ups), 2):
            x = self.ups[idx](x)
            skip_connection = skip_connections[idx//2]

            if x.shape != skip_connection.shape:
                x = TF.resize(x, size=skip_connection.shape[2:])

            concat_skip = torch.cat((skip_connection, x), dim=1)
            x = self.ups[idx+1](concat_skip)

        return self.final_conv(x)
```

Dynamic learning rate adjustment

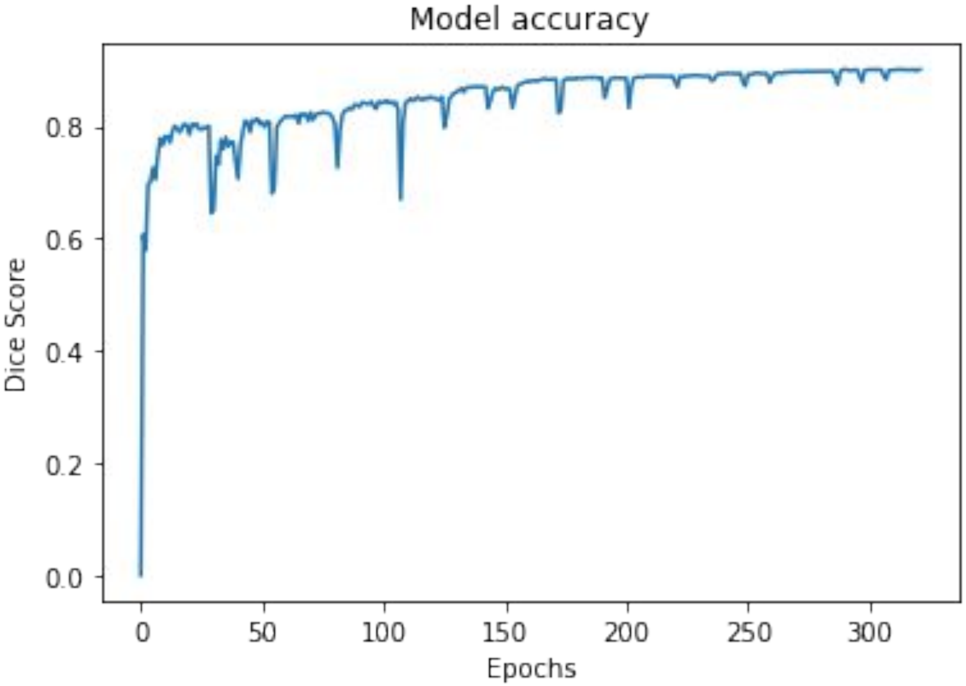
Algorithm 1 Learning rate adjustment

```
learning rate ← 1e-3
highest dice score ← 0
worsen streak ← 0
if worsen streak < 30 then
    train model for one epoch
    dice score ← model dice score
    if dice score > highest dice score then
        highest dice score ← dice score
        worsen streak ← 0
        Save model checkpoint
    else
        worsen streak+=1
        Decay learning rate by one step
        if worsen streak == 5 then
            learning rate ← 1e-3
        if worsen streak == 15 then
            learning rate ← 1e-3
```

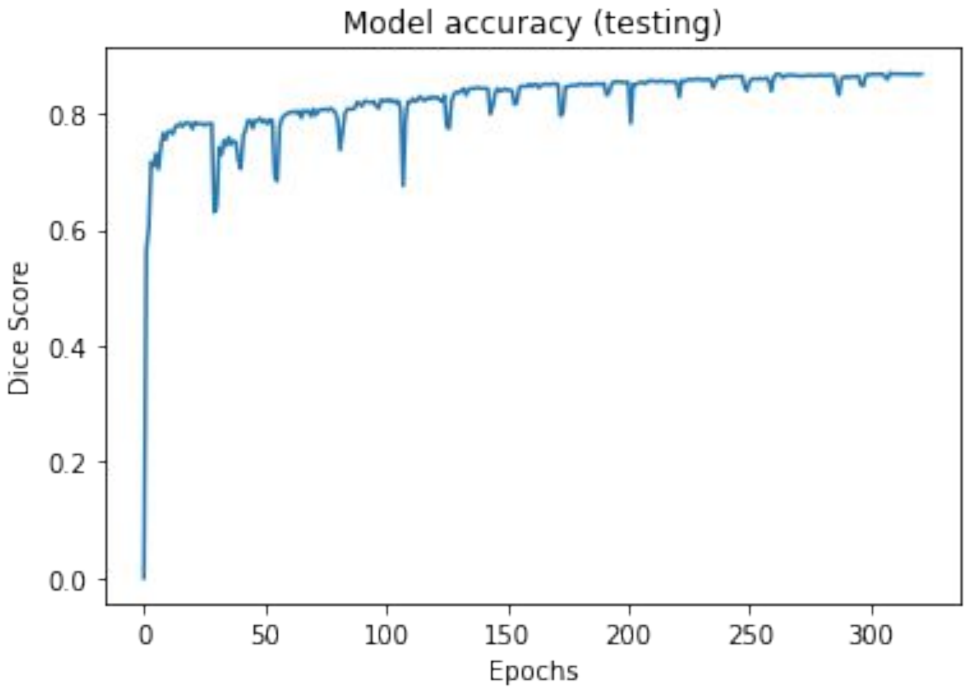
Result measurement

- Key metrics
  - Pixel accuracy
    - Not a good metric
  - Dice coefficient
    - (2 x intersection) / (union + intersection)
    - Emphasizes correct pixel prediction on intersections

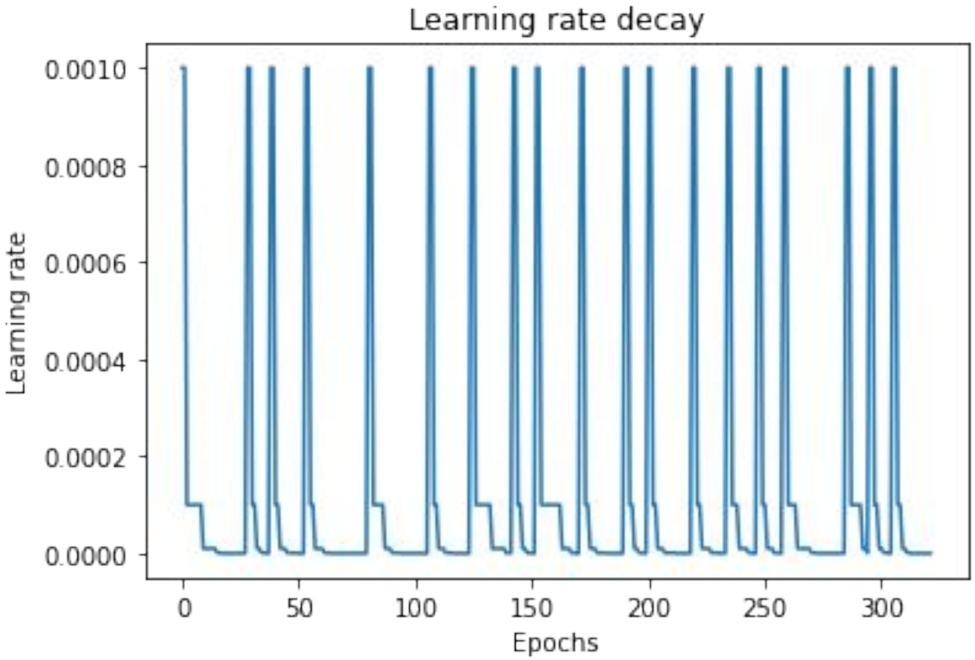
Model accuracy, validation, 91.2%



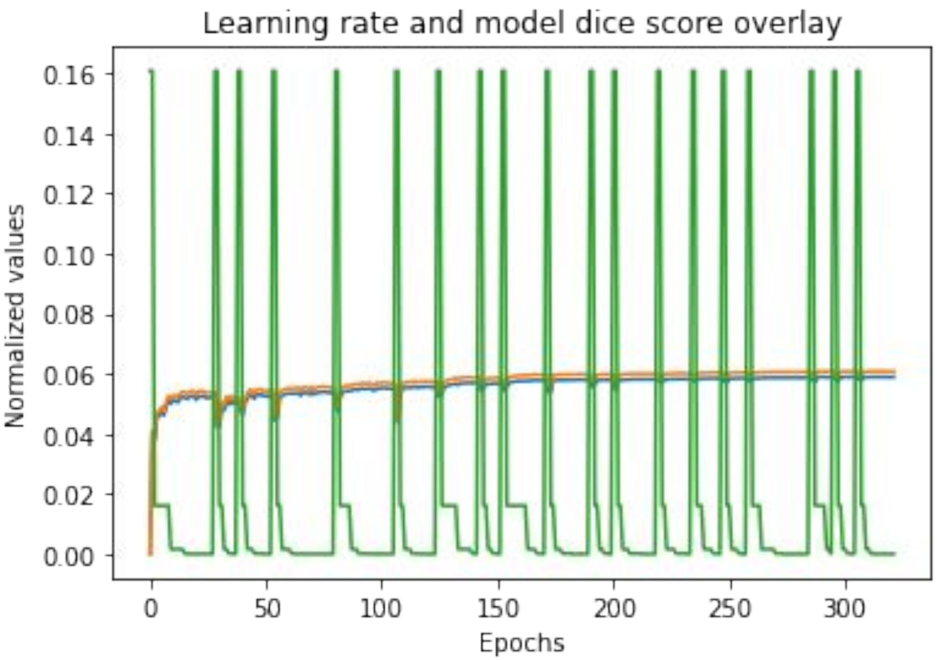
Model accuracy, testing, 86.4%



Learning rate decay



Dice coefficient and learning rate normalized



Previous Achievements

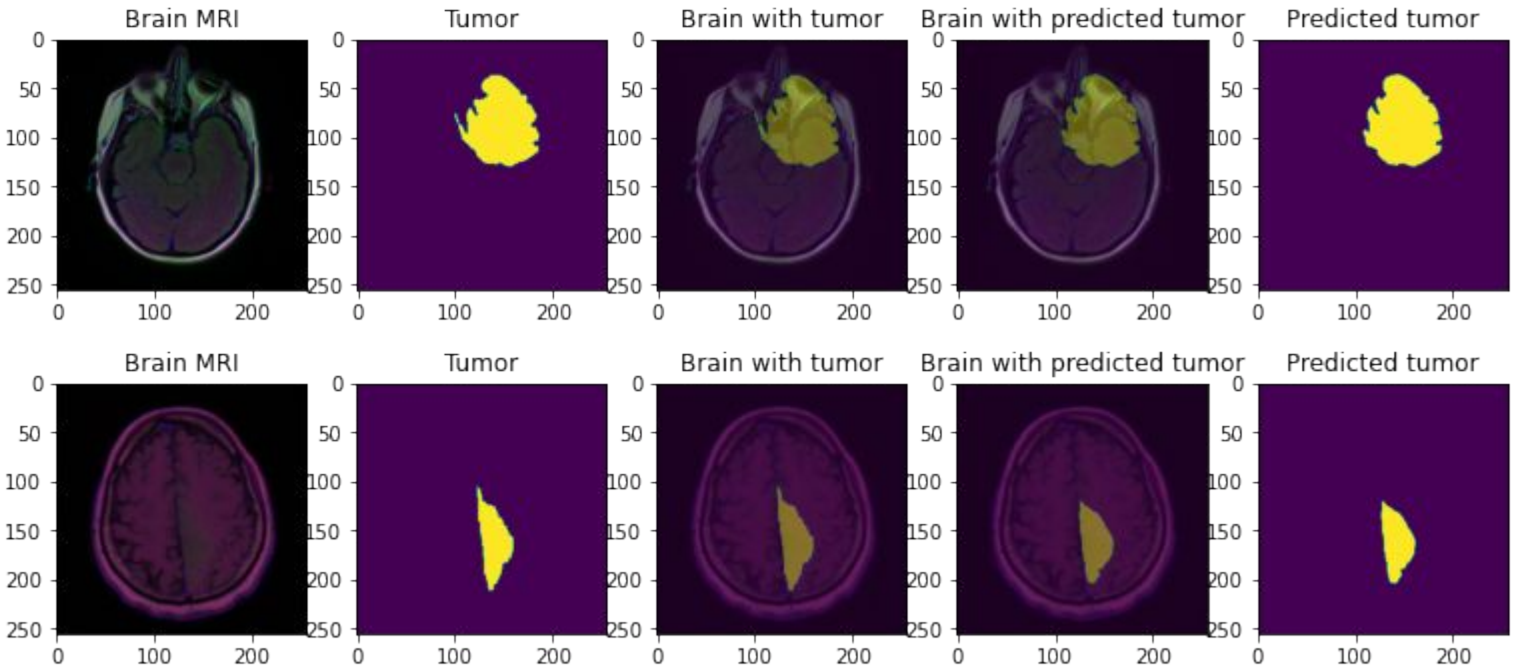
- Mateusz Buda, Ashirbani Saha, Maciej A. Mazurowski. 2019. Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm. Computers in Biology and Medicine (June 2019), 218-225. DOI:https://doi.org/10.1016/j.compbiomed.2019.05.002
  - In addition to segmentation, utilized biological identification to classify characteristics of these tumors
  - Dice coefficient of 82%
  - We achieved 86.4%, improvement of ~4.4%

Model Performance

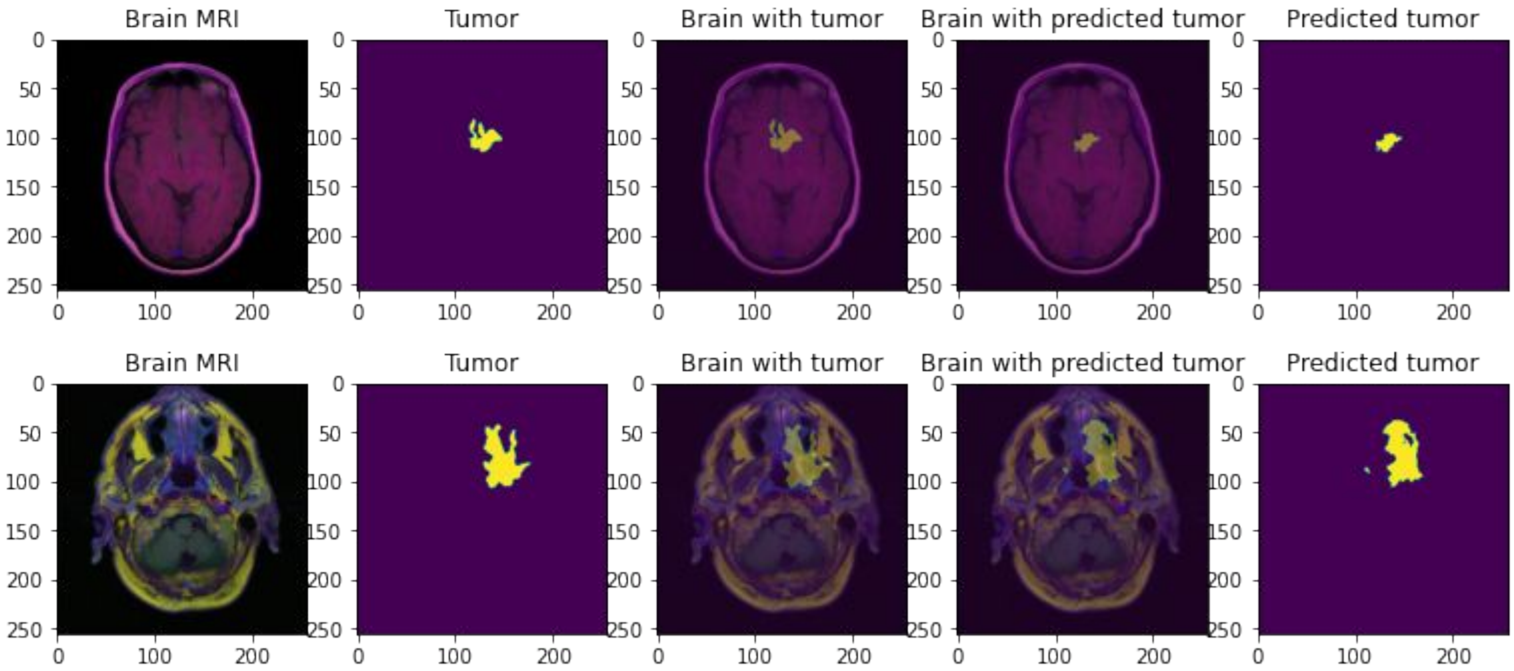
- Categorized problems into three sections

1. Expected performance
2. Excellent performance
3. Incorrect performance

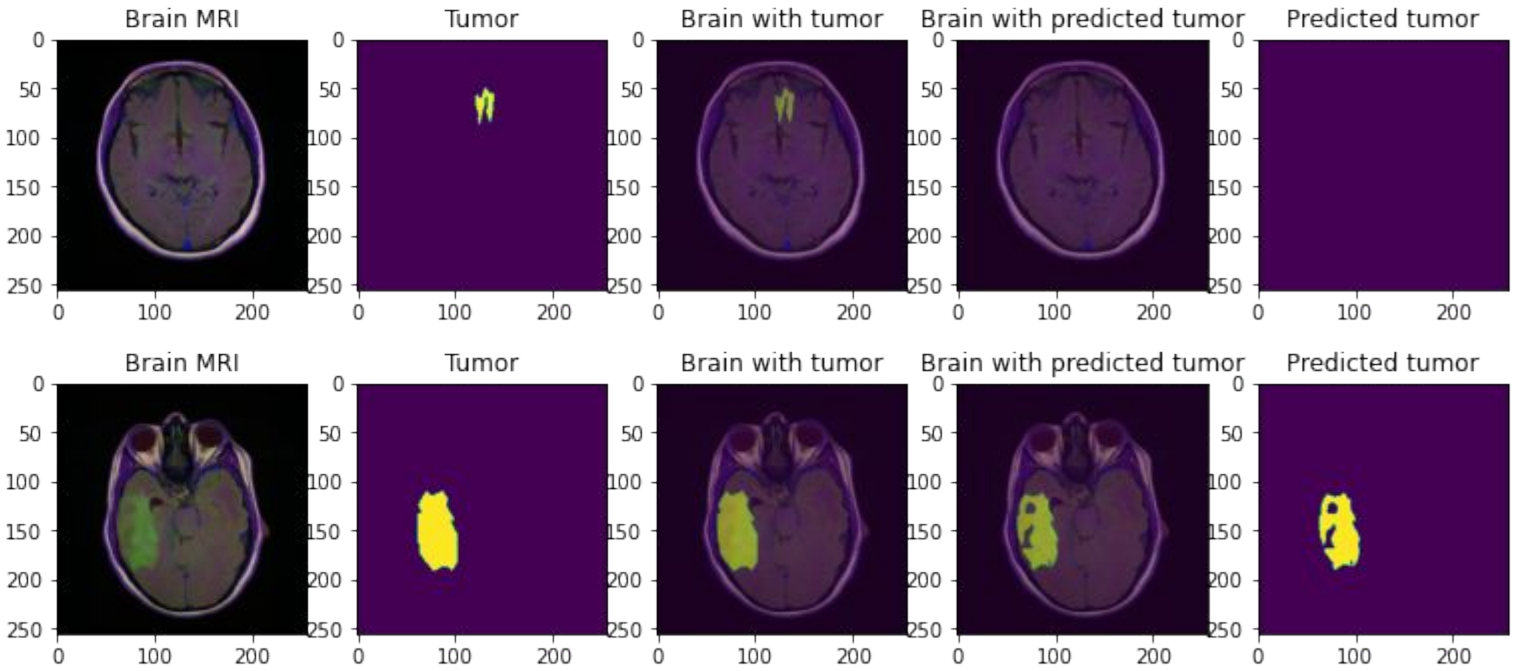
Expected performance



Excellent performance



Incorrect performance



Limitations

- Data size limitations
  - 1007 MB total, 110 patients
- Network architecture
  - Not a “faithful” recreation

Future Work

- Test other model architectures
  - GoogLeNet (didn't have enough time to implement and test)
- Packaging
  - FaaS?
  - Edge deployment?
- Data improvements
- Network modifications

Q&A

Questions are welcome :)

Thank you!