Using Image Classification and Semantic Segmentation for Tumor Detection in MRI Data

Thomas Blue, Frank Burkhart, Brian Manuel

May 2, 2024

1 Introduction

With a recent explosion in available video and image data, the field of computer vision has found profound use in a wide range of applied fields. While much attention towards computer vision applications is paid towards technologies such as autonomous vehicles, facial recognition, and augmented reality, the field which computer vision has had perhaps the most profound impact on is medical imaging [4]. The developments of machine learning in medical imaging has revolutionized the way health professionals and researchers analyze and interpret medical data. Image classification and segmentation serve as the landmark techniques for computer vision in distinguishing between normal and abnormal medical imagery. Our project focused on these two areas of computer vision problems, classification, and image segmentation. We applied these two techniques to a small data set of brain Magnetic Resonance Images (MRI) in order to classify images with and without tumors, and then performed semantic segmentation to distinguish the tumors from the rest of the image.

First, we illustrate a few different machine learning methods for binary image classification of tumorous and non-tumorous MRIs. The dataset we have chosen to work is relatively simple for a first computer vision project, but comes with the challenge of having less than 300 unprocessed MRIs to work with. Despite this small number of images to work with, we implemented data augmentation, regularization, and pretrained models to achieve a test accuracy that is better than our baseline. We use convolutional Neural Networks (CNNs) to do so, implementing both a pretrained model VGG19, and a model we trained ourselves. CNN's have been at the forefront of computer vision, bringing significant advancements in image classification tasks. These networks excel in handling image data and are effective for distinguishing complex patterns in medical images [7].

For image segmentation the U-Net model has demonstrated remarkable success in segmenting medical images with little data needed [3]. The U-Net architecture works by using a symmetric encoder-decoder network that excels in capturing both context and localization, critical for accurate segmentation [5]. Once again the relative lack of size from our data set produced additional challenges given the even fewer number of images with a tumor present to draw from in the data set for the segmentation task.

2 Image Classification

2.1 Classification Data and Processing

For both our image segmentation and classification tasks, we use a public data set of MRI images found on Kaggle [2]. The data consists of 273 axial images of brain MRIs. Each of these images is labeled as one of two classes. Either the MRI has been positively identified as containing a brain tumor, or not. We will first discuss how we prepared the data for binary classification. Following this discussion, we will show how we generated masks for the same dataset to prepare for image segmentation.

While binary classification is a fairly simple task, only 273 images to work with can pose challenges. With such few data points, we will need employ several techniques to achieve high test accuracy and avoid over fitting to our training data. Table 1 shows the class distribution of the dataset, as well as how we split our data from each classes into training and testing sets. Despite the number of images being slightly skewed towards the positive label, we will still consider the baseline model to have 50% accuracy. Thus, we aim to build a classifier that achieves test accuracy greater than this baseline.

Class	Data Split	
	Training	Testing
Yes	124	31
No	77	41

Table 1: Training and testing split by class

We performed exploratory data analysis to detect any abnormalities in the dataset. This was done by ensuring no corrupted images or duplicate image ids were present for any MRI. After these basic checks, we began image processing. The images were very messy. Some came in RGB format, while others were already converted to gray scale. To further complicate things, many images had different resolutions and would need to be resized. Finally, the images were not cropped, which resulted in random artifacts beyond the border of the brain being present in the MRI. The left panel of Figure 1 illustrates a few examples of uncropped images from each class.

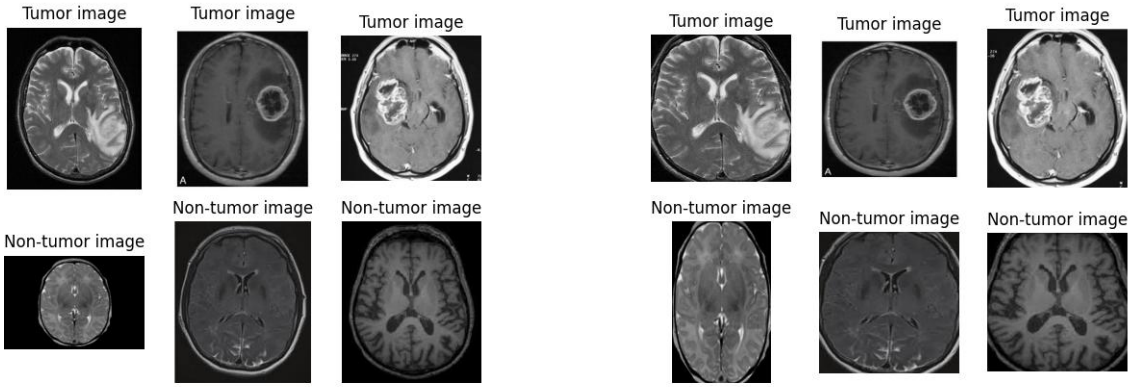

Figure 1: Left: Uncropped, unprocessed example images from both classes. Right: The same images, only now they have been cropped.

In order to fix these issues, we performed several steps of image processing. The first, and most involved step, was to crop each image to only include the extreme points of the height and width of the borders of the brain. This was done using various functions in the CV2 Python package [6]. The results of this cropping are shown in the right panel of Figure 1. We then checked each image to ensure that the cropping was successfully.

Because of the small amount of training data, we used Pytorch transformations to perform data augmentation. The idea here is to slightly alter the images for each batch of training data loaded into the forward pass of our models. This has the effect of increasing the size of the dataset, without actually generating any new images. To do this, we resized each image to $(224, 224)$, performed a horizontal flip and rotation with a 50% probability, applied a Gaussian blur with a kernel size of 3, and then applied normalization. This was done on each crop of training data. For our test data, we only perform the resizing, tensor transformation, and normalization.

2.2 Methods and Results for Classification

Now that we had our data processed and was ready to be augmented, we trained two classifiers. For our first classifier, we implemented a simple untrained CNN with two convolutional layers and two fully connected layers. Each convolutional layer used a 3×3 kernel with a padding and stride of 1. After each of these layers, we applied a ReLU activation function on the output maps and then performed max pooling. We use a cross entropy loss function and an Adam optimizer with a learning rate of 0.01, momentum of 0.9, and a weight decay of 0.0004. The model architecture and parameters are further described in tables 2 and 3.

Layer	Type	Output Size
1	Conv2d(3, 16, 3, 1, 1)	224×224
2	MaxPool2d(2, 2)	112×112
3	Conv2d(16, 32, 3, 1, 1)	112×112
4	MaxPool2d(2, 2)	56×56
5	Linear(32*56*56, 128)	128
6	Linear(128, *2*)	*2*

Table 2: Model Architecture

Parameter	Value
Learning Rate (α)	0.001
Momentum	0.9
Weight Decay	0.0004

Table 3: Adam Parameters

The added weight decay introduces regularization to our model, which is crucial for a dataset as small as ours. We trained the model on 50 epochs and tracked the training and validation accuracy after each epoch. We then plotted a confusion matrix for the final predictions on a test set. The results illustrated in Figure 2.

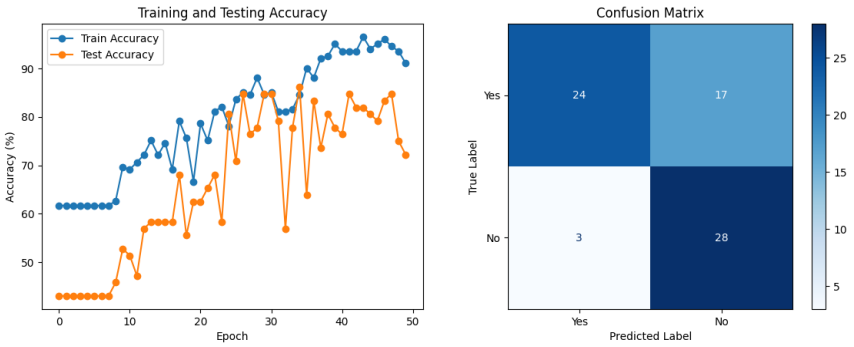

Figure 2: Train and test accuracy of the CNN we trained from scratch.

The results of this model were surprisingly good. It outperformed the baseline model, with both training and testing accuracy tending to increase with more epochs. However, as we can see in the plot of training accuracy against test accuracy, the model test accuracy is highly variable. Further, there is still a decent amount of mis-classified images reported in the confusion matrix. Instead of trying our luck at training this model for a few more hours, we decided to move to a pre-trained model to see if transfer learning could help improve accuracy.

For our pre-trained model, we decided to go with VGG19 from the PyTorch pre-trained models library. We again used a cross entropy loss function with Adam optimization. The same learning rate, momentum, and weight decay were used as in the previous model. To use the model, we froze all pre-trained layers, and changed the head of the model to perform binary classification. We processed our images to be $(224, 224)$, which is the image size VGG19 accepts. We again ran the model with 50 epochs and plotted the results in Figure 3.

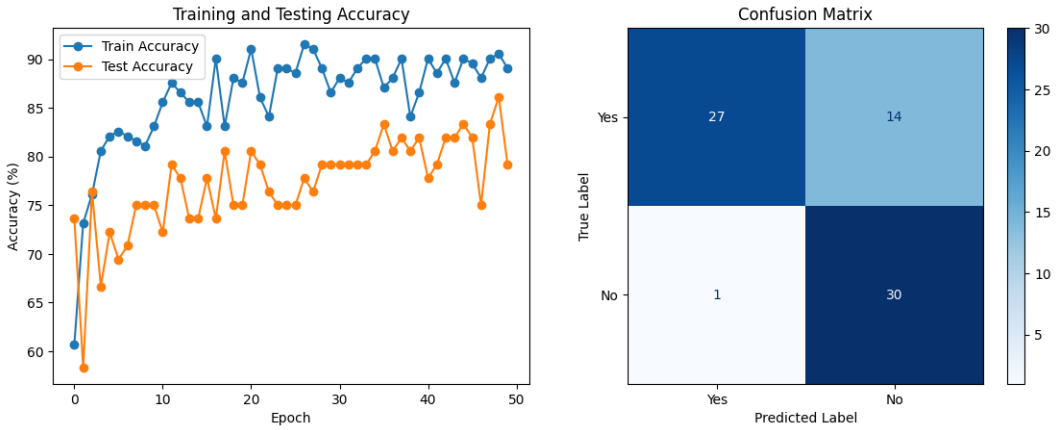

Figure 3: Train and test accuracy using VGG19 pretrained model

Here we can see a huge improvement over the previous model. Specifically, we see less variance in the testing accuracy than that of the previous model. We also see an increase in train and test accuracy as the number of epochs increases. The confusion matrix improves from the previous model, but the model still seems to struggle with producing false negatives. The mean test accuracy for across each epoch is given for both models in given in table 4. We see that both models outperform the baseline, with VGG19 having the highest mean test accuracy.

Model	Mean Test Accuracy
Baseline	50%
Basic CNN	66.22%
VGG19	77.19%

Table 4: Mean test accuracy for each model against the baseline accuracy

3 Image Segmentation

3.1 Data Preparation for Image Segmentation

As previously stated, we worked with the same Kaggle data set of brain MRI's for our segmentation task. In order to perform image segmentation, we first needed to generate masks for our original dataset, since it did not come with any. We manually created our own segmentation masks for these images using LabelBox [1], a tool that allows for precise image masking. To do this we utilized several of LabelBox's tools for mask generation, identifying the boundaries of each image and then accessing the stored data from their cloud server to download and run on our local machines. The masks generated through LabelBox were then utilized as the ground truth data against which the model's predictions are compared.

It is important to note however that none of us are neuroscientists, but we highlighted the tumors in each image as best we could. We illustrate some of our difficulties in Figure 4.

Figure 4: Left: A MRI with with difficulty in creating the mask. Right: A MRI image that was easy to generate the mask for.

Initially, each MRI scan and mask was resized to a resolution of 572×572. This is the image size that U-net takes as input. We then normalized both the brain MRI and the mask and converted each set of images to a gray scale format. This is also required by the U-net model. With this resizing and, normalization, and format transformation complete, the images and masks were now ready to be trained on.

3.2 Image Segmentation Methods and Results

One of the most commonly used models used for medical data image segmentation is U-net. The inputs for this model are images and corresponding masks (in our case brain MRI's and our masks) and the output is the predicted segment (for our case, the brain tumor). For this model, the output is a mapping of binary image of pixels, the same as used for the masks shown above in the data processing section.

The U-Net model comprises three primary components: an encoder, a decoder, and the model core itself. The model is described as follows and in table 5 below:

Encoder: The encoder consists of several blocks, each containing two convolutional layers with ReLU activation followed by a max pooling layer. Each convolutional layer uses 3x3 filters and operates with padding to preserve the dimensions of the input image. The max pooling layers, with a 2x2 filter and stride of 2, progressively reduce the spatial dimensions of the feature maps, effectively compressing the input data and enabling the model to capture essential features at multiple scales.

Bottleneck: The bottleneck works as the transition between the encoder and the decoder. It consists of two convolutional layers with 1024 filters each, again using 3x3 kernels and ReLU activation. This section aims to process the most abstracted features extracted by the encoder.

Decoder: the decoder incrementally reconstructs the spatial dimensions of the output. Each decoder block starts with an up-sampling layer followed by a convolutional layer that reduces the number of filters by half, compared to the preceding block. Each up-sampling step is followed by a concatenation operation with corresponding feature maps from the

encoder, which reintroduces spatial and contextual information lost during downsampling. This is followed by two more convolutional layers with ReLU activation to refine the features.

Output: The final layer of the model is a convolutional layer with a 1x1 kernel. It maps the feature maps to the desired number of output classes—here, just one, using a sigmoid activation function to produce a binary mask indicating the presence or absence of a tumor in each pixel.

Layer	**Type of Layer**	**Output Size**
Input	Input Layer	(256, 256, 1)
Encoder 1	Conv2D + ReLU + Conv2D + ReLU + MaxPool	(128, 128, 64)
Encoder 2	Conv2D + ReLU + Conv2D + ReLU + MaxPool	(64, 64, 128)
Encoder 3	Conv2D + ReLU + Conv2D + ReLU + MaxPool	(32, 32, 256)
Encoder 4	Conv2D + ReLU + Conv2D + ReLU	(32, 32, 512)
Bottleneck	Conv2D + ReLU + Conv2D + ReLU	(32, 32, 1024)
Decoder 4	UpSampling2D + Conv2D + Concatenate + Conv2D + ReLU + Conv2D + ReLU	(64, 64, 512)
Decoder 3	UpSampling2D + Conv2D + Concatenate + Conv2D + ReLU + Conv2D + ReLU	(128, 128, 256)
Decoder 2	UpSampling2D + Conv2D + Concatenate + Conv2D + ReLU + Conv2D + ReLU	(256, 256, 128)
Decoder 1	UpSampling2D + Conv2D + Concatenate + Conv2D + ReLU + Conv2D + ReLU	(256, 256, 64)
Output	Conv2D (Sigmoid)	(256, 256, 1)

Table 5: U-Net Model table. We use an Adam optimizer with learning rate of 0.001 and cross entropy loss function.

After training the model on the brain tumor images with the generated masks, we were able to make image segmentation predictions. We trained two different models with U-net for comparison. The first was a model that had the full dataset but due to heavy computational load, we only ran it with 2 epochs. To speed up training, we tried a second model using only a third of the data set, but ran for 10 epochs. Figure 5 shows the an example result from each of these trained models.

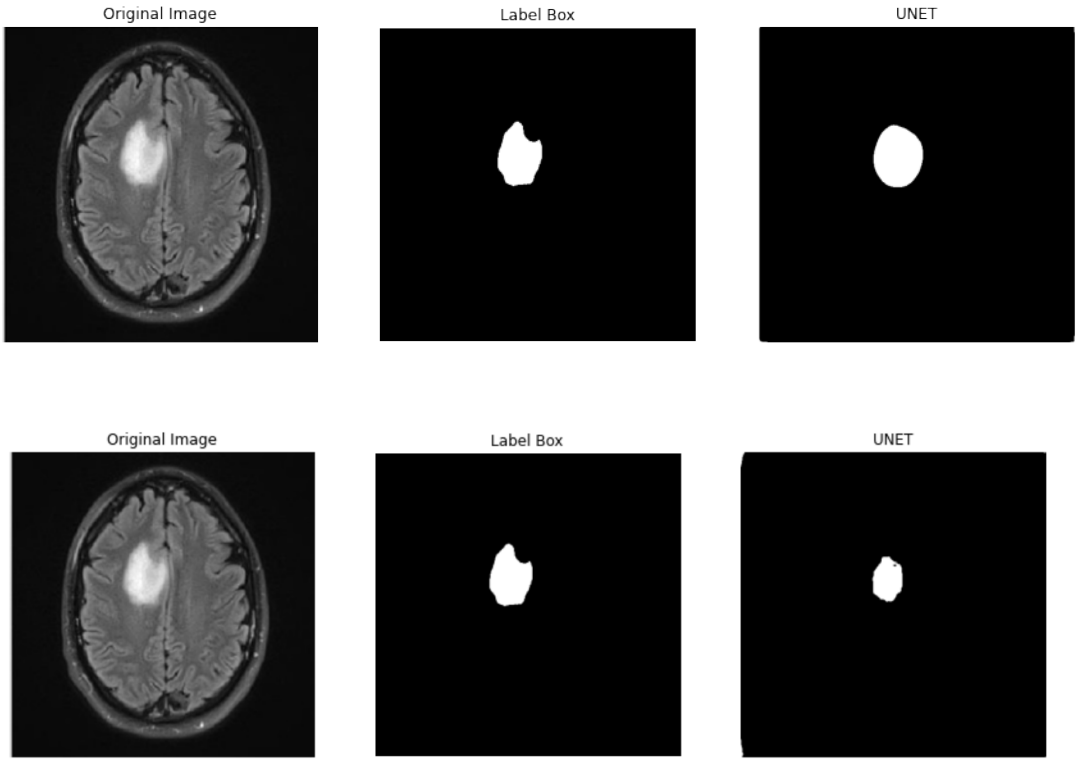

Figure 5: Row 1: Shows the input image, ground truth mask, and U-net prediction trained on entire dataset for 2 epochs. Row 2: Same setup, only now we train with less data and run the model for 10 epochs.

We see from these example outputs that the model ran with fewer epochs and more training data seemed to outperform the model run on ten epochs with few training examples. Of course, this lines up with what you would expect. After further inspection the outputted segmentation from each model, we find that U-net does a decent job at performing semantic segmentation with such few examples.

4 Conclusion

Overall, we are fairly happy with the results for both our binary classification and two dimensional semantic image segmentation tasks. For the binary classification task, we were successfully able to implement crucial data processing steps to prepare our images for each model. We overcame the problem of having a small dataset by utilizing data augmentation, regularization, and using a pre-trained model. Our first CNN had decent results, but variable test accuracy lead to a poor mean testing accuracy score. Improvements were immediately seen after implementing VGG19, with less variability and higher mean test accuracy. In future work, we would like to use a held out validation set to implement hyper parameter

tuning of our learning rate and weight decay.

For the image segmentation task, we were able to generate high quality image masks using the Labelbox software to use to train our model. From this we then were able to run multiple iterations with several different parameters over the data set to generate masks and found that the model with 2 epochs over the entire data set outperformed the model with 10 epochs run over a third of the data set. In future work, we would ideally be able to have expert verification in creation of the masks in addition to recording model accuracy through a measurement such as pixel accuracy or the Jaccard Index. Also working with a different data set with multiple slices of the same tumor we could use the image segmentation techniques to generate a three dimensional segmentation.

References

[1] Labelbox, 2024.

[2] Navoneel Chakrabarty. Brain mri images for brain tumor detection, 2019.

[3] Getao Du, Xu Cao, Jimin Liang, Xueli Chen, and Yonghua Zhan. Medical image segmentation based on u-net: A review. *Journal of Imaging Science & Technology*, 64(2), 2020.

[4] Andre Esteva, Katherine Chou, Serena Yeung, Nikhil Naik, Ali Madani, Ali Mottaghi, Yun Liu, Eric Topol, Jeff Dean, and Richard Socher. Deep learning-enabled medical computer vision. *NPJ digital medicine*, 4(1):5, 2021.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[6] Adrian Rosebrock. Crop image with opencv, January 19 2019.

[7] Hyeon-Joong Yoo. Deep convolution neural networks in computer vision: a review. *IEIE Transactions on Smart Processing and Computing*, 4(1):35–43, 2015.