MRI Brain Classification Final Report

*Nicholas Baker*

*Computer Science*

*University of Guelph*

*Guelph, Canada*

*nbaker05, 1100494*

*Arnab Sagar*

*Computer Science*

*University of Guelph*

*Guelph, Canada*

*arnab, 1088717*

*ABSTRACT–*This project is about the comparison of the performance of four existing algorithms against one another. They will be compared against MRI images of brains, some brains have tumors and others don't. The goal of the models is to classify the differences between the MRIs. The models being chosen are Resnet50, VGG-16, InceptionV3, and YOLOv8. The results that were found were YOLOv8 being the fastest model to classify and VGG-16 being the slowest. In the end, using models is recommended when classifying tumors within the brain.

1. **INTRODUCTION**

Our project is geared towards a thorough comparison of the performance of existing algorithms across diverse applications, primarily focusing on the benchmarking of different algorithms for object detection in images—specifically utilizing MRI images of brains with and without tumors.

The project encompasses several key objectives. Initially, we located a suitable dataset containing brain images, encompassing both instances with tumors and those without. Following dataset acquisition, the data will undergo a meticulous preprocessing phase to ensure its cleanliness and suitability for the models, with a focus on mitigating potential noise within the dataset. Subsequently, we will shift our attention to the selection and understanding of various models employed in object detection. This involves gaining insights into the operational mechanisms of each model, especially concerning the different dynamics of the chosen dataset. The next phase would involve the actual training of each selected model on the prepared dataset. We conducted validation and testing, during which crucial metrics such as processing time, inference time, accuracy, and loss were recorded. Once we had all the results we performed an extensive analysis. Finally, we will compare the results of each of the models.

1. **MOTIVATION AND OBJECTIVES**

The motivation for this project would be the comparison between the performances of existing algorithms for object detection. The focus is on tumor classifications from MRI scans of brains. The reason for this is because we wanted to apply these models to real-world applications. With the medical field being so vast, we thought that MRI images of brains would be a good start.

Our objectives were at first to analyze and compare different models, seeing which ones operated best. From there we realized that was a mighty task and required thought. First off we would need to find MRI images or a dataset to use for training and testing the model. Next, we then preprocess the data to limit the noise and optimize for the models. The dataset is then split into training and validation. Following this we downloaded the models and trained and validated them. When training and validating we will be looking specifically at the accuracy, loss, processing time, and inference time. Finally, we compare the model's results against one another to see if one is better than the others for this application.

1. **METHODOLOGY**
2. Datasets description

The data set that was chosen for the project was [1]. This dataset would hold MRI images of brains that have been split into two folders: images of brains with no tumors and images with tumors [1]. These images would be a top-down view of the brains so we don't have to worry about the noise that would be caused by other positions.

1. Python tools

The tools used for the project were the same as the interim report. Firstly, we are using Google Collab for pair programming which allows us to implement in the same environment at the same time. Google Collab is the same as Jupyter Notebook just within the Google cloud. We are running all code within Python. This was a good choice for implementation as we did not have to worry about hardware limitations. As we would operate and test in the same environment.

1. Libraries

The main libraries we are running are Pytorch, Numpy, Time, and Matplotlib. Numpy, Time, and Matplotlib are used when visualizing the results. Mainly, Matplotlib would be used to plot line graphs, and bar graphs and show the images within the dataset. Time would be used to get the processing time and inference time for each model. Pytorch would be used to train and validate the models. Further Torchvison would be used to get most of the models. Those models being Resnet50, Vgg-16, and InceptionV3. These would be obtained through Pytorch as the integration was built in.

1. Algorithms (briefly explain the algorithms)

ResNet50 is a variant of the ResNet model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer [2]. It has 3.8 x 10^9 Floating points operations. It is a widely used ResNet model. Skip connections, also known as residual connections, are a key feature of the ResNet50 architecture [2]. They are used to allow the network to learn deeper architectures without suffering from the problem of vanishing gradients. In ResNet50, skip connections are used in the identity block and convolutional block. The identity block passes the input through a series of convolutional layers and adds the input back to the output, while the convolutional block uses a 1x1 convolutional layer to reduce the number of filters before the 3x3 convolutional layer and then adds the input back to the output [2].

The 16 in VGG16 refers to 16 layers that have weights [3]. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum up to 21 layers but have only sixteen weight layers i.e., learnable parameters layer. As previously mentioned, VGG16 takes input tensor size as 224, 244 with 3 RGB channels [3]. The most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2. The convolution and max pool layers are consistently arranged throughout the whole architecture [3]. Conv-1 Layer has 64 filters, Conv-2 has 128 filters, Conv-3 has 256 filters, Conv 4 and Conv 5 has 512 filters. Three Connected (FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, and the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer [3].

The algorithm behind the Inception model makes use of the Inception layer, which is a combination of a 1×1 Convolutional layer, a 3×3 Convolutional layer, and a 5×5 Convolutional layer with their output filter banks concatenated into a single output vector forming the input of the next stage. ResNet performs extremely well with deeper architectures, along with deeper network architectures. Replacing bigger convolutions with smaller convolutions leads to faster training.

The algorithm behind the YOLOv8 model consists of a head and a backbone [4]. A modified version of the CSPDarknet53 architecture forms the backbone [4]. This architecture consists of 53 convolutional layers and employs cross-stage partial connections to improve information flow between the different layers. The head of YOLOv8 consists of multiple convolutional layers followed by a series of fully connected layers. These layers are responsible for predicting bounding boxes, objectness scores, and class probabilities for the objects detected in an image [4].

1. Parameter initialization methods

We chose to standardize the parameters across each model. To see the value of the hyperparameters go to the hyperparameter section.

1. Steps taken to preprocess data and build the algorithm

We preprocessed the images by cropping out unnecessary parts of each image. Those being text layered over the images and negative space that only provided noise to the data. We also resized the images for optimized resolution for each model. An example is Resnet50 and Vgg-16 preferring a resolution of 224x224, and Inception preferring a resolution of 229x229. YOLOV8 can handle arbitrary-sized images as long as both sides are a multiple of 32. This is because the maximum stride of the backbone is 32 and it is a fully convolutional network. A further change was made by inversing the colours of the images so brains would be much clearer within the image. This was done since some images had colours blended, which made the separation (contour) between the brain and skull hard to differentiate. Furthermore, the data was transformed and turned into tensors and then further processed using the PyTorch Dataloader, making it more useable as inputs for the models that were chosen.

1. **RESULTS**
2. Hyperparameters

The hyperparameters chosen were as follows. The number of epochs was chosen to be 15, Accuracy didn’t go up from that point onwards. The batch size used was 4 as this would be a standard across all the models. The learning rate was 0.01 across all the models as a standard. The optimizer used was SGD for everything minus the YOLO model. These were the standard hyperparameters chosen to compare the models against one another.

1. Algorithm performance

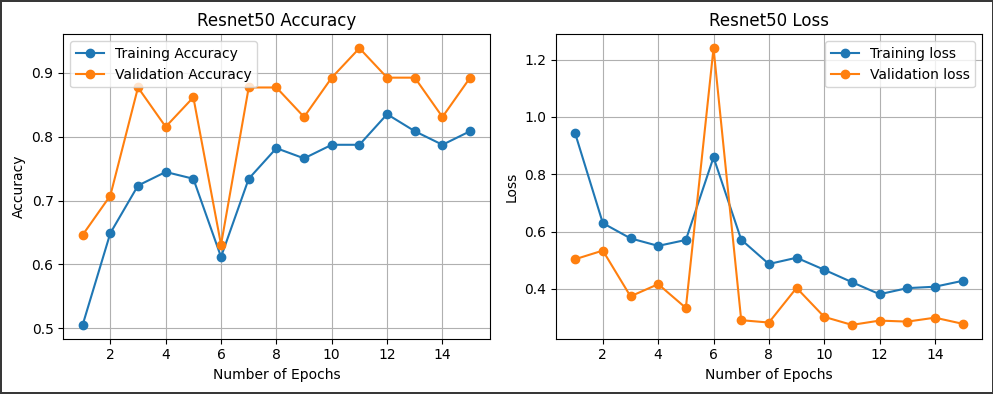


Fig 1. Resnet results

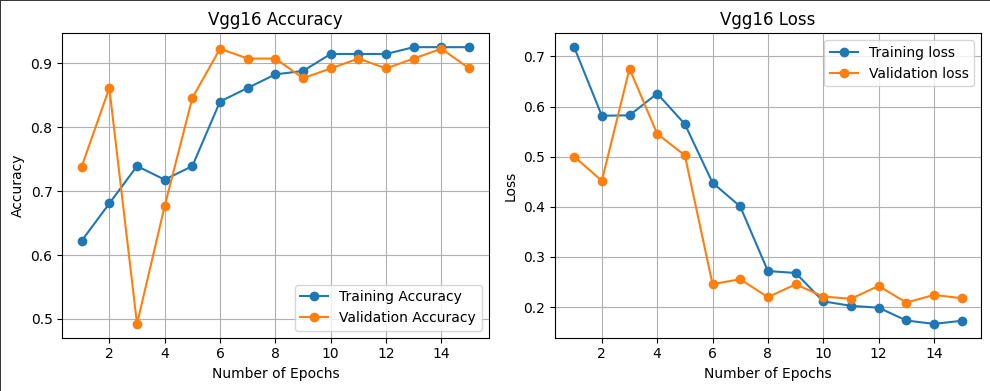


Fig 2. VGG results

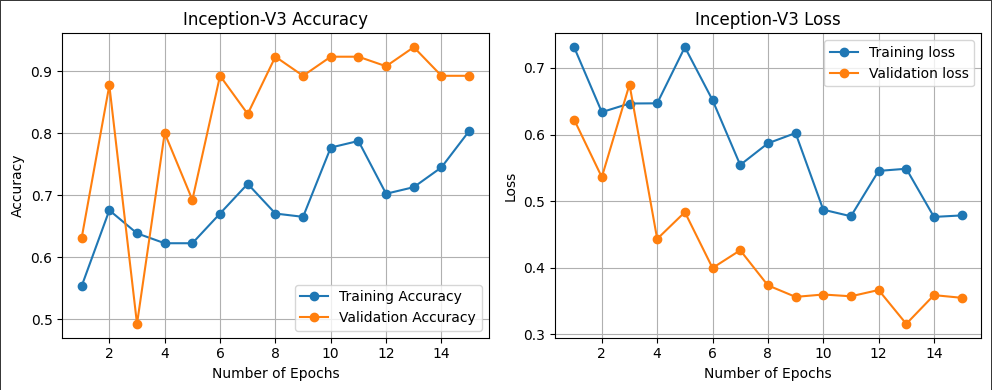


Fig 3. InceptionV3 results

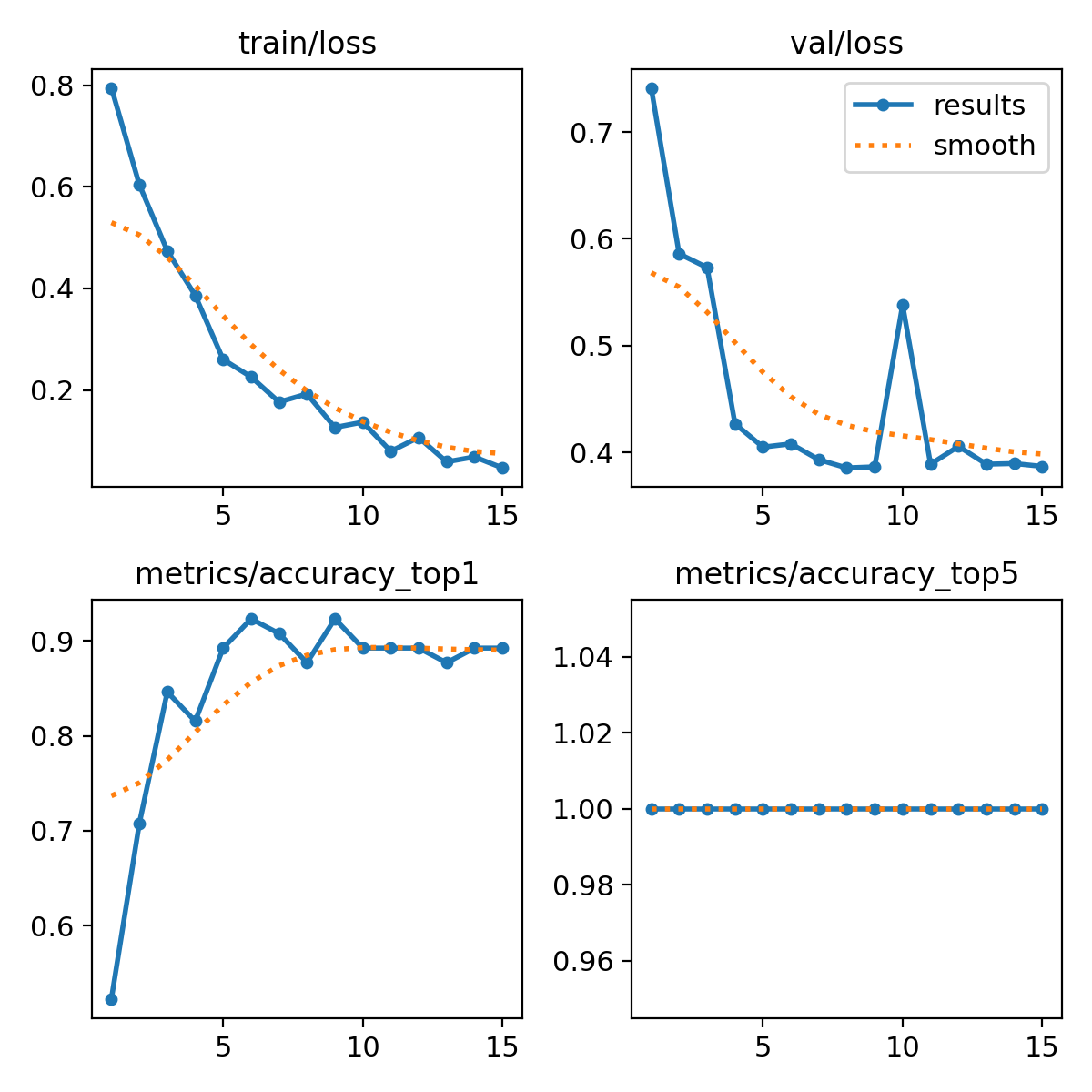


Fig 4. YOLOv8 results

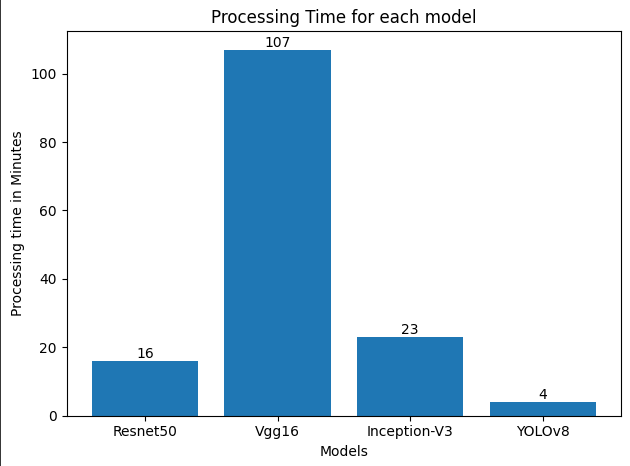


Fig 5. Processing time results

When looking at the results we can see that all the models performed very similarly in accuracy to one another. The range of accuracies went from 50%-93% the more epochs were run. The loss values were also similar to each other as they performed better as they performed more epochs. Inference time is when the values change, Resnet would process training in one epoch at 45 and validate at 15 seconds. Vgg-16 would process training in one epoch at 6 minutes 30 seconds and validate at 40 seconds. InceptionV3 would process training in one epoch at 1 minute and validate at 25 seconds. Finally, there is processing time going from slowest to fastest. The first is Vgg-16 with a time of 107 minutes, InceptionV3 with a time of 23 minutes, Resnet50 with a time of 16 minutes, and YOLOv8 with 4 minutes.

1. Comparative analysis results

The results would show their accuracies were all similar to one another so there is no clear winner when it comes to loss and accuracies. The differences between the models start to show when looking at the timings of one another. Inference time and processing time clearly showed each model's strengths and weaknesses. YOLOv8 would be the fastest model of the bunch showing high accuracy and low processing times.

1. **DISCUSSION**
2. Findings

Looking at the results we can see that the different models perform around the same just with different processing times. The results show that the YOLO model performed the fastest of the bunch, with VGG-16 performing the slowest. This would make sense as to how the models are structured. VGG-16 has around 125 million parameters to adjust for. Similar models like Resnet50 and InceptionV3 have around 25-23 parameters. The results make sense for the models as VGG-16 has the most and longest time, and Resnet50 and InceptionV3 have less showing their times to be similar.YOLO is designed completely differently and shows so by the fact that it takes less time to get high accuracy and low loss values.

1. Interpretation of the results

Given the exceptionally low computation time for the YOLO model, we can argue that YOLO is the best model to leverage for brain tumor classification. This is because it can achieve such a high accuracy score in a fraction of the time. Thinking of the real-world application of having an MRI scan done, in the future you will not need to wait for a doctor to look it over. All you will need is a model to view the data and classify whether there is a tumor. So within our environment, YOLOv8 would be the best model to use in this case.

1. Pros and cons of implemented solution

One of the biggest pros of the implemented solution was the use of transfer learning, which reduced the project’s startup time by several folds. Moreover, the amount of data required and the computational expense for training the model were less, which would not have been the case if we were to train these algorithms from scratch. This holds for all the models we tested.

Now speaking of cons, perhaps the biggest con of this solution is the limited flexibility it offers. While the project deemed desirable results for our use case (classification of simpler MRI images into 2 possible output classes), the approach we took generally offers limited granular control over the algorithms. Hence the same approach might not fulfill a more involved/complex use case. For such complicated use cases, we may need to fine-tune the pre-trained model a bit more or use different layers to adapt the model based on the task.

1. **CONCLUSION**

In conclusion, our study shows that many object recognition models can classify tumors from MRI images. Some prove to be better when taking the time of recognition into account. For our study, YOLOv8 was proven to be the best of the 4 models that were chosen. This was because it achieved a high accuracy with a low loss with minimal time. YOLO was able to get this with a time of 4 minutes as the other models proceeded to take 15-plus minutes to complete. In the end, Object recognition models would greatly improve this area of the medical field.

1. **REFERENCES**

**[1]** Navoneel. "Brain MRI Images for Brain Tumor Detection." Kaggle. Available: <https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection>. Accessed: November 22, 2023.

**[2]** OpenGenus IQ. "ResNet50 Architecture." *OpenGenus IQ*,<https://iq.opengenus.org/resnet50-architecture/>.

**[3]** MyGreatLearning. "Everything You Need to Know About VGG16." *Medium*,<https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918/>.

**[4]** Labeler. "Understanding YOLOv8 Architecture: Applications & Features." *Labeler Blog*,<https://www.labellerr.com/blog/understanding-yolov8-architecture-applications-features/>.

**[5]** Md Ishtyaq Mahmud, Muntasir Mamun, Ahmed Abdelgawad. "A Deep Analysis of Brain Tumor Detection from MR Images Using Deep Learning Networks" Sensors, vol. 16, no. 4, pp. 176, Apr. 2016. Available: <https://www.mdpi.com/1999-4893/16/4/176>.

**[6]** PyTorch. "PyTorch Vision - Pre-trained models for Image Classification," PyTorch. Available: <https://pytorch.org/vision/stable/models.html#classification>.

**[7]** Ritika singh. "Dish Classification using ResNet50 Model with PyTorch," Medium. Available: <https://medium.com/@wularitz/image-classifier-for-dish-classification-using-resnet50-with-pytorch-5d11c02067b5>.

**[8]** Arnab Kumar Roy. "Brain Tumor Classification using ResNet50 with Intel oneAPI Toolkit," Medium. Available: <https://medium.com/@arnabroy02/brain-tumor-classification-using-resnet50-3f8648958799>.

**[9]** MindSpore. "Image Classification Using ResNet-50 Network," MindSpore. Available: <https://www.mindspore.cn/tutorial/training/en/r1.1/advanced_use/cv_resnet50.html#:~:text=ResNet%20is%20recommended.,Second%2C%20it%20is%20learnable>.

**[10]** Piotr Migdał. "Transfer learning with ResNet-50 in PyTorch," Kaggle. Available: <https://www.kaggle.com/code/pmigdal/transfer-learning-with-resnet-50-in-pytorch/notebook>.

**[11]** PyTorch. "Training a Classifier," PyTorch. Available: <https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html>

**[12]** Analytics Vidhya. "CNN Transfer Learning with VGG-16 and ResNet-50: Feature Extraction for Image Retrieval with Keras." *Medium*,<https://medium.com/analytics-vidhya/cnn-transfer-learning-with-vgg-16-and-resnet-50-feature-extraction-for-image-retrieval-with-keras-53320c580853/>.