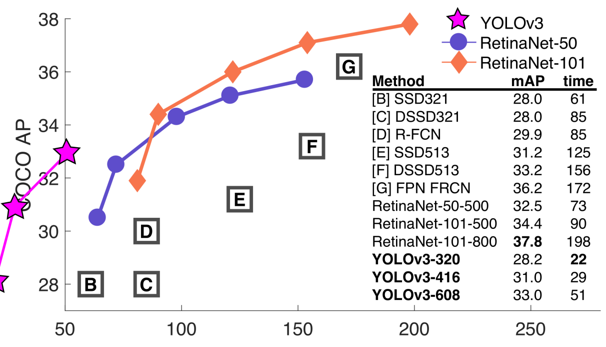
Brain Tumor Detection Using the RetinaNet Model

Jesse Annan
Georgia State University
Atlanta, USA
jesann404@gmail.com

Prof. Yichuan Zhao
Georgia State University
Atlanta, USA
yichuan@gsu.edu

Abstract

This project focuses on applying object detection techniques to a brain tumor dataset using a pre-implemented model. The primary objectives are to understand object detection architectures, preprocess medical images, and evaluate model performance. The methodology involves utilizing the RetinaNet model with a ResNet-50 backbone and Feature Pyramid Network (FPN) from the Detectron2 framework. The dataset comprises of 1,229 preprocessed and augmented images. A key challenge encountered was managing GPU memory constraints in Google Colab, which led to frequent runtime shutdowns. This study serves as a practical exploration of applying state-of-the-art object detection techniques to a critical healthcare problem.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 1: RetinaNet outperforms all previous one-stage and two-stage detectors on the COCO object detection benchmark. YOLOv3 runs significantly faster than other detection methods with comparable performance [11, 17].

1. Introduction

Object detection is a task in computer vision that involves two main components: identifying (classifying) objects and localizing them (drawing bounding boxes around the objects) within an image. In recent years, many automotive companies have been experimenting with autonomous driving technology. For instance, Tesla uses a combination of sensors, including cameras, LiDAR, and radar, to detect and localize the position of pedestrians, road signs, and other vehicles in traffic to determine if it's safe for the car to proceed in its intended direction. Some traffic lights and dedicated speed cameras employ technology to detect where vehicles stop and capture the license plates of speeding vehicles. This technology assists law enforcement in tracking and fining offenders, ultimately helping to protect the safety of other drivers. In retail environments, cameras are used to monitor customer activities. These systems can capture the faces of suspicious individuals, aiding in investigations when necessary. However, it's important to note that the use of facial recognition in retail settings is controversial and subject to varying regulations in different regions. In the medical field, accurately detecting and localizing brain tumors in MRI scans is essential for effective treatment planning and patient care. Despite significant advancements, the complexity and variability of brain tumor appearances pose substantial challenges to developing reliable object detection models. Object detection in medical imaging often requires specialized techniques due to the nature of medical images (e.g., 3D data, grayscale images) and the need for high precision. This project aims to deepen our understanding of object detection techniques by applying them to the specific problem of brain tumor detection. Using the Roboflow brain tumor dataset [20], we will explore an object detection model from Detectron2.

The primary goal is to learn the implementation, training, and evaluation of object detection models in the context of medical imaging. Through this project, we aim to achieve several learning objectives:

1. Gain theoretical knowledge of some novel object detection models,
2. Understand the preprocessing requirements for medical images,
3. Acquire hands-on experience with object detection algorithms, and
4. Evaluate model performance using appropriate met-

rics, such as mean Average Precision (mAP).

This report details our approach, the challenges encountered, and the insights gained, providing a comprehensive overview of the learning process involved in applying object detection to brain tumor detection.

2. Background and Related Work

2.1. Evolution of Object Detection Techniques

During the late 1990s and early 2000s, object detection primarily relied on hand-crafted features. These features were designed by experts to capture specific characteristics of objects, as computers lacked the ability to automatically learn meaningful features from data. Scale-Invariant Feature Transform (SIFT) [13], developed in 1999, is a prominent example of hand-crafted features used in detection. SIFT identifies potential interest points, known as keypoints, in an image and calculates the gradient magnitudes (changes in intensity) in the region around each keypoint. These keypoints are matched with other images for object recognition. SIFT is designed to be invariant to both scale and rotation, meaning it can identify the same features regardless of image size or object orientation. Another hand-crafted feature algorithm is the Histogram of Oriented Gradients (HOG) [2]. The HOG descriptor counts occurrences of gradient orientations in localized portions of an image and represents them using histograms. HOG creates features for object detection by capturing the edge structure and shape of objects within an image. These features are then fed to a Support Vector Machine (SVM), a type of machine learning classifier, for classification and detection. While these hand-crafted methods were effective, they had limitations in dealing with complex scenes and variable object appearances. This led to the rise of deep learning-based methods, which began to dominate object detection in 2014. These methods build upon the Convolutional Neural Network (CNN) architecture, originally developed in 1989 by Yann LeCun et al. [9]. CNNs are deep learning models specifically designed to process grid-like data such as images. They were initially widely utilized by post offices to process mail by recognizing handwritten digits. CNN-inspired object detection models typically have four components or steps:

1. The region proposal step that suggests portions of the image for the model to examine for potential objects,
2. Stacked CNNs for feature extraction, which learns to detect relevant features from the image data,
3. Object *bounding boxes* that localize the extent of a detected object, and
4. The object classification step used to determine the specific object (e.g., cat or dog) that has been localized

This approach, pioneered by the R-CNN [6] (Regions with CNN features) model and its successors, marked a significant shift from hand-crafted features to learned features, greatly improving the accuracy and flexibility of object detection systems.

While there are also approaches that utilize the transformer architecture for object detection, this project focuses on CNN-based object detection models due to their historical importance and widespread use in the field.

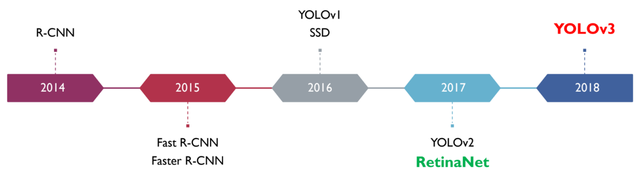


Figure 2: Object Detection Models Timeline.

2.2. Object Detection Models: From R-CNN to YOLOv3

R-CNN [6], introduced in 2014, is credited as the first deep learning-based method for object detection. It operates in multiple stages, beginning with region proposal using selective search. This algorithm generates about 2000 potential object locations by grouping pixels based on color, texture, and other features. Each proposed region is then warped to a standard size and fed through a CNN backbone network – AlexNet [8] – to extract features. These extracted features are used to train separate SVM classifiers for each object class, followed by a regression model that adjusts the bounding box coordinates for better object localization. Finally, Non-Maximum Suppression (NMS) is applied to eliminate overlapping detections, keeping the highest-scoring detection and removing others that overlap significantly based on Intersection over Union. While R-CNN improved accuracy over traditional methods, it was computationally expensive and slow, processing each region proposal separately. Faster R-CNN [18], developed in 2015, aimed to address R-CNN's speed issues while maintaining high accuracy. It processes the entire image once through a CNN to create a feature map. A new component, the Region Proposal Network (RPN), slides over this feature map, proposing potential object regions. At each location, it suggests multiple *anchor boxes* of various sizes and shapes. A Region of Interest (ROI) Pooling layer then extracts fixed-size features for each proposed region from the feature map. Classification and bounding box refinement steps are similar to R-CNN but uses the features from ROI Pooling. Faster R-CNN significantly improved speed by sharing computations and introducing the efficient RPN. You Only Look Once (YOLO) [15], introduced in 2016, brought a single-stage approach that prioritized speed. YOLO divides the image into an $S \times S$ grid, with each grid cell predicting bounding boxes with confidence scores indicating the likelihood of containing an object. Each grid cell also predicts class probabilities, regardless of the number of bounding boxes. All predictions are made in one network forward pass, hence the name "You Only Look Once." While YOLO achieved real-time detection speeds, it struggled with small objects and precise localization. Single Shot MultiBox Detector (SSD) [12], also from 2016, improved upon YOLO's accuracy while maintaining speed. SSD uses several feature maps from different network layers for detection. Instead of YOLO's grid cells, SSD uses *default boxes* of various sizes and aspect ratios at each feature map location. By using multiple feature maps, SSD can detect objects of various sizes more effectively. This approach allowed SSD to achieve better accuracy than YOLOv1, especially for small objects, while still maintaining real-time speed. RetinaNet [11], introduced in 2017, addressed the class imbalance problem in single-stage detectors. This innovative architecture will be discussed in detail in Section 4. YOLOv2 [16], released in 2017, addressed these issues presented in YOLOv1 [15] by improving localization through the use of dimension clusters (anchor boxes) and introducing a new backbone network called Darknet-19 [16]. Building upon these improvements, YOLOv3, launched in 2018, further refined the model with a more powerful backbone, Darknet-53 [17], and introduced multi-scale predictions inspired by FPN [10]. YOLOv3 incorporates several key features that contribute to its performance. It retained the dimension clusters from v2, which uses k-means clustering to determine optimal anchor box sizes. This approach significantly improves object localization by providing better initial guesses for bounding box dimensions. The new Darknet-53 backbone, a 53-layer convolutional network, strikes a balance between depth and efficiency. It outperforms its predecessor Darknet-19 and competes with deeper networks like ResNet-101 [7] or ResNet-152 [7], while being more computationally efficient. YOLOv3 also implements three detection scales, allowing better detection of objects at different sizes. This feature pyramid approach helps address one of the main weaknesses of earlier YOLO versions. Interestingly, the authors found that focal loss, introduced by RetinaNet [11] to address class imbalance, actually reduced YOLOv3's accuracy by 2 points. They concluded that their model's structure, with residual and upsampling layers, already addressed this issue effectively.

The evolution of object detection models (Figures 1 & 2) shows a clear trend towards faster, more accurate, and more efficient architectures. From the multi-stage approach of R-CNN to the single-stage, real-time capabilities of YOLO and SSD, and finally to the balanced approach of RetinaNet, each model built upon its predecessors to address specific challenges in the field of object detection. This progression demonstrates the rapid advancements in deep learning techniques and their application to complex computer vision tasks.

Key Concepts: Bounding Boxes, Anchor Boxes, and Default Boxes

1. *Bounding Boxes:* The final output rectangles that localize detected objects in an image.
2. *Anchor Boxes:* Predefined templates of various sizes and aspect ratios used in two-stage detectors like Faster R-CNN as initial guesses for object locations.
3. *Default Boxes:* Similar to anchor boxes, but used in single-stage detectors like SSD. Applied to feature maps at multiple scales for multi-scale detection.

Anchor and default boxes serve as starting points that models refine to produce final bounding boxes.

2.3. Object Detection in Medical Imaging

Object detection in medical imaging is a crucial application of deep learning, aimed at identifying and locating specific structures or abnormalities in medical images. Recent advancements in this field, particularly in brain tumor detection (Figure 3), have shown promising results. Ezhilarasi et al. [4] showcase how existing object detection algorithms can be adapted for medical imaging. They use Faster R-CNN for tumor detection and classification, implementing preprocessing techniques such as the CAMSHIFT algorithm and Visual Object Tagging Tool. They also apply transfer learning by fine-tuning the pre-trained AlexNet. Their study, conducted on a dataset of 50 MRI brain images, classifies tumors into four types: benign, malignant, glial, and astrocytoma. The authors compared end-to-end training with a four-stage approach, and found that end-to-end training achieved better accuracy and faster inference time. This study illustrates the importance of choosing appropriate algorithms and training strategies when applying object detection to medical imaging tasks. Dipu et al. [3] provide a comprehensive study comparing seven different object detection algorithms for brain tumor detection. They utilize the Brain-Tumor-Progression dataset from The Cancer Imaging Archive, converting images from DICOM to JPG format to leverage CNN-based methods. The authors emphasize the importance of dataset quality, preprocessing steps, and data augmentation techniques in achieving high accuracy. Their study concludes that YOLOv5 (0.95 mAP) shows promising results for real-time brain tumor detection, highlighting the potential impact of automated detection systems on the healthcare industry. While many studies focus solely on tumor classification, Mercaldo et al. [14] address the crucial task of tumor localization. They employ the YOLOv8 (small) object detection model for both detecting and localizing brain cancer in MRI images. Using a dataset of 300 brain MRIs, their method achieved impressive results with a precision of 0.943, recall of 0.923, and mAP of 0.941 at an IoU threshold of 0.5. These results demonstrate the effectiveness of modern object detection algorithms in medical imaging tasks.

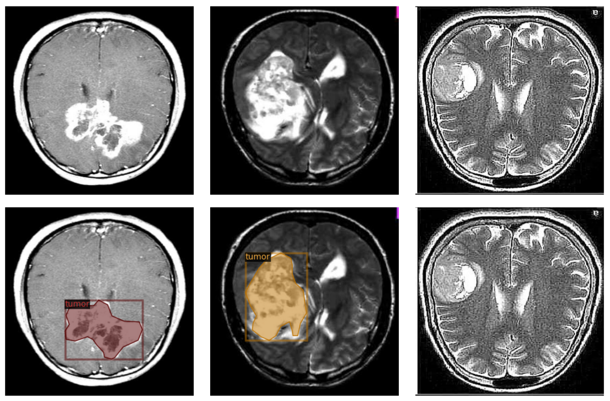


Figure 3: Sample Brain MRI Images. (Top) Original MRI scan of the brain and (Bottom) corresponding annotated version highlighting the detected tumor regions.

3. Dataset and Preprocessing

The brain tumor dataset used in this project was originally created by Zaky Indra [20] and is an open-source dataset uploaded to the Roboflow platform. RoboFlow is a platform that provides tools and services for building, training, and deploying computer vision models. It offers a comprehensive suite of features for handling various aspects of computer vision projects, including dataset management, annotation, preprocessing, augmentation, and model training. The original dataset consists of 1,229 images, which were preprocessed and augmented to create the final dataset used in this study. The dataset contains two classes: tumor and not tumor. After preprocessing and augmentation, the dataset was divided as follows:

- 922 training images
- 184 validation images
- 123 test images

Data Preprocessing and Augmentation: To artificially increase the size of the training set, make the model more robust, and improve generalization, the author employed various data preprocessing and augmentation techniques:

1. Auto-Orient[1]: Applied to ensure consistent image orientation.
2. Resize[1]: All images were stretched to a uniform size of 640×640 pixels.
3. Outputs per training example[2]: 3 augmented versions were created for each original image.
4. Grayscale[2]: Applied to 15% of the images.
5. Blur[2]: Up to 1 pixel blur was applied.
6. Noise[2]: Random noise was added to up to 1.49% of pixels.
7. Flip[2]: Horizontal flipping was applied to images and bounding boxes.

[1]Preprocessing step
[2]Augmentation step

These preprocessing and augmentation techniques were crucial in expanding our original dataset of 1,229 images to a more robust training set of 2,766 images, while also creating appropriate validation and test sets. This expanded dataset allows for better model training and more reliable performance evaluation.

4. Model Selection and Implementation

4.1. RetinaNet

RetinaNet [11] was selected for this project due to its balance of speed, simplicity, and effectiveness in object detection, making it an excellent learning tool. This innovative architecture (Figure 4) builds upon two key concepts: ResNet [7] and Feature Pyramid Network (FPN) [10]. ResNet, developed by Kaiming He et al., introduced the groundbreaking idea of residual layers (also known as bottleneck or skip connection layers), which learn identity functions by adding the input of a block to its output. This clever approach revolutionized deep learning by addressing the vanishing gradient problem that had previously hindered the development of very deep networks [7]. The FPN, on the other hand, enhances feature representation by combining consecutive feature maps, resulting in a more fine-grained and multi-scale feature hierarchy. This multi-scale approach allows the network to effectively detect both large and small objects within an image. On top of these two backbones (ResNet followed by FPN), the authors construct two (specialized) subnetworks: one for classifying objects and another for determining their precise locations in the image. However, the true innovation of RetinaNet lies in its novel loss function, Focal Loss (FL). This function, as explained in the original paper [11], ”is designed to address the one-stage object detection scenarios in which there is an extreme imbalance between foreground and background classes during training”, a problem that two-stage detectors mitigate through region proposal algorithms. Focal Loss is mathematically expressed as

$$FL(p_t) = -\alpha(1 - p_t)^{\lambda} log(p_t) ,$$

where p_t represents the model’s estimated probability for the correct class, α serves as a balancing factor, and λ (optimally set to 2) acts as a focusing parameter that emphasizes misclassified examples. By addressing the foreground-background imbalance, Focal Loss enables RetinaNet to achieve accuracy levels comparable to two-stage detectors while maintaining the computational efficiency of single-stage approaches.

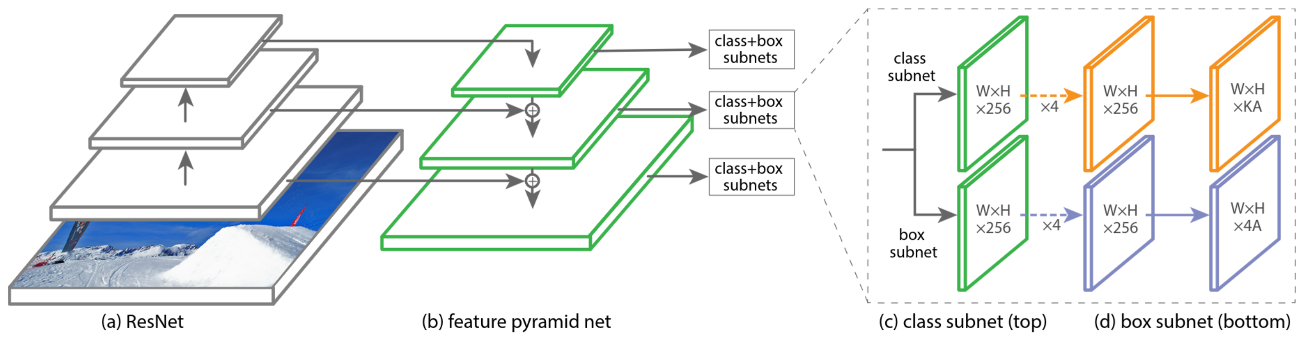


Figure 4: RetinaNet network architecture [11] uses an FPN [10] backbone on top of a feedforward ResNet architecture [7] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). (Source [11])

4.2. Overview of the Implementation Process

To begin with, we set up the environment required for object detection using Detectron2 [19]. Detectron2 is a library developed by Facebook AI Research (FAIR) for object detection, segmentation, and other computer vision tasks. This involved configuring the GPU settings and ensuring the necessary libraries were installed. We used Google Colab Pro, which provides an NVIDIA A100 GPU with 40GB of RAM to expedite the training process. The dataset 3, consisting of 1229 MRI images was downloaded and extracted to the appropriate directories. We used the Roboflow platform to obtain a dataset annotated in COCO format, which is suitable for Detectron2. Prior to annotation by the author, images were preprocessed using standard medical imaging techniques such as contrast enhancement (grayscale and blur) and noise reduction/injection (Section 3. Next, we registered the training, validation, and test datasets using `register_coco_instances`. This step maps the dataset paths to Detectron2's dataset catalog. Figure 3 shows sample MRI images from the training set. We then used a pre-trained RetinaNet model with ResNet-50 as the core backbone (as detailed in Table 1) from Detectron2's model zoo and fine-tuned it on our dataset. The configuration (Listing 2) was adjusted to suit our dataset specifics, including the number of classes (2: tumor and non-tumor) and learning rate. The configuration choices were made based on empirical testing and hardware constraints. Training for 2000 iterations, we used a relatively high learning rate (0.01) to speed up initial convergence, with step-wise decay to fine-tune at later stages (i.e between iteration 1300 and 1700). The batch size of 64[3] was the largest our GPU could handle, balancing between speed and memory usage. One significant challenge we encountered during the implementation was related to GPU memory constraints. Initially, we set a large batch size to speed up training, but this led to runtime shutdowns due to insufficient GPU memory. We addressed this by first purchasing the Google Colab Pro with 200 compute units and gradually reducing the batch size until we found an optimal value of 64, which balanced between training speed and memory usage. Finally, the trained model was used for inference on the test set. The results, including recall and mAP for tumor detection, will be discussed in Section 5.

[3]64 when using A100 40GB GPU and 32 when using L4 23GB GPU

	Type	Filters	Size	Output
	Convolutional	64	7 × 7 / 2	112 × 112
	Max Pooling		3 × 3 / 2	56 × 56
3×	Convolutional	64	1 × 1	
	Convolutional	64	3 × 3	
	Convolutional	256	1 × 1	
	Residual			56 × 56
				fpn route
4×	Convolutional	128	1 × 1	
	Convolutional	128	3 × 3	
	Convolutional	512	1 × 1	
	Residual			28 × 28
				fpn route
6×	Convolutional	256	1 × 1	
	Convolutional	256	3 × 3	
	Convolutional	1024	1 × 1	
	Residual			14 × 14
				fpn route
3×	Convolutional	512	1 × 1	
	Convolutional	512	3 × 3	
	Convolutional	2048	1 × 1	
	Residual			7 × 7
				fpn route

Table 1: ResNet-50 Backbone Architecture [7] with FPN routes

5. Training and Evaluation

5.1. Training Process

We used Google Colab Pro's NVIDIA A100 40GB GPU for model training, with the entire process consuming 39.2 GB of GPU memory. We selected Detectron2's pre-trained `retinanet_50_fpn_3x` model as our base architecture. This RetinaNet model, built on a ResNet-50 backbone with Feature Pyramid Network (FPN), offers a good balance between accuracy and computational efficiency for our tumor detection task. Key aspects of our training configuration include:

```
cfg.DATALOADER.NUM_WORKERS = os.cpu_count - 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url
    ("COCO-Detection/retinanet_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 64
cfg.SOLVER.BASE_LR = 0.01
cfg.SOLVER.MAX_ITER = 2000
cfg.SOLVER.STEPS = (1300, 1700)
cfg.MODEL.RETINANET.NUM_CLASSES = 2
```

Listing 1: Retinanet Configuration

`NUM_WORKERS`: Set to two less than the available CPUs on Google Colab, optimizing data loading without overwhelming the system. `IMS_PER_BATCH`: 64, the maximum number of images per batch our Google Colab session could efficiently process. `BASE_LR` and `STEPS`: ensures optimal precision through adaptive learning rate adjustment (Figure 8). Plus they help fine-tune the saved `WEIGHTS`.

Our training dataset consisted of 1229 MRI images, most annotated with bounding boxes indicating tumor locations. The dataset was split into training (75%), validation (15%), and test (10%) sets. The model was trained for 2000 iterations. We observed a consistent reduction in both bounding box loss (Figure 5) and classification loss (Figure 6), indicating effective learning throughout the training process. The following visualizations illustrates the loss curves:

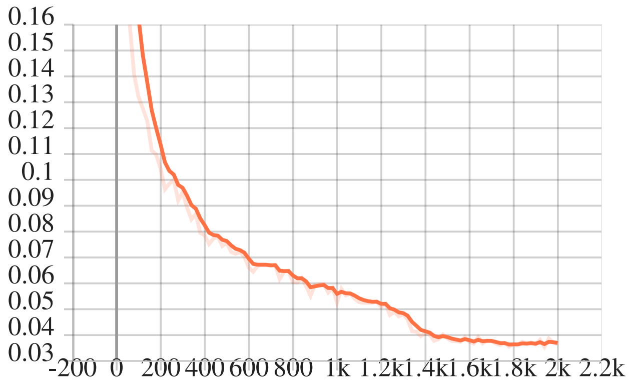


Figure 5: Bounding Box Loss

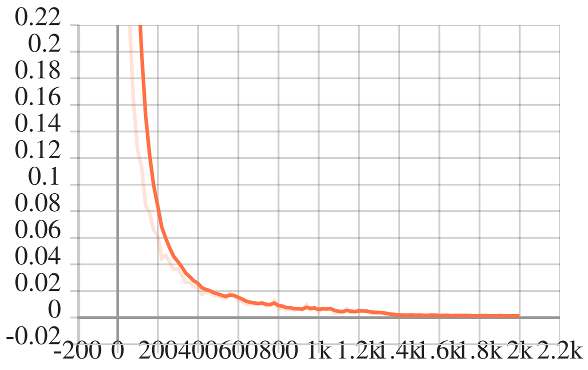


Figure 6: Classification Loss

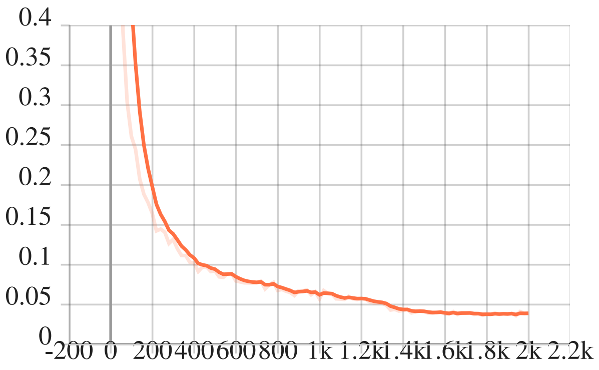


Figure 7: Total Loss

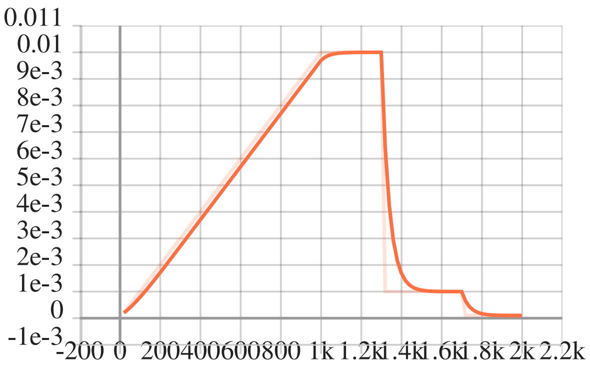


Figure 8: Learning Rate

5.2. Evaluation Metrics

To evaluate the model's performance, we used the following metrics:

- **Recall:** reflects the model's ability to detect all relevant objects. In our context of tumor detection in MRI images, recall represents the proportion of actual tumors that were correctly identified.

$$Recall = \frac{True\ Positives}{True\ Positives\ +\ False\ Negatives}.$$

A high recall is crucial in medical imaging to minimize missed diagnoses.

- **Precision:** measures the accuracy of positive predictions. In our case, it represents the proportion of predicted tumors that are actually tumors.

$$Precision = \frac{True\ Positives}{True\ Positives\ +\ FalsePositives}.$$

High precision is important to minimize false alarms and unnecessary follow-up procedures.

- **Mean Average Precision (mAP):** is our primary evaluation metric, providing a summary of the model's performance across all classes and IoU thresholds. It's calculated by taking the mean of the average precision across all classes and IoU thresholds.
- **Intersection over Union (IoU):** quantifies the overlap between predicted bounding boxes and ground truth bounding boxes. It's crucial for assessing the accuracy of tumor localization.
- **Non-Maximum Suppression (NMS):** is a post-processing technique used to eliminate redundant bounding boxes, retaining only the most confident predictions for overlapping regions.

5.3. Analysis of Results – Test Set

Our test set comprised 123 MRI images, independent from the training and validation sets, representing a diverse range of tumor sizes and locations. We calculate mAP and Recall at various IoU thresholds to examine the model's performance under different strictness levels.

Metric	IoU Threshold			Tumor Size	
	0.5	0.75	0.5 : 0.95	M	L
Recall	–	–	0.856	0.760	0.860
mAP	0.940	0.910	0.797	0.604	0.813

Table 2: Evaluation Scores

The high mAP scores at IoU thresholds of 0.5 and 0.75 indicate excellent performance in detecting tumors with high overlap with the ground truth (Table 2). The overall mAP of 0.797 suggests robust performance across various IoU thresholds. The high overall recall (0.856) demonstrates the model's proficiency in identifying most ground truth tumors. The discrepancy between recall for large (0.860) and medium (0.760) tumors indicates better performance on larger tumors.

6. Learning Outcomes and Discussion

Our initial goals were to gain theoretical knowledge of novel object detection models and acquire hands-on experience implementing an object detection algorithm, specifically YOLOv3, from scratch. We successfully implemented

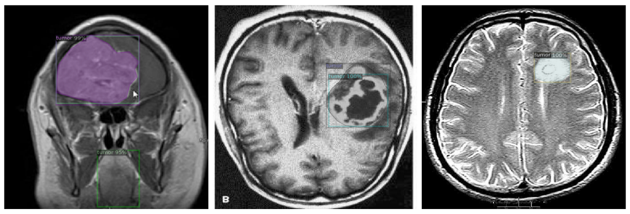


Figure 9: Sample predictions from the test set demonstrating various detection scenarios. (Left) The model correctly identifies a tumor but also incorrectly predicts a tumor in a non-tumor region (green, 95% confidence), illustrating a false positive case. (Middle) The model accurately detects the presence of a tumor, but the predicted bounding box (teal) does not fully encompass the entire extent of the tumor visible in the image. (Right) An example of a perfect prediction, where the model correctly identifies and localizes the tumor with a single bounding box. In other cases, two overlapping bounding boxes cover the tumor region, indicating that the NMS did not fully suppress redundant detections.

the YOLOv3 architecture but faced challenges in the training process. Despite this setback, we achieved three out of four of our initial learning objectives outlined in Section 1. We successfully built the YOLOv3 architecture but encountered significant challenges during the training phase.
Key issues included:

- Difficulty in adapting an existing dataloader to our brain tumor dataset
- Complications in implementing the loss function
- Limited understanding of the complete object detection pipeline, hindering our ability to debug effectively

These challenges provided valuable learning experiences, pushing us to deepen our understanding of deep learning frameworks. Due to the challenges with YOLOv3, we transitioned to using RetinaNet from Detectron2. Our RetinaNet implementation showed promising results, with an mAP of $\sim 80\%$ for tumor detection across all tumor sizes. However, we observed limitations in detecting medium-sized tumors, with mAP dropping to $\sim 60\%$ for this category 5. Throughout the project, we found the following resources particularly helpful:

- Michigan Online's Deep Learning for Computer Vision on Youtube
- Daniel Bourke's pytorch tutorial on Youtube
- Aladdin Persson's GitHub repository for YOLOv3 (Dataloader and loss function) implementation
- Detectron2 documentation and tutorials
- Computer Vision Research papers

This project enabled us to develop several skills such as implementing deep learning architectures from scratch, working with large-scale datasets and GPU acceleration, adapting pre-trained models for custom tasks, and Evaluating and interpreting object detection results. Looking forward, we identify several areas for future work and personal growth including developing a more comprehensive understanding of the entire object detection pipeline, developing a custom loss function, creating or customizing an existing dataloader to our specific dataset, and investigating ensemble methods to improve accuracy.

In conclusion, while we encountered setbacks, this project provided invaluable practical experience and deepened our understanding of object detection techniques. Our implementations of the YOLOv3 architecture and RetinaNet are available on GitHub (MATH8820): `https://github.com/annan92419/GaState/tree/main/grad_projects/` for review and feedback.

7. Conclusion

The model demonstrates strong overall performance, particularly for large tumors. The lower performance on medium-sized tumors may be due to factors such as:

1. Imbalanced representation in the training data
2. Challenges in distinguishing medium tumors from surrounding tissues

Limitations of our current approach include:

1. Potential bias towards larger tumors
2. Limited testing on diverse MRI sequences
3. Runtime shutdonws and reduced training iterations due to limited compute units on Google Colab

Future work could focus on:

1. Fine-tuning the model to improve detection of medium-sized tumors
2. Exploring data augmentation techniques to address size imbalance
3. Testing on a more diverse dataset to ensure generalizability
4. Investigating the integration of 3D information from MRI sequences

Our RetinaNet model demonstrates promising performance in automated tumor detection from MRI images, with particular strength in identifying large tumors. While there's room for improvement in detecting medium-sized tumors, the overall high mAP and recall scores indicate the potential of this approach for assisting in medical image analysis.

References

[1] Roboflow 100. brain tumor dataset. `https://universe.roboflow.com/roboflow-100/brain-tumor-m2pbp`, may 2023. visited on 2024-07-21.

[2] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[3] Nadim Mahmud Dipu, Sifatul Alam Shohan, and K. M. A Salam. Brain tumor detection using various deep learning algorithms. In *2021 International Conference on Science & Contemporary Technologies (ICSCT)*, pages 1–6, 2021.

[4] R. Ezhilarasi and P. Varalakshmi. Tumor detection in the brain using faster r-cnn. In *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2018 2nd International Conference on*, pages 388–392, 2018.

[5] Yousef Ghanem. Brain tumor detection dataset. `https://universe.roboflow.com/yousef-ghanem-jzj4y/brain-tumor-detection-fpf1f`, jul 2022. visited on 2024-07-21.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[9] Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[11] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[13] David Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[14] Francesco Mercaldo, Luca Brunese, Fabio Martinelli, Antonella Santone, and Mario Cesarelli. Object detection for brain cancer detection and localization. *Applied Sciences*, 13(16), 2023.

[15] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object de-

tection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[16] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[17] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[18] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[19] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[20] zaky indra. brain tumor dataset. https://universe.roboflow.com/zaky-indra-w86c4/brain-tumor-gsh0d, apr 2024. visited on 2024-07-16.

A. Appendix

A.1. Project Code File

```
!pip install pyyaml
!pip install 'git+https://github.com/facebookresearch/detectron2.git'
!pip install tensorboard

import os
import cv2
import random
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# Import detectron2 utilities
from detectron2 import model_zoo
from detectron2.config import get_cfg
from detectron2.data import MetadataCatalog
from detectron2.data import DatasetCatalog
from detectron2.engine import DefaultTrainer
from detectron2.engine import DefaultPredictor
from detectron2.evaluation import COCOEvaluator
from detectron2.utils.visualizer import Visualizer
from detectron2.utils.visualizer import ColorMode
from detectron2.evaluation import inference_on_dataset
from detectron2.data import build_detection_test_loader
from detectron2.data.datasets import register_coco_instances


# Download and unzip the dataset
!curl -L "https://universe.roboflow.com/ds/jKDai2wt3U?key=lBM8Mf5pSR" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip


# Register datasets
register_coco_instances("brain_tumor_train", {}, "/content/train/_annotations.coco.json", "/content/train")
register_coco_instances("brain_tumor_val", {}, "/content/valid/_annotations.coco.json", "/content/valid")
register_coco_instances("brain_tumor_test", {}, "/content/test/_annotations.coco.json", "/content/test")


# Visualize training data
my_dataset_train_metadata = MetadataCatalog.get("brain_tumor_train")
dataset_dicts = DatasetCatalog.get("brain_tumor_train")

# Sample 3 images from the dataset
sampled_dicts = random.sample(dataset_dicts, 3)

# Create a figure with 2 rows and 3 columns
fig, axes = plt.subplots(2, 3, figsize=(6, 4))
for idx, d in enumerate(sampled_dicts):
    img = cv2.imread(d["file_name"])
    axes[0, idx].imshow(img[:, :, ::-1])
    axes[0, idx].axis('off')
    visualizer = Visualizer(img[:, :, ::-1], metadata=my_dataset_train_metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    axes[1, idx].imshow(vis.get_image()[:, :, ::-1])
    axes[1, idx].axis('off')
plt.tight_layout()
plt.savefig("brain_tumor_viz.png", dpi=300)
plt.show()


# Configuration
cfg = get_cfg()

# is there a gpu
if torch.cuda.is_available():
    cfg.MODEL.DEVICE='cuda'
    print('gpu available')
else:
    cfg.MODEL.DEVICE='cpu'
    print('Change runtime to GPU, CUDA is required for AMP training!')

cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/retinanet_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("brain_tumor_train",)
cfg.DATASETS.TEST = ("brain_tumor_val",)
cfg.DATALOADER.NUM_WORKERS = os.cpu_count() - 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/retinanet_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 64
cfg.SOLVER.BASE_LR = 0.01
cfg.SOLVER.MAX_ITER = 2000
cfg.SOLVER.STEPS = (1300, 1700)
cfg.MODEL.RETINANET.NUM_CLASSES = 2
cfg.TEST.EVAL_PERIOD = 150

```

```
# Custom Trainer class
class BrainTumorTrainer(DefaultTrainer):
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        if output_folder is None:
            os.makedirs("coco_eval", exist_ok=True)
            output_folder = "coco_eval"
        return COCOEvaluator(dataset_name, cfg, False, output_folder)

# Train the model
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = BrainTumorTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()


# Look at training curves in tensorboard:
%load_ext tensorboard
%tensorboard --logdir output


# Evaluate the model
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.RETINANET.SCORE_THRESH_TEST = 0.7
predictor = DefaultPredictor(cfg)
evaluator = COCOEvaluator("brain_tumor_test", cfg, False, output_dir="./output/")
test_loader = build_detection_test_loader(cfg, "brain_tumor_test")
results = inference_on_dataset(predictor.model, test_loader, evaluator)
results


# Print evaluation results
print("Evaluation results:", results)

# Visualize some predictions and ground truths
dataset_dicts = DatasetCatalog.get("brain_tumor_test")
sampled_dicts = random.sample(dataset_dicts, 3)

# Create a figure with 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(20, 10))

for idx, d in enumerate(sampled_dicts):
    im = cv2.imread(d["file_name"])

    # Ground Truth
    v_gt = Visualizer(im[:, :, ::-1],
                      metadata=MetadataCatalog.get("brain_tumor_test"),
                      scale=0.8)
    out_gt = v_gt.draw_dataset_dict(d)

    # Predictions
    outputs = predictor(im)
    v_pred = Visualizer(im[:, :, ::-1],
                        metadata=MetadataCatalog.get("brain_tumor_test"),
                        scale=0.8,
                        instance_mode=ColorMode.IMAGE_BW)
    out_pred = v_pred.draw_instance_predictions(outputs["instances"].to("cpu"))

    # Combine Ground Truth and Prediction on the same subplot
    axes[idx].imshow(out_gt.get_image()[:, :, ::-1])
    axes[idx].imshow(out_pred.get_image()[:, :, ::-1], alpha=0.5)  # Overlay prediction with some transparency
    axes[idx].axis('off')
    axes[idx].set_title(f"Prediction {idx+1}")

plt.tight_layout()
plt.savefig("predictions_combined_with_gt.png", dpi=300)
plt.show()
```

Listing 2: Complete Project Code

A.2. Unsuccessful Approaches

During our research, we explored several strategies that didn't work (besides the YOLOv3 implementation). These attempts, while unsuccessful, provided valuable insights into the limitations of our resources and guided our final methodology

Expanded Dataset: Our initial plan involved training the model on a substantially larger dataset comprising 9,900 RGB MRI original samples (6,930 for training, 1,980 for validation, and 990 for testing). This Brain Tumor dataset, originally created by Yousef Ghanem [5] and maintained by the Roboflow 100 dataset collection [1], promised a more comprehensive training experience. However, we encountered two significant obstacles:

1. The computational resources and GPU capacity required to store and process this volume of data exceeded our budget constraints.
2. The extended runtime necessary for processing this dataset was incompatible with the limitations of Google Colab, which has a tendency to terminate long-running sessions.

Extended Training Duration: We initially aimed to train the model for a minimum of 5,000 iterations, anticipating that this extended training period would significantly enhance model performance. Unfortunately, we encountered persistent issues with Google Colab terminating our sessions after approximately 2,000 iterations 1 2. This limitation forced us to reduce our training iterations to 2,000 to ensure consistent and complete training runs.