Table of Content

Contents

[Dataset 7](#_Toc198692757)

[Model Performance Analysis Based on Confusion Matrix 7](#_Toc198692758)

[Matrix Components 8](#_Toc198692759)

[Predicted Labels (columns): 9](#_Toc198692760)

[Values in the Matrix 9](#_Toc198692761)

[ Breakdown of the values: 9](#_Toc198692762)

[Interpretation of Performance 10](#_Toc198692763)

[ High True Positives (TP): 10](#_Toc198692764)

[ False Positives (FP): 10](#_Toc198692765)

[Color Gradient 10](#_Toc198692766)

[Conclusion 10](#_Toc198692767)

[Model Performance Analysis Based on F1-Confidence Curve 10](#_Toc198692768)

[Overview of the F1-Score 11](#_Toc198692769)

[Classes Analyzed 11](#_Toc198692770)

[Observations from the Curves 11](#_Toc198692771)

[Overall Performance 12](#_Toc198692772)

[Confidence Levels 12](#_Toc198692773)

[Recommendations for Improvement 12](#_Toc198692774)

[Conclusion 12](#_Toc198692775)

[Model Performance Analysis Based on PR-Curve 13](#_Toc198692776)

[Key Metrics 13](#_Toc198692777)

[Class Performance 13](#_Toc198692778)

[ NO\_tumor (Precision: 0.980) 13](#_Toc198692779)

[ Meningioma (Precision: 0.942) 14](#_Toc198692780)

[ Pituitary (Precision: 0.783) 14](#_Toc198692781)

[ Glioma (Precision: 0.648) 14](#_Toc198692782)

[ Space-occupying lesion (Precision: 0.001) 14](#_Toc198692783)

[Overall Performance 14](#_Toc198692784)

[ All Classes (mAP@0.5: 0.671) 14](#_Toc198692785)

[Summary 14](#_Toc198692786)

[Model Performance Overview 15](#_Toc198692787)

[Train/Box Loss 15](#_Toc198692788)

[Train/Cls Loss 15](#_Toc198692789)

[Train/Dfl Loss 16](#_Toc198692790)

[Val/Box Loss 16](#_Toc198692791)

[Val/Cls Loss 16](#_Toc198692792)

[Val/Dfl Loss 16](#_Toc198692793)

[Metrics/Precision (B) 17](#_Toc198692794)

[Metrics/Recall (B) 17](#_Toc198692795)

[Metrics/mAP50 (B) 17](#_Toc198692796)

[Metrics/mAP50-95 (B) 17](#_Toc198692797)

[Summary 17](#_Toc198692798)

[Model Performance and Quality Metrics 18](#_Toc198692799)

[2. Suggestions to Improve Model Before Inference 18](#_Toc198692800)

[**Dataset** 19](#_Toc198692801)

[**Model Performance Analysis Based on Confusion Matrix** 19](#_Toc198692802)

[Overall Performance 20](#_Toc198692803)

[ Class-Specific Performance 20](#_Toc198692804)

[ Key Observations 22](#_Toc198692805)

[Recommendations for Improvement 22](#_Toc198692806)

[ Address Class Imbalance: 22](#_Toc198692807)

[ Improve Background Distinction: 22](#_Toc198692808)

[ Space-occupying Lesion Detection: 22](#_Toc198692809)

[ Model Architecture: 23](#_Toc198692810)

[ Post-processing: 23](#_Toc198692811)

[**Model Performance Analysis Based on F1-Confidence Curve** 23](#_Toc198692812)

[Overall Model Performance 24](#_Toc198692813)

[Class-Specific Performance 24](#_Toc198692814)

[ NO\_tumor Class: 24](#_Toc198692815)

[ Meningioma Class: 25](#_Toc198692816)

[ Pituitary Class: 25](#_Toc198692817)

[ Glioma Class: 25](#_Toc198692818)

[ Space-occupying lesion Class: 25](#_Toc198692819)

[Key Observations 25](#_Toc198692820)

[ Class Performance Hierarchy: 25](#_Toc198692821)

[ Confidence Threshold Sensitivity: 26](#_Toc198692822)

[ Class Imbalance Effects: 26](#_Toc198692823)

[Recommendations 26](#_Toc198692824)

[ Class-Specific Confidence Thresholds: 26](#_Toc198692825)

[ Address Class Imbalance: 26](#_Toc198692826)

[ Model Refinement: 26](#_Toc198692827)

[ Validation Strategy: 27](#_Toc198692828)

[**Model Performance Analysis Based on PR-Curve** 27](#_Toc198692829)

[Overall Model Performance 28](#_Toc198692830)

[Class-Specific Performance Analysis 28](#_Toc198692831)

[ NO\_tumor (0.954 AP): 28](#_Toc198692832)

[ Meningioma (0.952 AP): 28](#_Toc198692833)

[ Pituitary (0.764 AP): 28](#_Toc198692834)

[ Glioma (0.713 AP): 29](#_Toc198692835)

[ Space-occupying lesion (0.000 AP): 29](#_Toc198692836)

[ Comparative Analysis 29](#_Toc198692837)

[Key Observations 29](#_Toc198692838)

[Recommendations for Improvement 30](#_Toc198692839)

[ Address Class Imbalance: 30](#_Toc198692840)

[ Model Architecture Adjustments: 30](#_Toc198692841)

[**Model Performance Overview** 31](#_Toc198692842)

[Loss Curves Analysis 31](#_Toc198692843)

[ Training Losses 31](#_Toc198692844)

[ Validation Losses 32](#_Toc198692845)

[Interpretation: 32](#_Toc198692846)

[ Precision (metrics/precision(B)): 32](#_Toc198692847)

[ Recall (metrics/recall(B)): 32](#_Toc198692848)

[ mAP@0.5 (metrics/mAP50(B)): 32](#_Toc198692849)

[ mAP@0.5:0.95 (metrics/mAP50-95(B)): 32](#_Toc198692850)

[Generalization & Overfitting Check 32](#_Toc198692851)

[Potential Issues & Recommendations 33](#_Toc198692852)

[ Strengths 33](#_Toc198692853)

[ Weaknesses 33](#_Toc198692854)

[ Next Steps 33](#_Toc198692855)

[Summary Table 34](#_Toc198692856)

[Conclusion 34](#_Toc198692857)

[Analysis of Labels Correlogram for Brain Tumor Detection Dataset 34](#_Toc198692858)

[Distribution of Tumor Locations (x, y) 35](#_Toc198692859)

[Distribution of Tumor Sizes (width, height) 36](#_Toc198692860)

[Correlation Between Parameters 36](#_Toc198692861)

[Potential Issues and Recommendations 37](#_Toc198692862)

[Summary Table 37](#_Toc198692863)

[Actionable Insights 37](#_Toc198692864)

[Summary: 38](#_Toc198692865)

[Task 2: Chapter 1 and 2 of Hands-On Machine Learning with Scikit-Learn, Keras and Tensor Flow 38](#_Toc198692866)

[Exercise: 1 38](#_Toc198692867)

[ How would you define Machine Learning? 38](#_Toc198692868)

[ Can you name four types of problems where it shines? 38](#_Toc198692869)

[ What is a labeled training set? 39](#_Toc198692870)

[ What are the two most common supervised tasks? 39](#_Toc198692871)

[ Can you name four common unsupervised tasks? 40](#_Toc198692872)

[ What type of Machine Learning algorithm would you use to allow a robot to walk in various unknown terrains? 40](#_Toc198692873)

[ What type of algorithm would you use to segment your customers into multiple groups? 40](#_Toc198692874)

[ Would you frame the problem of spam detection as a supervised learning problem or an unsupervised learning problem? 41](#_Toc198692875)

[ What is an online learning system? 41](#_Toc198692876)

[ What is out-of-core learning? 41](#_Toc198692877)

[ What type of learning algorithm relies on a similarity measure to make predictions? 41](#_Toc198692878)

[ What is the difference between a model parameter and a learning algorithm’s hyperparameter? 42](#_Toc198692879)

[ What do model-based learning algorithms search for? What is the most common strategy they use to succeed? How do they make predictions? 42](#_Toc198692880)

[ Can you name four of the main challenges in Machine Learning? 43](#_Toc198692881)

[ If your model performs great on the training data but generalizes poorly to new instances, what is happening? Can you name three possible solutions? 43](#_Toc198692882)

[ What is a test set and why would you want to use it? 44](#_Toc198692883)

[ What is the purpose of a validation set? 44](#_Toc198692884)

[ What can go wrong if you tune hyperparameters using the test set? 44](#_Toc198692885)

[ What is repeated cross-validation and why would you prefer it to using a single validation set? 44](#_Toc198692886)

[Exercise 2: 45](#_Toc198692887)

[Task 1: Implement the code presented in the book and analyze the outputs: 45](#_Toc198692888)

[ Get the data 45](#_Toc198692889)

[ Strated Dataset 47](#_Toc198692890)

[ Data preparation 49](#_Toc198692891)

[ Model Used 50](#_Toc198692892)

[ Random Forest Cross-Validation Scores for trained model: 51](#_Toc198692893)

[ Feature Importance 53](#_Toc198692894)

[ Best hyperparameters found by Randomized Search: 54](#_Toc198692895)

[ Evaluation Results of Fine Tuned Model 56](#_Toc198692896)

[Task 2: Try adding a transformer in the preparation pipeline to select only the most important attributes. 58](#_Toc198692897)

[ Get the data 58](#_Toc198692898)

[ Strated Dataset 60](#_Toc198692899)

[ Data preparation 62](#_Toc198692900)

[ Model Used 64](#_Toc198692901)

[ Random Forest Cross-Validation Scores for trained model: 65](#_Toc198692902)

[ Feature Importances: 66](#_Toc198692903)

[ Best hyperparameters found by Randomized Search: 67](#_Toc198692904)

[ Evaluation of Fine Tuned Model 69](#_Toc198692905)

YOLOv11n Model Training Evaluation Report for Original Dataset

# Dataset

The dataset is downloaded from [roboflow.com](https://universe.roboflow.com/brain-tumor-detection-wsera/tumor-detection-ko5jp)

The dataset includes 1956 images.

Glioma-Meningioma-Pituitary-No are annotated in YOLOv11 format.

The following pre-processing was applied to each image:

\* Auto-orientation of pixel data (with EXIF-orientation stripping)

\* Resize to 640x640 (Stretch)

No image augmentation techniques were applied.

# Model Performance Analysis Based on Confusion Matrix

A confusion matrix is a useful tool for evaluating the performance of a classification model. It summarizes the results of a classification task by comparing the true labels with the predicted labels.

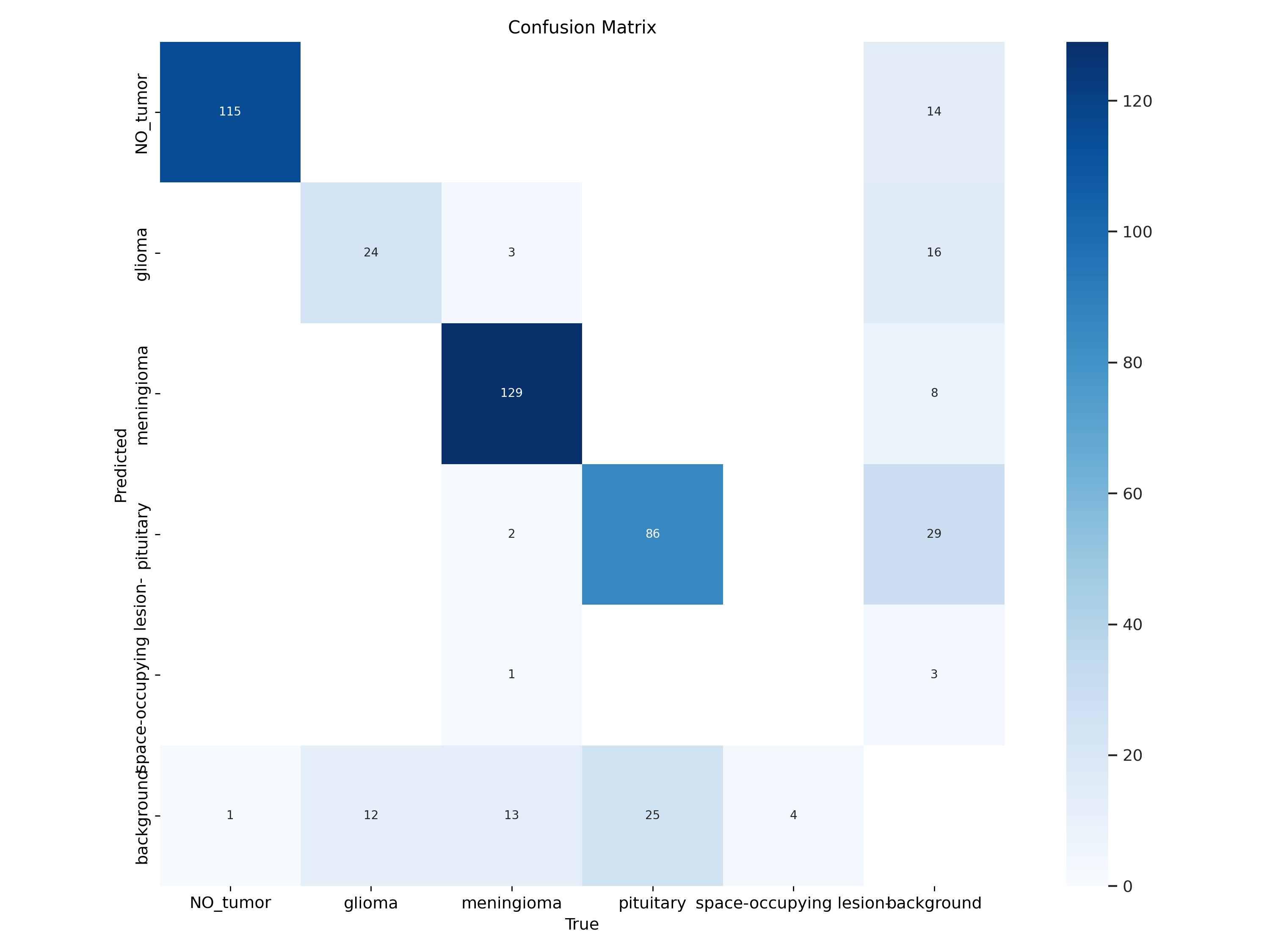


Figure . Confusion matrix

## Matrix Components

- True Labels (rows):

- NO\_tumor

- glioma

- meningioma

- pituitary

- space-occupying lesion

- background

## Predicted Labels (columns):

Same categories as the true labels.

## Values in the Matrix

Each cell in the matrix represents the count of instances for the respective true and predicted classes.

### Breakdown of the values:

#### - NO\_tumor:

- 115 correctly predicted as NO\_tumor

- 14 misclassified as background, and 1 background wrongly predicted as NO\_tumor

#### - glioma:

- 24 correctly predicted as glioma

- 16 misclassified as background, 3 as meningioma, and 12 background wrongly predicted as glioma

#### - meningioma:

- 129 correctly predicted as meningioma

- 8 misclassified as background. 2 pituitary cases, 1 space lesion, and 13 background wrongly predicted as meningioma

#### - pituitary:

- 86 correctly predicted as pituitary

- 29 misclassified as background, and 2 meningioma. 25 background cases wrongly predicted as pituitary

#### - space-occupying lesion:

- o correctly predicted as background

- 3 misclassified as background, 4 wrongly predicted as background

## Interpretation of Performance

### High True Positives (TP):

The model performs well for categories like NO\_tumor and glioma, showing a high number of correctly classified instances.

### False Positives (FP):

Some categories like glioma and pituitary have noticeable misclassifications, indicating areas where the model may need improvement.

## Color Gradient

The color gradient represents the magnitude of values in the matrix, with darker shades indicating higher counts.

## Conclusion

The confusion matrix provides insights into the classification performance, indicating areas of strength and weakness in predicting various tumor types. The data can guide further model enhancements and adjustments.

# Model Performance Analysis Based on F1-Confidence Curve

The F1-Confidence Curve presented in the image provides insights into the performance of the trained YOLO model for brain tumor detection across different classes. Here’s a structured breakdown of the information conveyed by the curve:

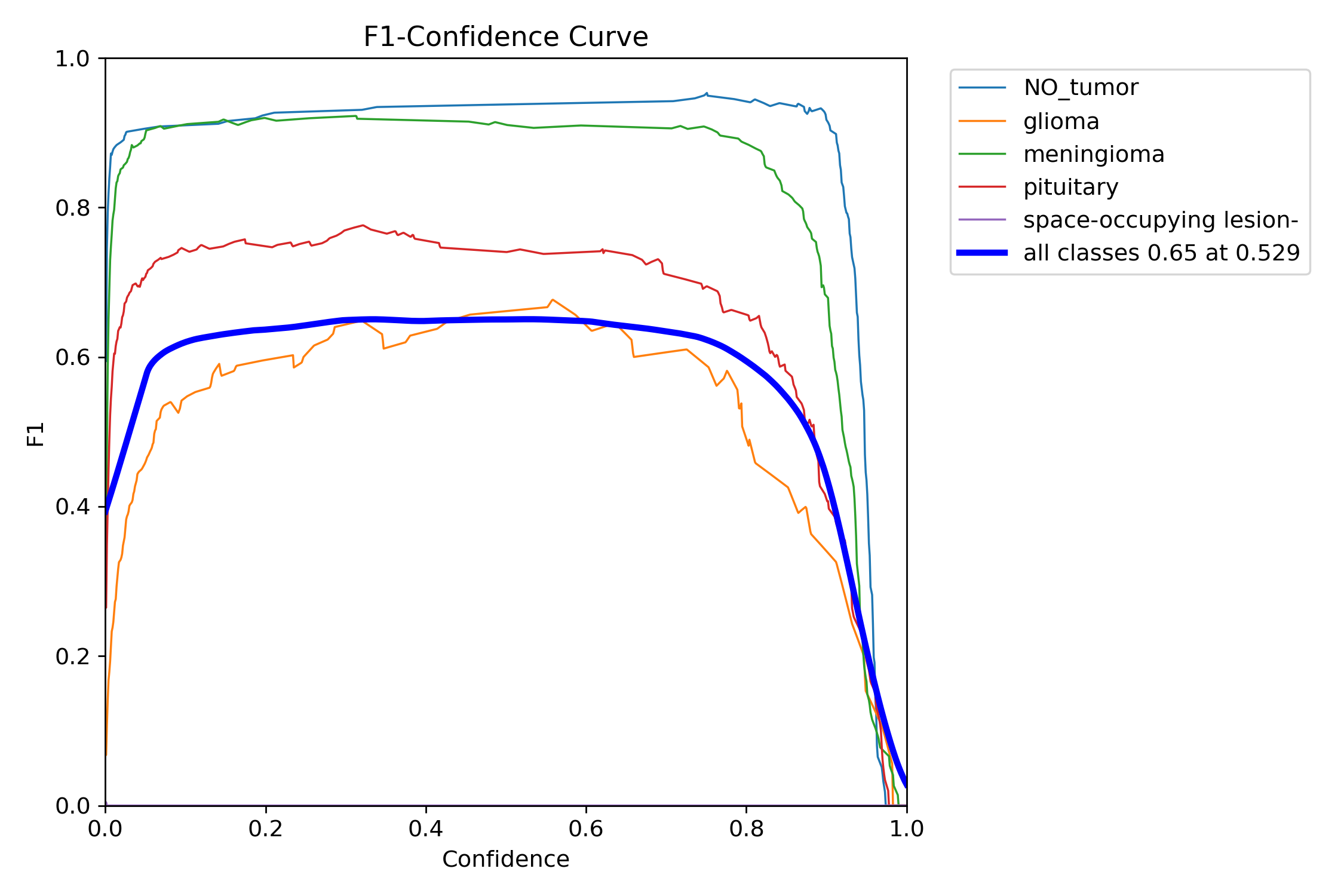


Figure . F1-score is a measure of a model's accuracy that considers both precision and recall

## Overview of the F1-Score

* The F1-score is a measure of a model's accuracy that considers both precision and recall.
* It ranges from 0 to 1, where 1 indicates perfect precision and recall.

## Classes Analyzed

* The performance is evaluated for the following tumor types:
  + NO Tumor
  + Glioma
  + Meningioma
  + Pituitary Tumor
  + Space-occupying Lesion

## Observations from the Curves

* NO Tumor (Blue Line):
  + Shows a strong F1-score, maintaining high values (around 0.85) across multiple confidence levels.
* Glioma (Orange Line):
  + The F1-score is modest, peaking significantly lower than NO Tumor. It suggests that the model struggles to balance precision and recall for this class.
* Meningioma (Green Line):
  + Similar performance to Glioma, peaking below the 0.5 mark.
* Pituitary Tumor (Red Line):
  + F1-score is lower than NO Tumor, and approaches the 0.4 range.
* Space-occupying Lesion (Purple Line):
  + Shows the lowest F1-score, indicating consistent challenges in correctly identifying this class.

## Overall Performance

* All Classes Combined (Thick Blue Line):
  + Achieved an overall F1-score of 0.65 at 0.529 Confidence.
  + Indicates moderate performance in detecting tumors across all classes but also highlights areas for significant improvement, particularly with classes like space-occupying lesions and gliomas.

## Confidence Levels

* At low confidence levels (0.0 to 0.2), precision and recall are notably inconsistent, resulting in lower F1-scores.
* As confidence increases up to 0.65, the F1-scores improve, reflecting the model's capability in correctly identifying tumors.
* Beyond a certain confidence level, the F1-score stabilizes, indicating a threshold where the model has optimized its classification abilities.

## Recommendations for Improvement

* Data Augmentation: Increase data diversity, especially for underperforming classes.
* Model Tuning: Fine-tune hyper parameters to improve recall and precision in low-confidence areas.
* Class Balancing: Ensure class representation is balanced in the training dataset to prevent bias.

## Conclusion

The F1-curve indicates that while the model performs well overall, significant variability exists among different tumor types. Specifically, the NO Tumor class shows a strong predictive capability, while crucial tumor types like gliomas and space-occupying lesions require further optimization for improved detection.

# Model Performance Analysis Based on PR-Curve

The Precision-Recall (PR) curve provides valuable insights into the performance of the YOLOv11n model trained for brain tumor detection. Below is an analysis of the curve based on the provided image.

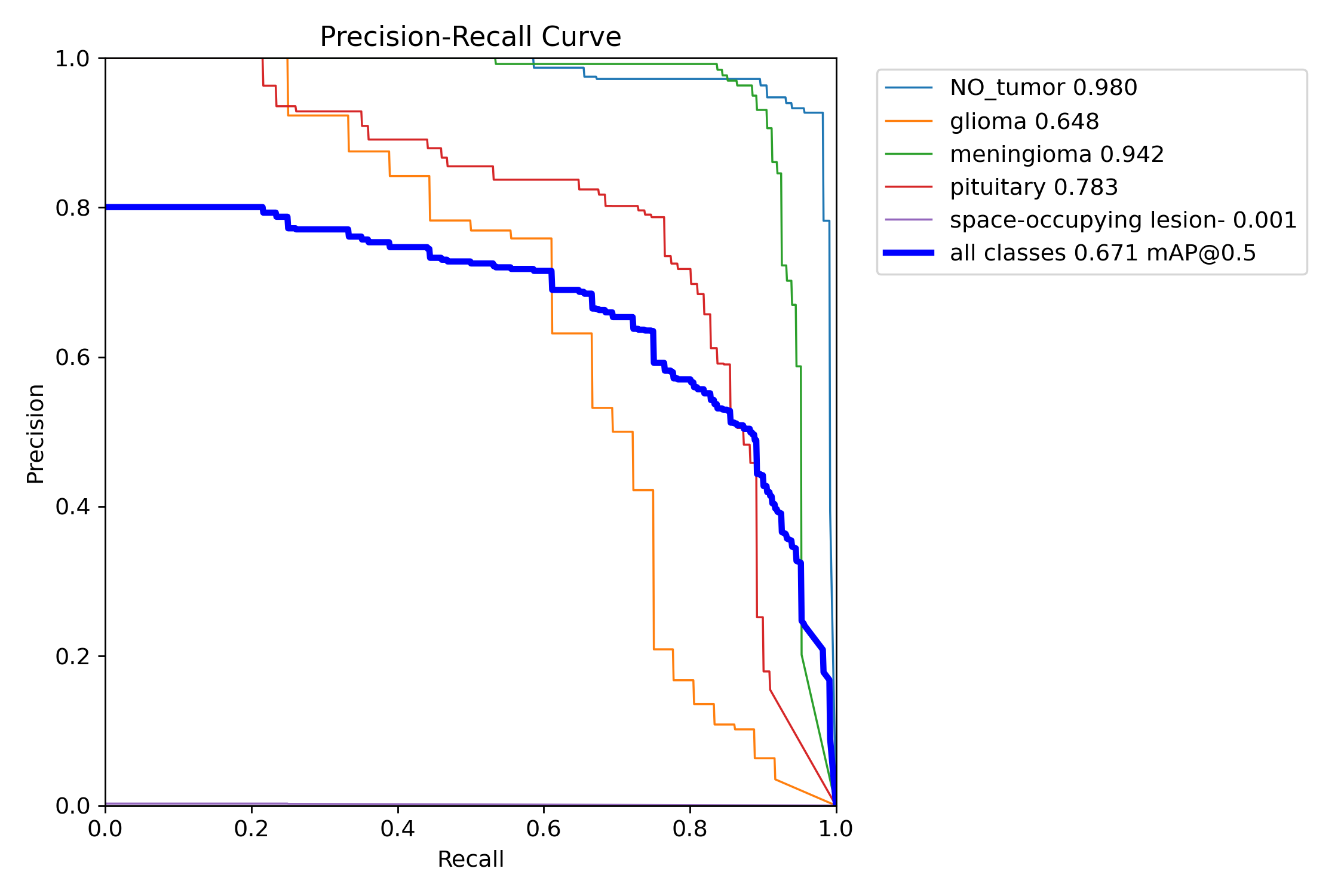


Figure . Precision Recall curve

## Key Metrics

* Each colored line represents a different class of tumors, showing the trade-off between precision and recall at various thresholds.
* The blue line represents the aggregated performance of all classes, noted as mAP (mean Average Precision) at a threshold of 0.5.

## Class Performance

### NO\_tumor (Precision: 0.980)

* + Analysis: Extremely high precision indicates that when the model predicts no tumor, it’s very likely correct.
  + Recall: The model maintains relatively high recall, suggesting it can effectively identify instances of no tumor.

### Meningioma (Precision: 0.942)

* + Analysis: This class also shows high precision, indicating reliable tumor classification.
  + Recall: Similar to NO\_tumor, it suggests good detection of meningioma’s.

### Pituitary (Precision: 0.783)

* + Analysis: Moderate precision indicates some errors in classification, meaning predictions may not be as reliable compared to the first two classes.
  + Recall: The recall for this class will need further exploration to confirm the effectiveness of detections.

### Glioma (Precision: 0.648)

* + Analysis: Lower precision indicates a higher likelihood of false positives.
  + Recall: The ability to detect gliomas may need improvement, as indicated by the lower score.

### Space-occupying lesion (Precision: 0.001)

* + Analysis: This class shows extremely low precision, indicating that the model struggles significantly with this category.
  + Recall: The model likely fails to detect space-occupying lesions adequately.

## Overall Performance

### All Classes (mAP@0.5: 0.671)

* + Analysis: This aggregated score indicates a moderate overall performance. The lower score for some classes suggests room for improvement, particularly in accurately detecting less prevalent types of tumors.

## Summary

The model demonstrates excellent precision for the NO\_tumor and meningioma categories, but struggles substantially with the space-occupying lesion class and shows moderate performance in gliomas and pituitary tumors.

**Future work may involve enhancing data diversity**, exploring model architecture improvements, or **adjusting thresholds to better handle less frequent classes**.

This analysis indicates that while the model performs well with common tumor types, **optimization is necessary for rarer classes**.

# Model Performance Overview

The performance metrics and loss curves for the YOLOv11 model trained on a brain tumor detection dataset show varying trends, providing insights into the training and validation processes.

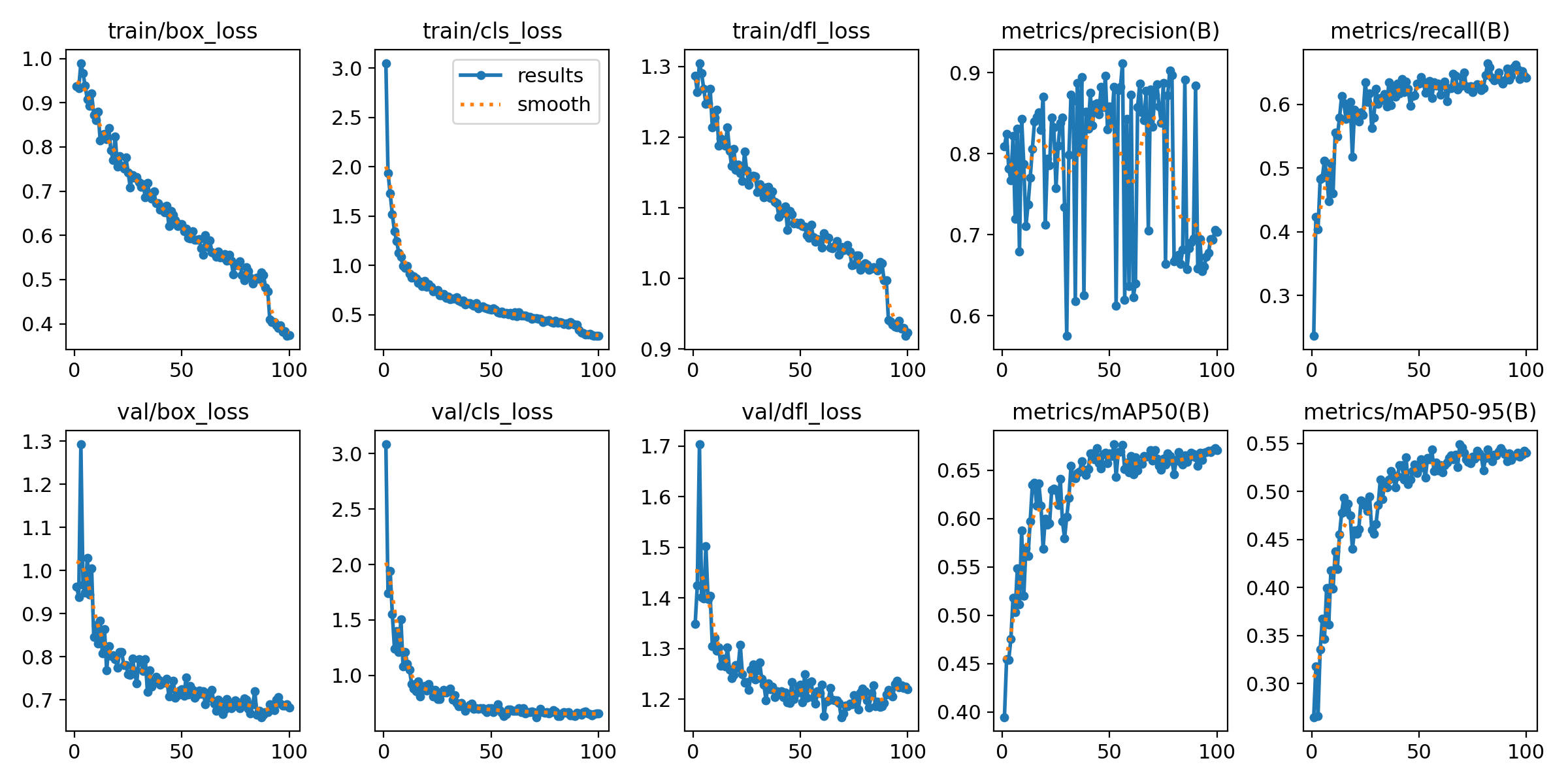


Figure . Model training results: box loss, class loss, dfl loss, metrics vs precision (mAP50), and mAP(50-95)

## Train/Box Loss

* **Description**: Displays the loss related to the bounding box predictions during training.
* **Observation**:
  + Declining trend indicating that the model is learning to make better bounding box predictions.
  + Loss stabilizes closer to 0.4 after about 70 epochs.

## Train/Cls Loss

* **Description**: Reflects the loss associated with classification tasks during training.
* **Observation**:
  + Decreasing loss, initially at around 2.5, dropping to just above 1.0, showing improved classification accuracy.

## Train/Dfl Loss

* **Description**: Represents the distribution focal loss during training.
* **Observation**:
  + Decrease in loss value, stabilizing about 0.9 towards the end of training.
  + Suggests effective learning of object distributions.

## Val/Box Loss

* **Description**: Loss related to bounding box predictions during validation.
* **Observation**:
  + Decreases to about 0.8, indicating good performance on unseen data.
  + Slight fluctuations suggest some variability in performance.

## Val/Cls Loss

* **Description**: Classification loss during validation.
* **Observation**:
  + Shows a declining trend as well, settling around 1.1, which is acceptable indicating reliable classification.

## Val/Dfl Loss

* **Description**: Validation loss for distribution focal loss.
* **Observation**:
  + Slightly higher than training loss, indicating potential difficulties in generalization.
  + Values stabilize around 1.4 by the end of the validation phase.

## Metrics/Precision (B)

* **Description**: Measures the precision of bounding box predictions.
* **Observation**:
  + Fluctuates throughout epochs but generally trends upwards, reaching approximately 0.7.
  + Suggests many correct predictions amid challenges.

## Metrics/Recall (B)

* **Description**: Reflects the model’s ability to correctly identify all relevant instances.
* **Observation**:
  + Steady growth, reaching nearly 0.6, indicating good model comprehensive performance on detection.

## Metrics/mAP50 (B)

* **Description**: Measures mean Average Precision at IoU threshold of 0.5.
* **Observation**:
  + Increases steadily, peaking above 0.5, demonstrating effective bounding box predictions across classes.

## Metrics/mAP50-95 (B)

* **Description**: Averaged mAP over multiple IoU thresholds from 0.5 to 0.95.
* **Observation**:
  + Trends upward, reaching about 0.55, indicating robust performance across different levels of overlap with ground truth boxes.

## Summary

* The model demonstrates a consistent learning pattern, reducing both training and validation losses steadily over epochs.
* Precision and recall indicate the model's reliability in detecting and classifying tumors, with room for improvement in generalization given the slightly elevated validation losses.
* Overall, the performance metrics suggest that the YOLOv11 model is effectively tailored for the brain tumor detection task based on the dataset used.

# Model Performance and Quality Metrics

This section outlines the evaluation metrics recorded during training and validation of the YOLOv8n object detection model. The results are summarized as follows:

• Box Loss (train/val): Steadily decreased, indicating successful learning of object localization.  
• Class Loss (train/val): Rapid initial decline and convergence, showing good classification ability.  
• DFL Loss (train/val): Gradual decrease, demonstrating improved object boundary confidence.

• Precision(B): Shows volatility with fluctuations between 0.6 and 0.95. This may be due to annotation noise or unstable predictions.  
• Recall(B): Steadily increased to around 0.65, reflecting improved object detection coverage.  
• mAP50(B): Increased to approximately 0.68, representing strong detection accuracy for the YOLOv11n model.  
• mAP50-95(B): Improved to approximately 0.54, showing good overall detection performance.

Overall, the model shows no signs of overfitting, with training and validation losses decreasing consistently. The recall and mAP metrics demonstrate progressive improvement in detection capabilities.

# 2. Suggestions to Improve Model Before Inference

To further enhance model precision and reliability before using the model for inference, the following steps are recommended:

• Evaluate Label Quality: Review training dataset annotations for consistency and accuracy. Incorrect or overlapping bounding boxes can reduce model precision.  
• Balance Dataset: Check for class imbalance issues. Ensure all classes are sufficiently represented in the dataset.  
• Tune Confidence Threshold: During inference, consider adjusting the confidence threshold to reduce false positives. For example:  
 yolo detect predict model=best.pt conf=0.4  
• Use a Larger Model: Upgrade to YOLOv11s for potentially better precision and mAP performance if resources allow.  
• Apply Label Smoothing: This can help reduce overconfidence and improve generalization.  
• Perform Post-Training Testing: Run inference on a diverse sample of test images to analyze predictions and identify any specific model weaknesses.

YOLOv11n Model Training Evaluation Report for Dataset 2

**Dataset**

The dataset is downloaded from [roboflow.com](https://universe.roboflow.com/brain-tumor-detection-wsera/tumor-detection-ko5jp)

The dataset includes 1956 images.

Glioma-Meningioma-Pituitary-No are annotated in YOLOv11 format.

The following pre-processing was applied to each image:

\* Auto-orientation of pixel data (with EXIF-orientation stripping)

\* Resize to 640x640 (Stretch)

No image augmentation techniques were applied.

Key Difference: I have converted polygon labels to bounding box labels for all images in train/images. When I started training, yolo logs mention all labels correct except three.

**Model Performance Analysis Based on Confusion Matrix**

Based on the provided confusion matrix for the YOLO11n.pt model trained on a brain tumor detection dataset with 5 classes, here's a comprehensive analysis of the model's performance:

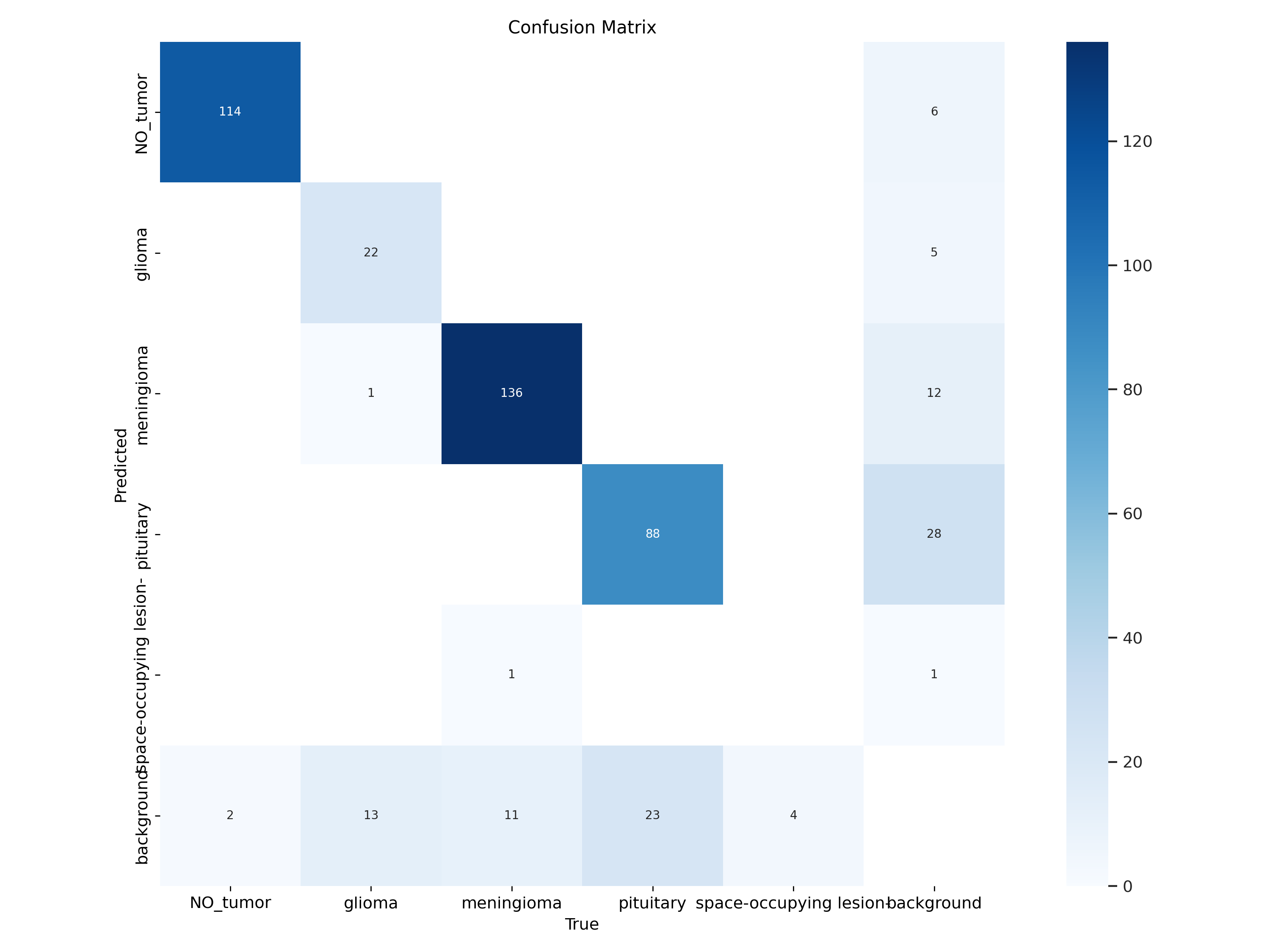


Figure . Model performance analysis based on Confusion Matrix

## Overall Performance

The model demonstrates varying levels of effectiveness across the different tumor classes, **with an estimated overall accuracy of approximately 77%**. The performance is notably strong for certain classes but shows significant weaknesses in others.

### Class-Specific Performance

#### NO\_tumor Class:

* Strong performance with 114 correct predictions out of 116 total cases
* High precision (95%) and recall (98.3%)
* Very few misclassifications, with only 2 cases classified as background
* 6 background cases were incorrectly classified as NO\_tumor

#### Meningioma Class:

* Excellent performance with 136 correct predictions
* High precision (91.3%) and recall (91.9%)
* Main error: 11 meningioma cases misclassified as background
* 12 background cases were incorrectly identified as meningioma

#### Pituitary Class:

* Good performance with 88 correct predictions
* Notable confusion with background class:
  + 23 pituitary cases misclassified as background
  + 28 background cases incorrectly classified as pituitary
* Moderate precision (75.9%) and recall (79.3%)

#### Glioma Class:

* Moderate performance with only 22 correct predictions
* Major issue with false negatives: 13 glioma cases classified as background
* One glioma case misclassified as meningioma
* Precision approximately 81.5% but lower recall around 61.1%

#### Space-occupying lesion Class:

* Very poor performance with only 1 correct prediction out of 5 true cases
* 4 cases were misclassified as background
* Low precision (33.3%) and recall (20%)
* Appears to be the least represented class in the dataset

### Key Observations

#### Class Imbalance:

* The dataset shows significant class imbalance with meningioma (approximately 148 samples) and NO\_tumor (approximately 116 samples) being the most common
* Space-occupying lesion class is severely underrepresented with only 5 samples

#### Background Misclassification Pattern:

* A recurring issue is the model's tendency to confuse actual tumor classes with background
* Approximately 53 actual tumor cases were classified as background
* About 52 background cases were incorrectly classified as tumor types

#### Confusion Between Classes:

* Very little inter-class confusion between tumor types
* Most misclassifications occur between a specific tumor class and the background

## Recommendations for Improvement

### Address Class Imbalance:

* + Implement data augmentation techniques specifically for underrepresented classes
  + Consider using weighted loss functions to give more importance to minority classes

### Improve Background Distinction:

* + Fine-tune the model to better differentiate between tumor tissues and background
  + Enhance preprocessing techniques to improve tissue-background separation

### Space-occupying Lesion Detection:

* + Collect more training examples for this rare class
  + Consider transfer learning or few-shot learning techniques to improve detection with limited samples

### Model Architecture:

* + Experiment with different YOLO variants or other detection architectures
  + Consider ensemble methods to improve overall performance

### Post-processing:

* + Implement confidence thresholding to reduce false positives from background

The model performs well on common tumor types but requires significant improvement for rare classes. The high number of background misclassifications suggests that boundary detection between tumor and non-tumor regions needs refinement.

**Model Performance Analysis Based on F1-Confidence Curve**

Based on the provided F1-confidence curve for the YOLOv11n model trained on brain tumor detection, I can offer a comprehensive analysis of how the model performs across different tumor classifications.

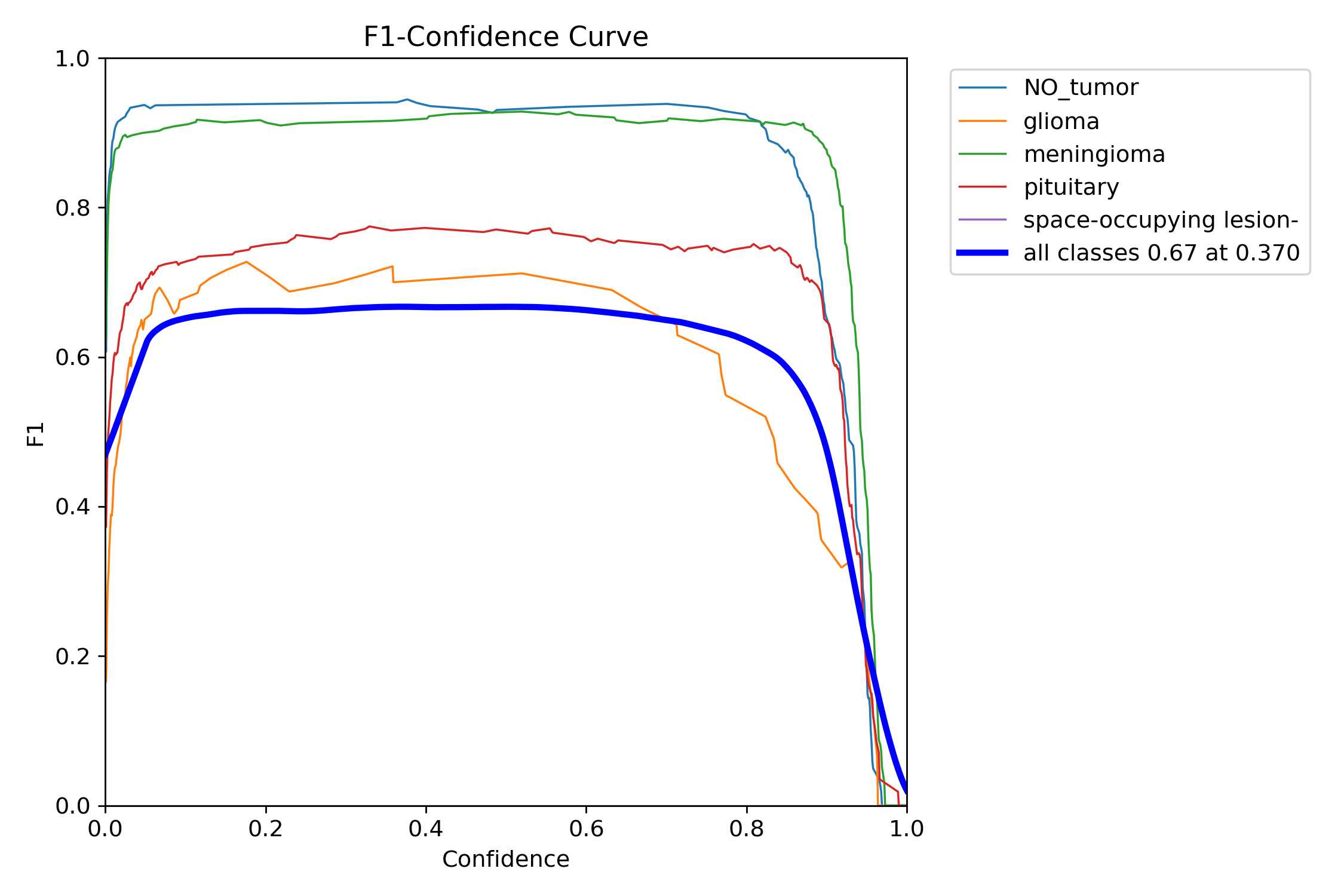


Figure . Model performance analysis based on F1-Confidence Curve

## Overall Model Performance

The model achieves an overall F1 score of 0.67 at an optimal confidence threshold of 0.370, indicating moderate performance across all classes. This threshold represents the best balance between precision and recall for the complete dataset.

## Class-Specific Performance

### NO\_tumor Class:

* + Exceptional performance with F1 scores consistently around 0.93-0.94
  + Maintains high F1 scores across a wide range of confidence thresholds (0.05-0.85)
  + Shows stable performance, indicating robust detection capabilities
  + Minimal degradation until very high confidence thresholds (>0.85)

### Meningioma Class:

* + Strong performance with F1 scores around 0.92-0.93
  + Similar stability pattern to NO\_tumor class
  + Maintains its high F1 score across various confidence thresholds
  + Represents one of the most reliably detected classes

### Pituitary Class:

* + Moderate performance with F1 scores around 0.75-0.77
  + Less stable than the top two classes, showing more sensitivity to threshold changes
  + Performance peaks in the middle confidence range (0.3-0.7)
  + Shows a gradual decline as confidence thresholds approach extremes

### Glioma Class:

* + Lower performance with F1 scores around 0.67-0.72
  + More variable performance across the confidence spectrum
  + Shows best results at moderate confidence levels (0.2-0.6)
  + Performance degrades more quickly at higher confidence thresholds compared to other classes

### Space-occupying lesion Class:

* + Performance curve is not distinctly visible in the chart
  + Likely indicates extremely poor performance or insufficient representation
  + May suggest the model struggles significantly with this class

## Key Observations

### Class Performance Hierarchy:

* + Strong detection: NO\_tumor and meningioma (F1 > 0.90)
  + Moderate detection: pituitary (F1 ≈ 0.75)
  + Weaker detection: glioma (F1 ≈ 0.70)
  + Poor detection: space-occupying lesion (insufficient data to assess)

### Confidence Threshold Sensitivity:

* + All classes maintain relatively stable F1 scores in the 0.1-0.8 confidence range
  + All classes show sharp performance drops when confidence exceeds 0.85
  + The overall model curve (bold blue) suggests an optimal confidence threshold of 0.370

### Class Imbalance Effects:

* + The significant variation in F1 scores between classes indicates potential class imbalance
  + Better-performing classes likely have more training samples
  + Space-occupying lesion class appears severely underrepresented

## Recommendations

### Class-Specific Confidence Thresholds:

* + Consider using different confidence thresholds for each class:
    - NO\_tumor and meningioma: 0.5-0.7 (higher precision without sacrificing recall)
    - Pituitary and glioma: 0.3-0.4 (balance precision and recall)

### Address Class Imbalance:

* + Employ data augmentation techniques for underrepresented classes
  + Implement weighted loss functions to prioritize learning from minority classes
  + Consider advanced sampling techniques during training

### Model Refinement:

* + Fine-tune the model with focus on improving glioma and space-occupying lesion detection
  + Experiment with different model architectures or ensemble approaches
  + Consider specialized data preprocessing for difficult classes

### Validation Strategy:

* + Implement stratified cross-validation to better assess performance across all classes
  + Evaluate model with additional metrics beyond F1 score (e.g., AUC-ROC, specificity)

The model shows promising results for common tumor types but requires significant improvement for less common classes. Addressing class imbalance and optimizing class-specific confidence thresholds could substantially improve overall performance.

**Model Performance Analysis Based on PR-Curve**

Based on the provided precision-recall (PR) curve, I can offer a comprehensive analysis of the YOLOv11n model's performance across the five brain tumor classes.

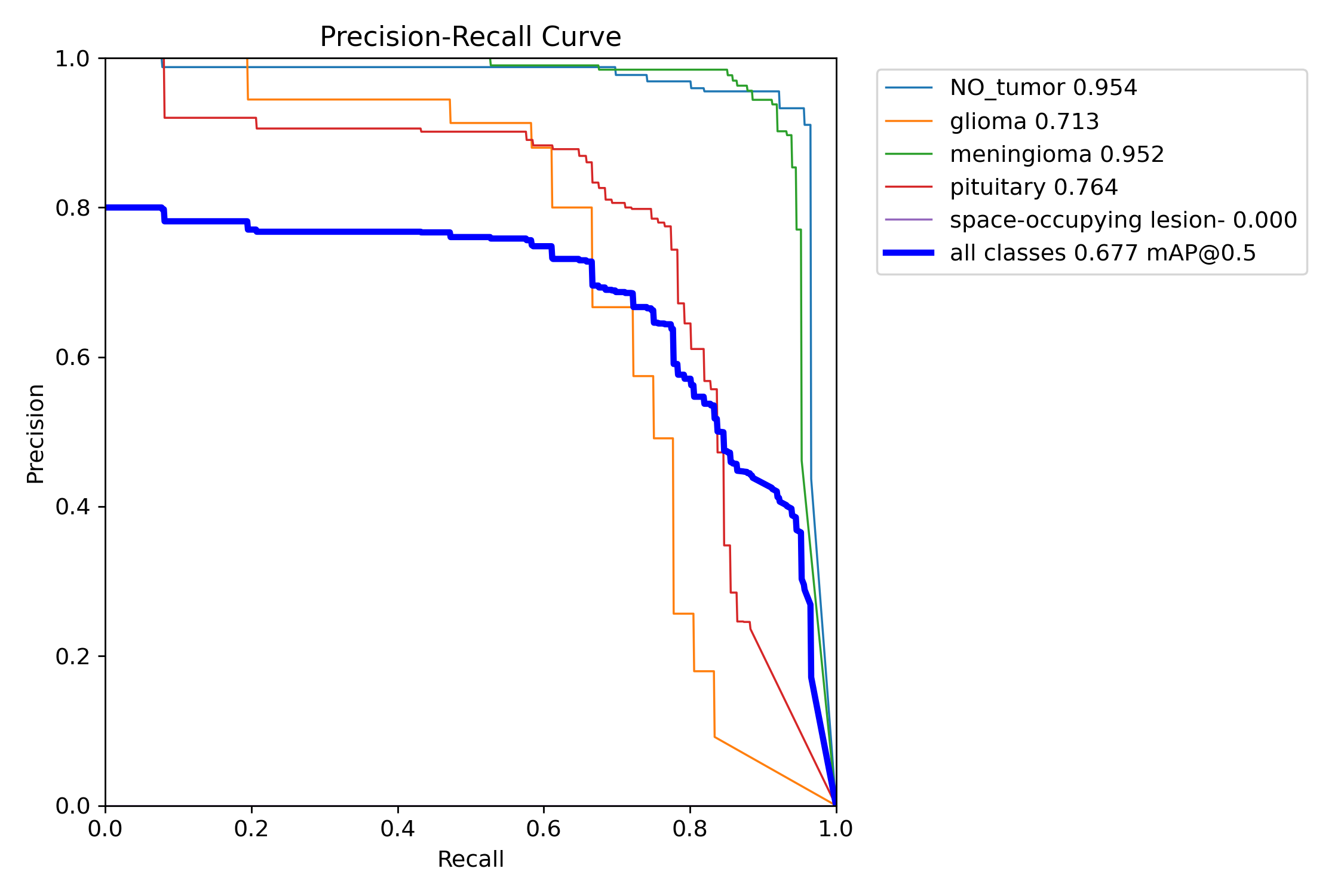


Figure . Model performance analysis based on Precision recall curve

## Overall Model Performance

The overall model performance measured by mean Average Precision (mAP@0.5) is 0.677, which indicates moderate detection capability across all classes. **The all-classes PR curve (bold blue line) shows a generally declining trend as recall increases, with precision starting around 0.80 at low recall and gradually decreasing.**

## Class-Specific Performance Analysis

### NO\_tumor (0.954 AP):

* + Exceptional performance with near-perfect precision (~1.0) maintained across a wide recall range (0-0.9)
  + The curve only shows significant precision drop when recall exceeds 0.9
  + This indicates the model rarely mistakes other classes or background for NO\_tumor
  + High AP value confirms reliable detection across virtually all confidence thresholds

### Meningioma (0.952 AP):

* + Nearly identical performance to NO\_tumor class
  + Maintains precision close to 1.0 for recall values up to approximately 0.9
  + Sharp drop in precision only at very high recall values
  + Suggests robust and reliable detection capabilities for this class

### Pituitary (0.764 AP):

* + Good but noticeably lower performance than the top two classes
  + Starts with high precision (~0.9) at low recall
  + Shows steady decline in precision as recall increases beyond 0.6
  + More pronounced performance degradation at high recall values
  + The curve shape indicates more false positives at lower confidence thresholds

### Glioma (0.713 AP):

* + Moderate performance with a distinctive step-like pattern in the PR curve
  + Maintains good precision (~0.95) initially but drops to ~0.9 at very low recall
  + Shows significant decline in precision when recall exceeds 0.7
  + Steep drops suggest discrete confidence thresholds where performance changes dramatically
  + Indicates challenges in maintaining precision while achieving high recall

### Space-occupying lesion (0.000 AP):

* + Complete failure in detection with 0.000 AP
  + No visible curve in the PR plot
  + Suggests the model cannot detect this class at all
  + Likely due to extreme class imbalance or insufficient training examples

### Comparative Analysis

The classes form three distinct performance tiers:

* **High-performance tier** (AP > 0.95): NO\_tumor and meningioma
* **Medium-performance tier** (AP 0.7-0.8): Pituitary and glioma
* **Failed detection tier** (AP = 0): Space-occupying lesion

This stratification suggests the model's effectiveness is highly dependent on class representation in the training data.

## Key Observations

* **Class Imbalance Impact**: The dramatic performance gap between classes strongly suggests significant class imbalance in the training dataset, with space-occupying lesion likely being severely underrepresented.
* **Precision-Recall Tradeoff**: For all classes except space-occupying lesion, the model can maintain high precision (>0.9) up to certain recall thresholds, after which performance degrades rapidly.
* **Detection Confidence**: The step-like pattern in the glioma curve indicates that confidence scores are not smoothly distributed, suggesting potential issues with the model's uncertainty estimation for this class.
* **Overall Detection Capability**: Despite variations, the model shows strong detection ability for 4 out of 5 classes, with only the space-occupying lesion class showing complete failure.

## Recommendations for Improvement

### Address Class Imbalance:

* + Implement data augmentation specifically for underrepresented classes
  + Consider synthetic data generation for the space-occupying lesion class
  + Apply class weighting in the loss function

### Model Architecture Adjustments:

* + Experiment with feature pyramid networks to improve detection of smaller tumors
  + Consider using transfer learning from larger YOLO variants

#### Training Strategy:

* + Implement curriculum learning by gradually introducing difficult examples
  + Use focal loss to address class imbalance
  + Employ hard negative mining to improve discrimination between similar classes

#### Data Quality:

* + Review and potentially clean the training data for the space-occupying lesion class
  + Ensure proper annotation consistency across all classes

The model demonstrates excellent performance for common tumor types but requires significant improvement for the space-occupying lesion class. With targeted improvements addressing the class imbalance and refinement of detection thresholds, overall performance could be substantially enhanced.

**Model Performance Overview**

The performance metrics and loss curves for the YOLOv11 model trained on a brain tumor detection dataset show varying trends, providing insights into the training and validation processes.

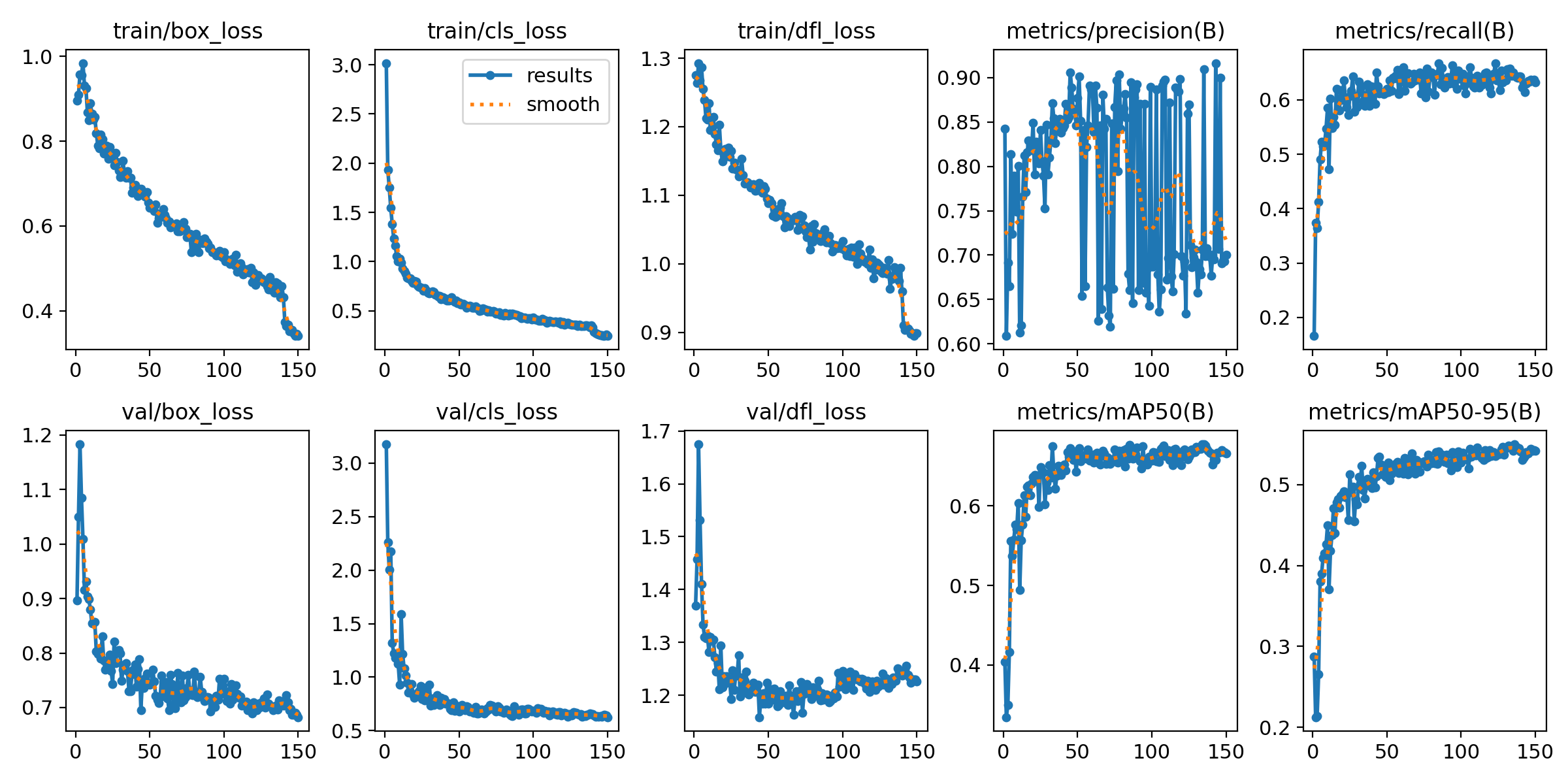


Figure . Model Performance Overview: showing box loss, class loss, dfl loss, mAP50, and mAP50-95 for training and validation

## **Loss Curves Analysis**

### Training Losses

* + **Box Loss:** Steadily decreases from ~1.0 to below 0.4, indicating improved bounding box localization.
  + **Classification Loss:** Drops sharply in early epochs and then decreases gradually, showing the model is learning to classify tumors effectively.
  + **DFL (Distribution Focal Loss):** Consistent downward trend, showing better box regression.

### Validation Losses

* + **Box, Classification, DFL Losses:** All decrease and stabilize, with no signs of divergence or overfitting. Validation losses are slightly higher than training losses, which is expected.

Interpretation:  
The model is learning well, with both training and validation losses decreasing and stabilizing, indicating good convergence and no overfitting.Precision, Recall, and mAP Metrics

### Precision (metrics/precision(B)):

* + Starts high, fluctuates in early epochs, then stabilizes around 0.8–0.9.
  + Some late-epoch noise, but the overall trend is positive.

### Recall (metrics/recall(B)):

* + Rapid improvement in the first 20 epochs, then plateaus around 0.65.
  + Indicates the model is correctly identifying most true positives, but there is room for improvement in catching all tumors.

### mAP@0.5 (metrics/mAP50(B)):

* + Increases quickly, stabilizing above 0.7.
  + Suggests good overall detection performance at a standard IoU threshold.

### mAP@0.5:0.95 (metrics/mAP50-95(B)):

* + Steadily rises and stabilizes just above 0.5.
  + Indicates moderate performance across stricter IoU thresholds.

Generalization & Overfitting Check

* + **Losses:** Validation and training losses are close, indicating minimal overfitting.
  + **Metrics:** Both mAP and recall/precision stabilize, showing consistent generalization.

Potential Issues & Recommendations

### Strengths

* + Strong localization and classification learning: Losses decrease smoothly.
  + Good precision: Model is confident in its positive predictions.
  + Solid mAP@0.5: Indicates reliable detection for most classes.

### Weaknesses

* **Recall is moderate:** Some tumors, especially rare types (like 'space-occupying lesion-'), may be missed.
* **mAP@0.5:0.95 is lower:** Performance drops with stricter localization requirements, suggesting room for improvement in bounding box accuracy.

### Next Steps

* + **Class Imbalance:** Augment data for underrepresented classes (e.g., 'space-occupying lesion-').
  + **Recall Boost:** Consider lowering detection thresholds or using recall-focused loss functions.
  + **Model Tuning:** Try larger YOLO variants or ensembling for further gains.
  + **Post-processing:** Adjust NMS (Non-Maximum Suppression) thresholds to balance precision and recall.

## Summary Table

The table 1 shows the overall performance stats of trained model in one table, indicating localization improved from 1 to 0.4, classification improves from 3 to 0.5, and exhibit good generalization (around 0.7). Model Precision is in range 0.8 to 0.9 with moderate recall around 0.65 (65%).

|  |  |  |
| --- | --- | --- |
| **Metric** | **Value (Approx.)** | **Interpretation** |
| Training Box Loss | ↓ 1.0 → 0.4 | Improved localization |
| Training Cls Loss | ↓ 3.0 → 0.5 | Improved classification |
| Validation Box Loss | ↓ 1.2 → 0.7 | Good generalization |
| Precision | 0.8–0.9 | High positive prediction confidence |
| Recall | ~0.65 | Moderate, some missed detections |
| mAP@0.5 | >0.7 | Good detection at standard threshold |
| mAP@0.5:0.95 | >0.5 | Moderate across stricter thresholds |

Table shows the overall performance stats of trained model in one table

## Conclusion

Your YOLOv11n model for brain tumor detection is well-trained, with strong precision and good mAP. The main area for improvement is recall, especially for rare tumor types. Focus future efforts on data balancing, recall optimization, and possibly model scaling for even better results.

# Analysis of Labels Correlogram for Brain Tumor Detection Dataset

The attached image is a **labels correlogram** (pair plot) for your brain tumor detection dataset, visualizing the distribution and relationships of bounding box parameters:

* **x** (center x-coordinate, normalized)
* **y** (center y-coordinate, normalized)
* **width** (normalized)
* **height** (normalized)

Below is a detailed analysis and insights based on this plot:

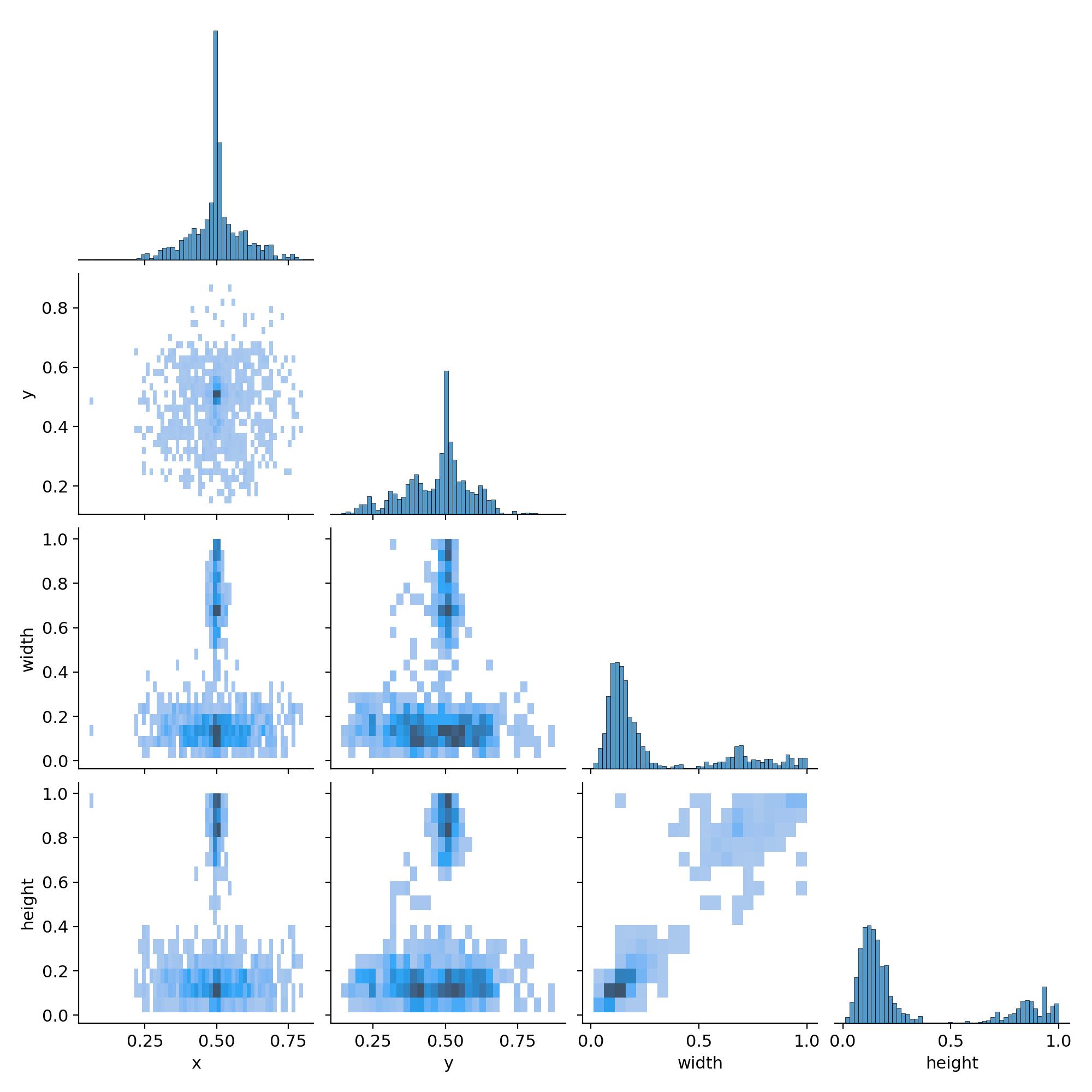


Table . Analysis of Labels Correlogram for Brain Tumor Detection Dataset

## Distribution of Tumor Locations (x, y)

* **x and y Histograms:**  
  **Both x and y coordinates are centered around 0.5**, indicating that most tumors are located near the center of the images.
* **x vs y Scatter:**  
  The scatter plot is densest in the middle, **confirming a strong central tendency**. This is typical for medical imaging datasets where the region of interest (e.g., the brain) is centered.

**Insight:**  
Your dataset is well-centered, which is beneficial for model training, as the model can focus on learning tumor features rather than searching the entire image.

## Distribution of Tumor Sizes (width, height)

* **Width and Height Histograms:**  
  Most bounding boxes have a width and height between 0.1 and 0.3 (normalized units), with a sharp peak and a long tail toward larger sizes.
* **Width vs Height Scatter:**  
  There is a cluster of small-to-medium boxes, but a few outliers with large bounding boxes (width or height close to 1.0). This suggests some tumors are very large, possibly spanning almost the entire image.

**Insight:**  
Most tumors are small to medium-sized, but there are some very large tumors. The presence of outliers may affect model performance, especially for rare, large tumors.

Correlation Between Parameters

* **x vs width / y vs height:**  
  There is a concentration of points in the center for x and y, but width and height are not strongly correlated with position.
* **Width vs Height:**  
  There is a positive correlation: larger widths tend to come with larger heights, which is expected (tumors that are large in one dimension are often large in another).

**Insight:**  
Tumor size is not strongly dependent on location, but larger tumors are generally proportional in both dimensions.

## Potential Issues and Recommendations

* **Outliers:**  
  **A few bounding boxes are extremely large (width or height near 1.0).** **These could be valid (e.g., very large tumors) or no tumor (background is predicted as tumor) or annotation errors**. Solution: Review these samples to ensure label quality.
* **Class Representation:**  
  This correlogram does not show per-class distributions. If certain tumor types (e.g., 'space-occupying lesion-') are rare or have different size/location patterns, consider stratified augmentation or analysis.

## Summary Table

The table 3 shows the distribution of tumor , distribution of tumor spread, in the dataset along with outliers and correlation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Distribution Center** | **Spread** | **Outliers Present?** | **Correlation** |
| x (center) | 0.5 | Moderate | No | None |
| y (center) | 0.5 | Moderate | No | None |
| width | 0.1–0.3 | Skewed right | Yes (up to 1.0) | Positive with height |
| height | 0.1–0.3 | Skewed right | Yes (up to 1.0) | Positive with width |

Table shows Analysis of Labels Correlogram for Brain Tumor Detection Dataset

## Actionable Insights

* **Dataset is well-centered:** This is good for model learning.
* **Most tumors are small/medium:** Model should be able to generalize well, but ensure anchor boxes are appropriate for this distribution.
* **Review outliers:** Check very large bounding boxes for annotation errors.
* **Consider per-class analysis:** If performance is poor for a class, check if its label distribution is different.

Summary:  
The dataset's bounding box labels are mostly well-distributed and centered, with a healthy variety of tumor sizes. Watch out for outliers and consider further analysis if you see class-specific performance issues. This strong label distribution provides a solid foundation for object detection models like YOLO.

# Task 2: Chapter 1 and 2 of Hands-On Machine Learning with Scikit-Learn, Keras and Tensor Flow

## Exercise: 1

### How would you define Machine Learning?

Machine Learning is essentially the **science (and art) of programming computers so they can learn from data**. Instead of giving a computer explicit instructions for every possible situation, you program it so that it can learn from examples, or "experience," and improve its performance on a specific task over time. Arthur Samuel, a pioneer in the field, defined it as giving computers the **ability to learn without being explicitly programmed**.

Tom Mitchell gave a more technical definition: a computer program learns from experience E with respect to some task T and performance measure P, if its performance on T, as measured by P, improves with experience E. A classic example is a spam filter, which learns to identify spam emails by being shown many examples of spam and regular emails.

### Can you name four types of problems where it shines?

Machine Learning is particularly useful for problems where:

* Existing solutions need a **lot of manual tweaking or extensive lists of rules**. A single ML algorithm can often simplify code and work better.
* The problems are **too complex to solve with traditional methods**, or there's no known way to write a standard program for them. ML techniques can find solutions in these cases.
* The environment is **constantly changing** (fluctuating). An ML system can automatically adjust to new data.
* You want to **gain insights about complex problems and large amounts of data**. ML techniques can help discover patterns that aren't obvious.

### What is a labeled training set?

In **supervised learning**, a labeled training set is the **data you provide to a machine learning algorithm that includes the desired solutions, called labels**.

For instance, if you're training a system to recognize handwritten digits, a labeled training set would include thousands of images of digits, each correctly marked with the number it represents (e.g., an image of a "4" labeled with "4"). For a spam filter, the training data includes emails marked as either "spam" or "ham" (not spam).

### What are the two most common supervised tasks?

The two most common tasks in supervised learning are:

* **Classification**: This involves training the system to **predict a category or class** for a given input. A spam filter classifying emails as spam or not spam is a classification task. Classifying handwritten digits into categories 0 through 9 is another example.
* **Regression**: This involves training the system to **predict a target numeric value**. Predicting the price of a car based on its features (like mileage, age, and brand) is a regression task. Predicting housing prices based on census data is another example.

### Can you name four common unsupervised tasks?

Unsupervised learning involves training data that is **unlabeled**, meaning the system learns without being given the correct answers. Four common unsupervised tasks are:

* **Clustering**: This groups similar instances together into clusters. For example, grouping visitors to a blog based on their behavior or interests without being told beforehand what groups exist. It can be used for customer segmentation.
* **Anomaly detection and novelty detection**: These tasks identify instances that look abnormal or are outliers compared to the training data. For example, detecting faulty items on a production line or unusual network traffic patterns.
* **Visualization and dimensionality reduction**: This reduces complex data with many features into a simpler form that's easier to visualize or handle. For instance, reducing a 1,000-dimensional dataset to 2D or 3D while preserving important information.
* **Association rule learning**: This discovers relationships between items in large datasets, such as identifying products that are often purchased together.

### What type of Machine Learning algorithm would you use to allow a robot to walk in various unknown terrains?

The sources suggest that **Reinforcement Learning** algorithms are implemented by many robots to learn how to walk. In Reinforcement Learning, a system (like a robot) learns by performing actions and receiving rewards or penalties, allowing it to figure out the best strategy (or "policy") to achieve a goal, such as navigating unknown terrains.

### What type of algorithm would you use to segment your customers into multiple groups?

You would use a **clustering algorithm**. Clustering is explicitly mentioned as a great tool for **customer segmentation**, where you group customers based on their characteristics or behavior (like purchases or website activity) to understand them better and tailor products or marketing.

### Would you frame the problem of spam detection as a supervised learning problem or an unsupervised learning problem?

The sources frame the problem of spam detection as a **supervised learning problem**. This is because the system is trained using a dataset where emails are already **labeled** as either "spam" or "ham". The algorithm learns from these labeled examples to classify new, unseen emails.

### What is an online learning system?

An online learning system is a Machine Learning system that is **capable of learning incrementally** from a stream of incoming data, *on the fly*. It processes data instances one by one or in small groups ("mini-batches").

This is useful for systems that need to rapidly adapt to changes or handle continuous data streams.

### What is out-of-core learning?

Out-of-core learning is a technique used with online learning algorithms to **train systems on huge datasets that are too large to fit into a single computer's main memory (RAM)**. The algorithm loads parts ("batches") of the data, performs a training step, and repeats this process until it has gone through all the data.

Although it uses online learning algorithms, it's usually done offline, not on a live system; it's essentially **incremental learning on a large dataset**.

### What type of learning algorithm relies on a similarity measure to make predictions?

**Instance-based learning** algorithms rely on a similarity measure to make predictions. These systems learn by simply remembering the training examples.

When they need to make a prediction for a new instance, they compare it to the examples they've seen using a similarity measure and predict based on the most similar examples. For example, classifying a new shape based on the class of the majority of its nearest similar shapes in the training data.

### What is the difference between a model parameter and a learning algorithm’s hyperparameter?

A **model parameter** is a value within the model itself that is **learned** by the training algorithm from the data. These parameters determine how the model makes predictions.

For example, in a simple linear model like life\_satisfaction = θ0 + θ1 × GDP\_per\_capita, θ0 and θ1 are the model parameters that the algorithm finds to make the line fit the data best.

A **learning algorithm’s hyperparameter** is a parameter of the learning algorithm itself (not the model). It is **not learned** from the data during training. Instead, it **must be set *before* training begins and remains constant** during training.

Hyperparameters control aspects of the learning process or the model's complexity, such as the amount of regularization to apply (controlling how much the model is constrained), the learning rate in Gradient Descent, or the depth of a Decision Tree. Tuning hyperparameters is a key part of building a Machine Learning system.

### What do model-based learning algorithms search for? What is the most common strategy they use to succeed? How do they make predictions?

Model-based learning algorithms **search for an optimal set of model parameters** that allows the model to generalize from the training data to new instances.

The most common strategy they use is to **minimize a cost function** (or maximize a utility function) that measures how well the model fits the training data. For example, in linear regression, they search for parameters that minimize the distance between the model's predictions and the actual training data values.

Once the model has been trained (i.e., the optimal parameters have been found), they **make predictions on new instances by applying the learned model**. For example, using the equation life\_satisfaction = θ0 + θ1 × GDP\_per\_capita with the learned values for θ0 and θ1 to predict life satisfaction for a country with a new GDP per capita.

### Can you name four of the main challenges in Machine Learning?

Some of the main challenges in Machine Learning arise from "bad data" or "bad algorithms". Four of these challenges include:

* **Insufficient Quantity of Training Data**: Most ML algorithms need a significant amount of data to work correctly, sometimes thousands or millions of examples.
* **Nonrepresentative Training Data**: If the training data doesn't accurately reflect the situations the model will encounter when making predictions, the model won't generalize well.
* **Poor-Quality Data**: Data with errors, outliers, or noise can make it harder for the algorithm to find underlying patterns or may lead it to find patterns in the noise itself.
* **Overfitting the Training Data**: This happens when a model learns the training data too well, including the noise or irrelevant details, causing it to perform poorly on new, unseen instances.

### If your model performs great on the training data but generalizes poorly to new instances, what is happening? Can you name three possible solutions?

If a model performs very well on the training data but poorly on new instances, it is **overfitting the training data**. This means the model has become too tailored to the specific training examples and has not learned the general patterns that apply to new data. Possible solutions mentioned in the sources include:

* **Simplify the model**: Use a less complex model (like a linear model instead of a high-degree polynomial) or constrain the model (using regularization).
* **Gather more training data**: Providing more examples can help the model learn the general patterns rather than just the specifics of the limited training set.
* **Reduce the noise in the training data**: Clean up errors and remove outliers from the training set.

### What is a test set and why would you want to use it?

A **test set** is a portion of the dataset that is set aside and **not used during training**. You use a test set to get an **estimate of the generalization error** the model will make on new instances before deploying it.

It's a way to evaluate how well your trained model is expected to perform in the real world on data it hasn't seen before. It's crucial to use a test set that is representative of the data the model will encounter in practice.

### What is the purpose of a validation set?

The purpose of a **validation set** is to **tune hyperparameters and compare different models**. You train multiple models with different hyperparameters (settings for the learning process) or even different types of models (e.g., Linear Regression vs. Decision Tree) and evaluate their performance on the validation set. This helps you select the best-performing model and hyperparameters without "contaminating" the test set.

### What can go wrong if you tune hyperparameters using the test set?

If you tune hyperparameters using the test set, you risk **overfitting the test set**. This means you might select a model and hyperparameters that perform exceptionally well specifically on the test data, but this performance might not reflect how well the model will generalize to truly new, unseen data.

The performance measure on the test set would become an **overoptimistic estimate** of the model's true performance.

### What is repeated cross-validation and why would you prefer it to using a single validation set?

The provided sources discuss using cross-validation to evaluate models, often mentioning splitting the training data into multiple "folds" and training/evaluating the model on different combinations of these folds.

This is a more robust way to evaluate a model than using a single fixed validation set, as it provides performance measures across different subsets of the data.

However, the sources provided do not contain information specifically defining "repeated cross-validation" or explaining why it might be preferred over simple k-fold cross-validation or using a single validation set. Therefore, I cannot answer this question based solely on the given material.

## Exercise 2:

## Task 1: Implement the code presented in the book and analyze the outputs:

### Get the data

This code provides functions to fetch and load housing data from a remote source. The **fetch\_housing\_data** function downloads a compressed dataset from the given URL, extracts it into the specified path, creating the directory if necessary. The **load\_housing\_data** function reads the extracted CSV file into a Pandas DataFrame, returning the loaded data for further processing (view code in Figure 9).

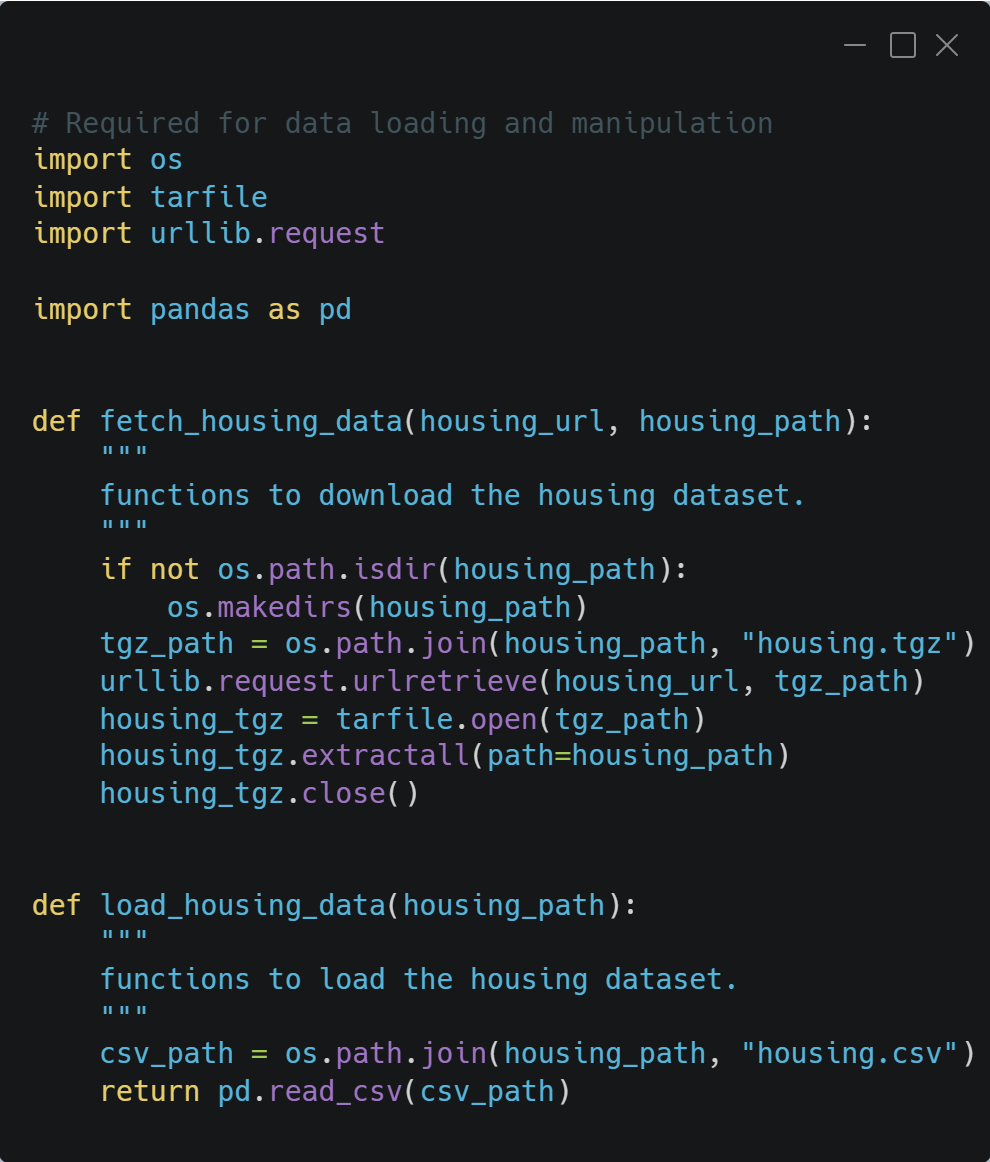


Figure shows the code to download the dataset from remote source and then load it in pandas dataframe

### Strated Dataset

I split the dataset into two sets: strat\_train\_set for training and strat\_test\_set for testing. Figure 10 shows the code used to split the dataset in train and test sets.

This code creates a stratified split of the housing dataset based on income category. It first discretizes the median\_income into five income categories. Then, StratifiedShuffleSplit ensures that the training and testing sets maintain the same income category distribution as the original dataset.

Finally, it removes the temporary income category column and separates features (housing data) from the target variable (median house value) in the training set, returning the processed training features, labels, and the stratified test set.



Figure shows the code used to split the dataset in train and test sets .

### Data preparation

The file named data\_preparation.py prepares housing data for machine learning. Figure 11 shows the full code snippet (double click or double tab on image to view the full code in OpenDocument)

It defines a **CombinedAttributesAdder** to create new features like rooms\_per\_household, population\_per\_household, and optionally bedrooms\_per\_room.

The **prepare\_data** function takes the raw housing dataframe as input.

It first dynamically determines column indices for specific features needed by the CombinedAttributesAdder.

Inside CombinedAttributesAdder:

* It inherits from **BaseEstimator** and **TransformerMixin** for scikit-learn compatibility.
* The transform method calculates the new combined features using numpy operations.
* It returns a **numpy array** with the original features plus the new ones.

The function prepare\_data then **separates numerical and categorical attributes.**

It constructs a **numerical pipeline (num\_pipeline)** consisting of:

* **SimpleImputer:** Fills missing values in numerical features with the median.
* **CombinedAttributesAdder**: Adds the combined features.
* **StandardScaler**: Standardizes numerical features (mean 0, std 1).

It also defines a categorical pipeline implicitly within ColumnTransformer.  
This pipeline applies OneHotEncoder with handle\_unknown='ignore' to the 'ocean\_proximity' feature.

**ColumnTransformer** is used to combine the pipelines:

* It applies num\_pipeline to numerical attributes.
* It applies OneHotEncoder to the 'ocean\_proximity' attribute.

Finally, **prepare\_data fits and transforms the input housing dataframe using the full pipeline.**The transformed data (housing\_prepared) and the pipeline itself (full\_pipeline) are returned.  
**housing\_prepared is ready for model training; full\_pipeline can be used to transform new data.**



Figure . Function to prepare data (double click or double tab to view the full code in OpenDocument)

### Model Used

Figure 12 shows the code to trains a Random Forest Regressor model (A powerful ensemble learning method that fits multiple decision trees and averages their predictions to reduce overfitting) on preprocessed housing data (housing\_prepared) to predict median house values (housing\_labels).

It initializes the model with 100 trees and a fixed random state for reproducibility. The trained model is then saved to a file named "random\_forest\_model.pkl" using the joblib library for later use. Finally, the function returns the trained model object.

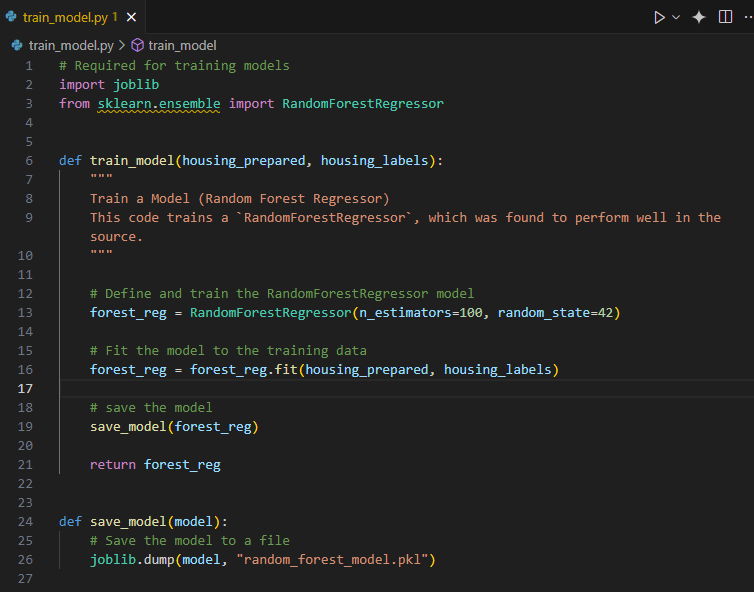


Figure shows code to train and save the model

### Random Forest Cross-Validation Scores for trained model:

Ran **10-fold cross-validation**, which splits training data into 10 parts, trains on 9, and tests on the 10th, rotating through all (view code in Fifure 13).



Figure show two functions. "evaluate\_model" will test the trained model on test set, and "display\_scores" function will return mean, standard deviation, and root mean square errors

**Model RMSE:**

[51559.63379638 48737.57100062 47210.51269766 51875.21247297 47577.50470123 51863.27467888 52746.34645573 50065.1762751 48664.66818196 54055.90894609]

**Mean:** 50435.58092066179

**Standard deviation:** 2203.3381412764606

**Interpretation:**

* **Mean RMSE**: 50,435

On average, your model's predictions are off by ~$50,435 in median house value.

* **Standard Deviation: 2,203**

Low variability — the model performs consistently across different data folds. **This consistency indicates good generalization on unseen data.**

### Feature Importance

The table 4 shows which columns of dataset are more important for model to predict the house value (see Figure 14 for code).

|  |  |
| --- | --- |
| **Feature** | **importance** |
| median\_income | 0.3452 |
| INLAND | 0.1587 |
| pop\_per\_hhold | 0.1107 |
| Longitude | 0.0714 |
| bedrooms\_per\_room | 0.0697 |
| Latitude | 0.0629 |
| rooms\_per\_hhold | 0.0595 |
| housing\_median\_age | 0.043 |
| Population | 0.0165 |
| total\_rooms | 0.0164 |
| total\_bedrooms | 0.0156 |
| Households | 0.0154 |
| <1H OCEAN | 0.0087 |
| NEAR OCEAN | 0.0036 |
| NEAR BAY | 0.0027 |
| ISLAND | 0.0001 |
|  |  |

Table shows which columns of dataset are more important for model to predict the house value

### Best hyperparameters found by Randomized Search:

I use RandomizedSearchCV for Hyperparameter Tuning. Figure 14 shows the full code snippet (Double click or double tab the image to view the full code in OpenDocument text).

The function  “analyze\_best\_model\_error” will print the best hyper parameters mention below. The function fine\_tune\_model() function in Figure 14 returns “rnd\_search” to main.py

**Best hyper\_parameters**: {'max\_features': 7, 'n\_estimators': 180}

**max\_features=7**: Allows the model to consider 7 features when splitting a node, balancing bias and variance.

**n\_estimators=180**: Uses 180 trees to make more robust predictions.



Figure show function "fine\_tune\_model" to tune the model according to best hyper- parameters, and analyze\_best\_model\_error function print the best features. (Double click or double tab the image to view the full code in OpenDocument text)

### Evaluation Results of Fine Tuned Model

Figure 15 shows the code used to evaluate the fine-tuned model to obtain the below mention results with 95% confidence level.

**Final RMSE on the test set:** 46981.841079394515

**95% Confidence Interval for Test RMSE (t-dist): [45009.73121871 48874.43992557]**

**Final RMSE: 46,981**

* The model performs slightly better on the final test set than on average cross-validation folds — a good sign, suggesting no overfitting.

**95% Confidence Interval: [45,009, 48,874]**

* If you repeat the testing many times, 95% of the time the RMSE would fall in this range.
* Interpretation: Your model will usually be off by ~$45k–$49k in predictions.



Figure shows the function will evaluate the fine-tuned model to determine root mean square error with 95% confidence interval

## Task 2: Try adding a transformer in the preparation pipeline to select only the most important attributes.

### Get the data

This code provides functions to fetch and load housing data from a remote source. The **fetch\_housing\_data** function downloads a compressed dataset from the given URL, extracts it into the specified path, creating the directory if necessary. The **load\_housing\_data** function reads the extracted CSV file into a Pandas DataFrame, returning the loaded data for further processing (view code in Figure 16).

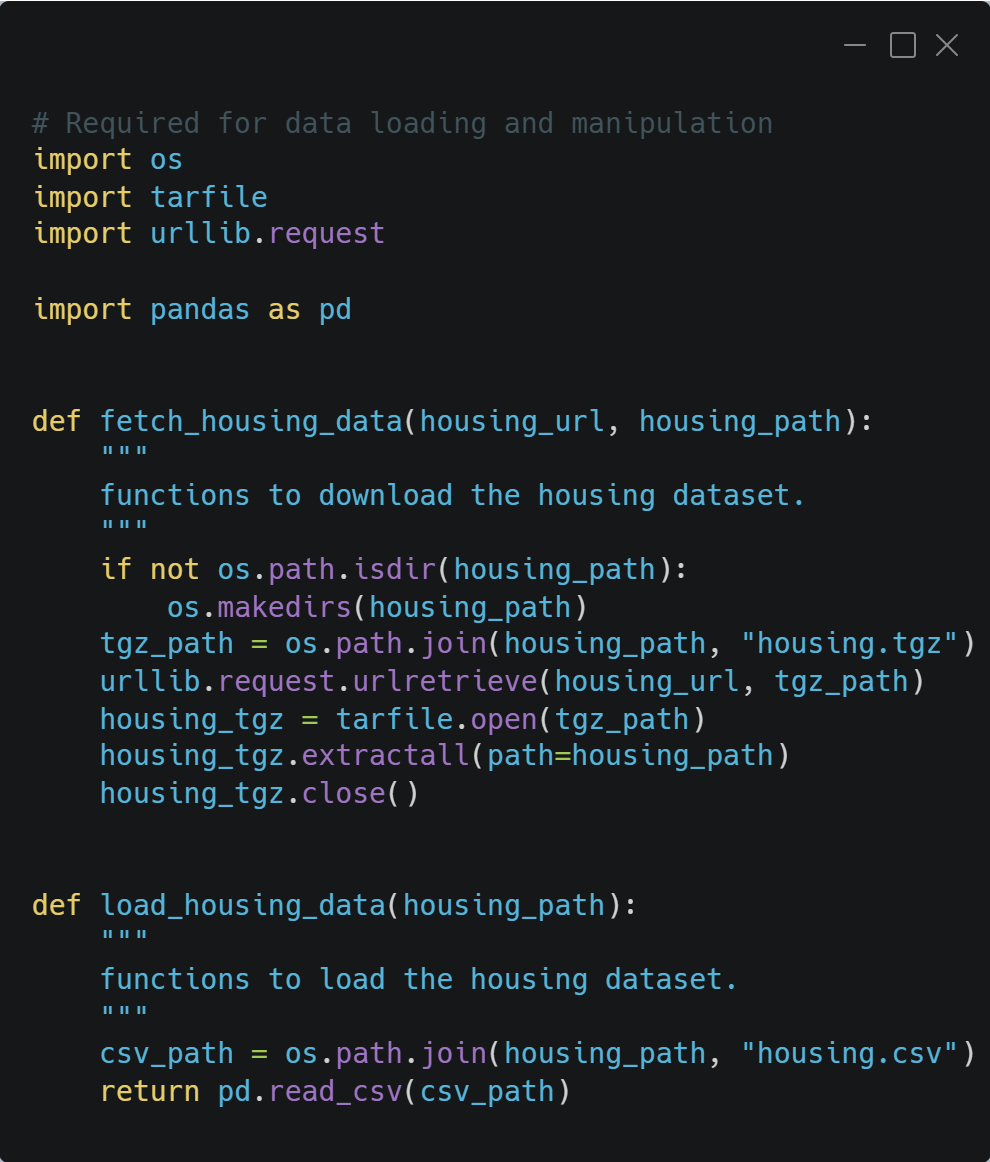


Figure split the dataset into two sets: strat\_train\_set for training and strat\_test\_set

### Strated Dataset

I split the dataset into two sets: strat\_train\_set for training and strat\_test\_set for testing. Figure 17 shows the code used to split the dataset in train and test sets.

This code creates a stratified split of the housing dataset based on income category. It first discretizes the median\_income into five income categories. Then, StratifiedShuffleSplit ensures that the training and testing sets maintain the same income category distribution as the original dataset.

Finally, it removes the temporary income category column and separates features (housing data) from the target variable (median house value) in the training set, returning the processed training features, labels, and the stratified test set.



Figure split the dataset into two sets: strat\_train\_set for training and strat\_test\_set for testing

### Data preparation

The file named data\_preparation.py prepares housing data for machine learning. Figure 18 shows the full code snippet (double click or double tab on image to view the full code in OpenDocument)

It defines a **CombinedAttributesAdder** to create new features like rooms\_per\_household, population\_per\_household, and optionally bedrooms\_per\_room.

The **prepare\_data** function takes the raw housing dataframe as input.

It first dynamically determines column indices for specific features needed by the CombinedAttributesAdder.

Inside CombinedAttributesAdder:

* It inherits from **BaseEstimator** and **TransformerMixin** for scikit-learn compatibility.
* The transform method calculates the new combined features using numpy operations.
* It returns a **numpy array** with the original features plus the new ones.

The function prepare\_data then **separates numerical and categorical attributes.**

It constructs a **numerical pipeline (num\_pipeline)** consisting of:

* **SimpleImputer:** Fills missing values in numerical features with the median.
* **CombinedAttributesAdder**: Adds the combined features.
* **StandardScaler**: Standardizes numerical features (mean 0, std 1).

It also defines a categorical pipeline implicitly within ColumnTransformer.  
This pipeline applies OneHotEncoder with handle\_unknown='ignore' to the 'ocean\_proximity' feature.

**ColumnTransformer** is used to combine the pipelines:

* It applies num\_pipeline to numerical attributes.
* It applies OneHotEncoder to the 'ocean\_proximity' attribute.

Finally, after all the previous steps, the code selects three specific features: **"median\_income", "population\_per\_household", and a binary feature indicating whether a house is "INLAND" (derived from the one-hot encoding of "ocean\_proximity")**.

This step suggests that prior analysis or domain expertise has identified these as the most important features for predicting house prices in this particular dataset. **The SpecificFeatureSelector class is responsible for extracting these three features from the pipeline (named as** “**full\_pipeline”).**

The transformed data with three selected features is returned as numpy array in variable “strat\_train\_set\_with\_selected\_features\_prepared”.

****

Figure shows the code to add a transformer in the preparation pipeline to select only the most important features. Double click or double tap the image to view full code snippet in OpenDocumentText

### Model Used

This code trains a Random Forest Regressor model to predict median house values.

It initializes the model with 100 trees and a fixed random state for reproducibility. The trained model is then saved to a file named "random\_forest\_model.pkl" using the joblib library for later use. Finally, the function returns the trained model object.



Figure shows the code to train the Random Regression model. It initializes the model with 100 trees and random =42.

### Random Forest Cross-Validation Scores for trained model:

* **Model and Evaluation**: The model performance is primarily assessed using RMSE (Root Mean Squared Error), which measures the typical prediction error in the same units as the target (see Figure 20 for code).
* **Cross-Validation:**The model was evaluated using **10-fold cross-validation**. The individual RMSE scores for each fold are listed in table 5. **The mean RMSE across folds is 90653.66, with a standard deviation of 1551.78**. This relatively low standard deviation suggests that the **model's performance is fairly consistent across different subsets of the training data**. A mean RMSE of ~$90,654 means that, on average, the model's predictions are off by about $90,654

****

Figure shows the code that evaluates the trained model by cross validating and print root mean square error, standard deviation.

|  |  |
| --- | --- |
| **rmse\_scores**  **(Model RMSE)** | 88771.21248772, 92007.34428995, 88496.93680784,  91732.18829013, 88870.97312144, 89605.64413855,  93064.89088879, 91199.02358689, 90556.10070022,  92232.28907002 |
| **mean\_rmse**  **(Mean RMSE)** | 90653.66033815352 |
| **std\_dev\_rmse**  **(Standard deviation)** | 1551.7764383124945 |

Table shows all the metrics to evalutae the trained model

### Feature Importances:

Table 6 shows the list of best features or important features of model (see Figure 21 for code)

**median\_income:** 50.16% importance - This is the most important feature, indicating that the median income in a given area is a strong predictor of housing prices.

**population\_per\_household:** 44.11% importance - The number of people per household also plays a significant role

**INLAND:** 5.73% importance - This is categorical feature. While much less important than the other two, it still has some predictive power.

|  |  |
| --- | --- |
| **feature** | **importance** |
| median\_income | 0.3452 |
| INLAND | 0.1587 |
| pop\_per\_hhold | 0.1107 |
| longitude | 0.0714 |
| bedrooms\_per\_room | 0.0697 |
| latitude | 0.0629 |
| rooms\_per\_hhold | 0.0595 |
| housing\_median\_age | 0.043 |
| population | 0.0165 |
| total\_rooms | 0.0164 |
| total\_bedrooms | 0.0156 |
| households | 0.0154 |
| <1H OCEAN | 0.0087 |
| NEAR OCEAN | 0.0036 |
| NEAR BAY | 0.0027 |
| ISLAND | 0.0001 |

Table shows the list of best features or important features of model

### Best hyperparameters found by Randomized Search:

I use RandomizedSearchCV for Hyperparameter Tuning. Figure 21 shows the full code snippet.

 print(rnd\_search.best\_params\_) in main.py will print the best hyper parameters mention below. Fine\_tune\_model() function in Figure 21 returns “rnd\_search” to main.py

**Best hyper-parameters:** {'max\_features': 3, 'n\_estimators': 150}

**max\_features=7**: Allows the model to consider 7 features when splitting a node, balancing bias and variance.

**estimators=180**: Uses 180 trees to make more robust predictions.



Figure shows the full code for fine tunning the model using RandomSearchCV .

### Evaluation of Fine Tuned Model

**Test Set Performance:**After training on the entire training set with the best hyperparameters, the model was evaluated on a separate test set. The final RMSE on the test set was **145363.26**. This is significantly higher than the cross-validation RMSE, suggesting that the test set distribution is different from the training data.

**Confidence Interval:**A 95% confidence interval for the test RMSE was calculated using the t-distribution: **[141567.88, 149062.03]**. This means we are 95% confident that the true RMSE of the model on unseen data would fall within this range. The relatively narrow range suggests a reasonably precise estimate of the model's test performance, despite the relatively high RMSE



Figure shows the code for evaluation of fine tuned model for selected features only