Brain Tumor MRI Image Classification with Deep Learning

By: Angeles Olvera

TABLE OF CONTENTS

Problem Identification **3**
- Problem Statement 3
- Context 3
- Criteria for success 3
- Scope of Solution Space 4
- Constraints within solution space 4
- Stakeholders to provide insight 4
- Key Data Source 4

Data Wrangling & Pre-Processing **5**
- Loading Image Data 5
- For Loops 5
- Applying CLAHE and Normalization Techniques to Grayscale MRI Images" 6

Exploratory Data Analysis (EDA) **6**
- Image Comparison 6
- Class Balances 8
- Pixel Intensity Heatmap 9
- Image Transformations 11

Model Selection & Training **11**
- ResNet50 12
- DenseNet121 13
- Xception 14

Model Predictions and Evaluations **15**
- Predictions Function 15
- Metric Evaluation Function 15
- ResNet50, DenseNet121 and Xception Evaluation scores 16
- Confusion Matrix 16

Model Optimization **18**
- Manual Optimization 18
- Confusion Matrix for Exception Model 18

Testing the Model Final Model **19**
- Optimized Xception Evaluation scores for Test Data 19
- Confusion Matrix for Optimized Xception Model on Test Data 19

Recommendations **20**

Problem Identification

Problem Statement

The client, NeuroScan Diagnostics, is a leading healthcare provider specializing in medical imaging and early disease detection. One of the biggest challenges in brain tumor diagnosis is the accurate and timely classification of tumors based on MRI scans. Misclassification or delayed detection can lead to incorrect treatment plans, increasing risks for patients.

To address this, NeuroScan Diagnostics seeks to implement an AI-powered deep learning model capable of identifying glioma tumors, meningioma tumors, pituitary tumors, or no tumor based on brain images. By integrating a robust classification system, the organization aims to support radiologists with automated tumor detection, improving diagnostic precision and reducing manual workload. A reliable deep learning model must be developed and deployed by the end of the fiscal year to ensure readiness for clinical validation and real world applications.

Context

Brain tumors are a significant challenge in modern medicine, requiring early and precise diagnosis to improve patient outcomes. Medical professionals rely on MRI scans to identify tumor types, but manual classification is time consuming and prone to human error. As a result, hospitals, radiology centers, and biotech companies are exploring AI powered solutions to assist specialists in detecting and diagnosing tumors more efficiently. NeuroScan Diagnostics, is based in San Diego, CA, and is a leader in AI powered medical imaging and diagnostics. San Diego is renowned for its cutting-edge research in neurology and biomedical sciences, making it the perfect environment for innovating advanced tumor detection models. Though NeuroScan is a new player in the industry, it has quickly established itself as a trusted provider of high-accuracy AI driven diagnostic tools, earning recognition for its contributions to radiology and healthcare AI. To maintain its reputation as an industry leader, their deep learning model should be capable of classifying glioma tumors, meningioma tumors, pituitary tumors, or no tumor with exceptional precision.

Criteria for success

The success of this deep learning model will be determined by its ability to accurately classify brain tumors based on MRI images. The model should demonstrate high classification accuracy across glioma, meningioma, pituitary tumor, and no tumor categories, ensuring reliable medical diagnoses. It must minimize false positives and false negatives to support confident clinical decision-making. Strong performance metrics, including precision, recall, F1-score, and AUC-ROC, will be used to validate its effectiveness.

Scope of Solution Space

To optimize performance, multiple deep learning image classification models will be implemented and compared. CNN architectures such as ResNet, EfficientNet, and MobileNetV2 will be tested to determine which provides the highest precision, recall, and F1-score for tumor detection. Additional techniques will be applied, including data augmentation to improve model generalization, hyperparameter tuning to optimize accuracy, and ensemble methods to combine model strengths. The final model must be clinically viable, ensuring predictions closely align with expert radiologists' assessments. Once validated, the best performing model will be integrated into NeuroScan's medical imaging workflow to support automated tumor detection in real world applications.

Constraints within solution space

Constraints within the solution space for this brain tumor classification project involve several technical and practical challenges. Medical imaging data is often complex, and variations in MRI scan quality, patient positioning, and imaging equipment may affect the reliability of the model's predictions. Ensuring that the deep learning model provides clinically viable results is essential, as inaccuracies in tumor classification could lead to misdiagnosis. The dataset may contain imbalanced class distributions, meaning some tumor types could have fewer samples than others, impacting model performance. Additionally, MRI images might have noise or artifacts that require extensive preprocessing to improve accuracy.

Stakeholders to provide insight

Lead Medical Imaging Specialist – Dr. Olivia Matthews
Dr. Olivia Matthews – will provide clinical expertise to ensure the model aligns with real world radiology practices

Key Data Source

The dataset consists of MRI brain images, structured into a training set and a test set to support model development and evaluation. The images are categorized into four classes: glioma tumor, meningioma tumor, pituitary tumor, and no tumor. The training set contains approximately 3,310 images, while the test set includes 394 images, both preprocessed to improve contrast and reduce noise. Each image is a 2D grayscale scan with a resolution of 240x240 pixels, maintaining consistency across the dataset

Brain Tumor balanced data

Data Wrangling & Pre-Processing

Loading Image Data

To begin, I downloaded the dataset from Kaggle, which was provided as a zipped folder. After extracting it, I uploaded the data into the Jupyter Notebook for further processing. The dataset was organized into two main directories, one for training data and one for testing data. Each of these directories contained four subfolders, representing the following classes: glioma tumors, meningioma tumors, pituitary tumors, and no tumor. Since the original dataset was compressed, I used the Python zipfile library to extract its contents into my working directory. Here's a snippet of the code I used:

```
import zipfile

zip_path = "path/to/dataset.zip"     # Replace with your zip file path
extract_path = "dataset_extracted"     # Set a folder placeholder name

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
```

In my case, I named the destination folder dataset_extracted. I removed my path and replaced it with a generic example because this notebook will be uploaded to Github and my path needs to be private. This block of code was later removed from my notebook for the same privacy reasons.

Next, I needed to ensure each folder of images was correctly mapped to the appropriate variables for my model. I started by creating a list of labels that represent the four categories each MRI image could fall into, 'glioma', 'meningioma', 'no_tumor' and 'pituitary'.

For Loops

I then wrote a for loop to iterate over each label in my list. For every label, the loop dynamically constructs the path to that label's corresponding folder within the training dataset, since I'm beginning with the training data first. Inside the training directory, there are four subfolders named exactly after the labels in my list ('glioma', 'meningioma', 'no_tumor' and 'pituitary'), which makes it easy to map each image to its appropriate category.

Inside that loop, I created another for loop to go through every folder in the folder path, this inner loop with tqdm, a library that adds a progress bar so I can visually track processing status. This is especially helpful because MRI files can be large, and tqdm gives a real time estimate of how many images are left and how long the process might take. Inside of this same loop is where each image is loaded and processed.

Applying CLAHE and Normalization Techniques to Grayscale MRI Images"

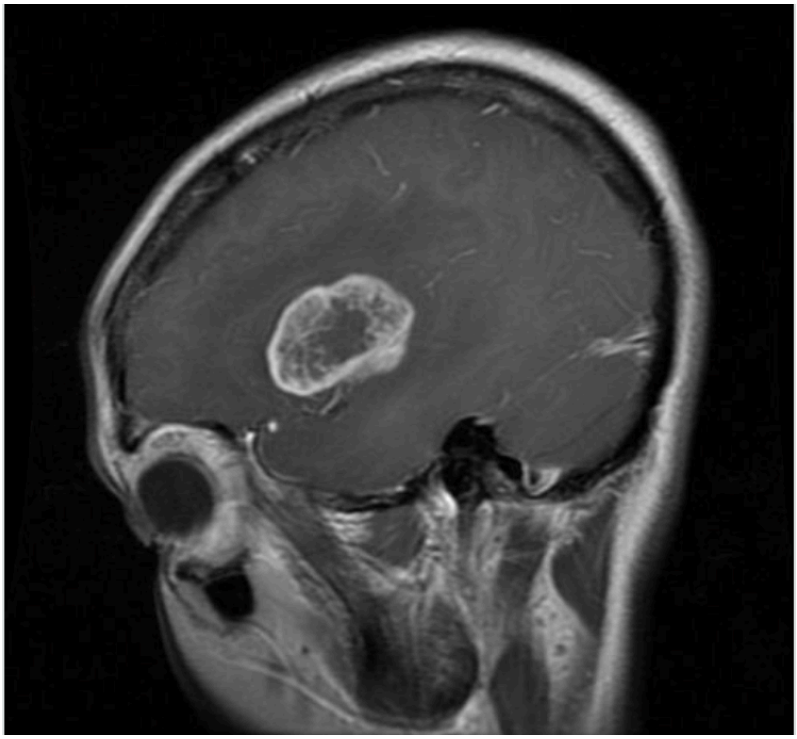
Within this same block of code I can pre-process the images before they are saved into a variable. The code will first take the image and load it into grayscale mode. The reason I chose grayscale mode "IMREAD_GRAYSCALE" is because my images are a combination of black and white MRI images. This is the best way to visualize them. Next the code resizes the images, originally the images I downloaded from kaggle were 2D grayscale images with a resolution of 240x240 pixels. I resized them to 224x224 pixels because the deep learning classification models I will use later on require this amount of pixels. Next I created and applied CLAHE which stands for Contrast Limited Adaptive Histogram Equalization. This technique enhances image contrast by dividing the image into small, localized tiles and applying histogram equalization to each one individually. I used it to bring out subtle features in the grayscale MRI scans such as texture or tumor boundaries without over amplifying noise. This is especially useful in medical images, where details often appear in low contrast regions. By setting clipLimit=2.0 and tileGridSize=(8,8), I ensured a balanced contrast enhancement that preserves local detail across the image. The neural networks I will be using later operate better when input values are small and on a consistent scale so my next step is to normalize the images. In grayscale images each pixel has a value from 0 to 255. Dividing each pixel value by 255.0 will turn each pixel value from 0 to 1 scale. Next each image that has been transformed in the training folder will be appended to a variable called X_train and each label will be appended to y_train. I repeated this step for the testing data folder and added the images to X_test and labels to y_test. These test images and labels I will not train until I have a model I'm confident about and see how it performs on unseen data. All the libraries I used throughout the entire model will be displayed at the very top of the model.

Exploratory Data Analysis (EDA)

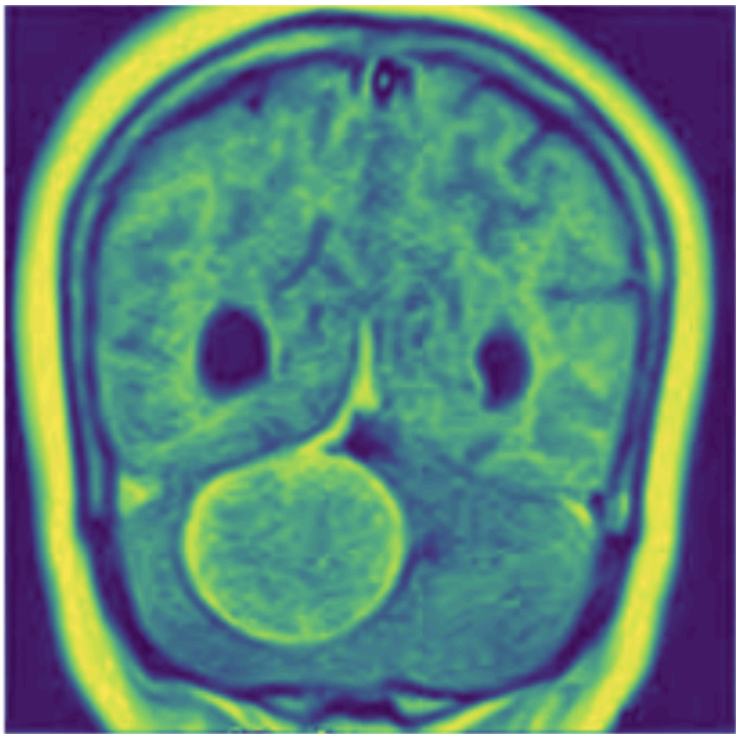
Image Comparison

Now moving on to the second step in my deep learning model is Exploratory Data Analysis. To begin I will print out random images from the training and testing data I just cleaned to see how they look and look out for any anomalies. After reviewing the images they look good and I myself with my human vision can see the images better than they seemed before.

Before



After



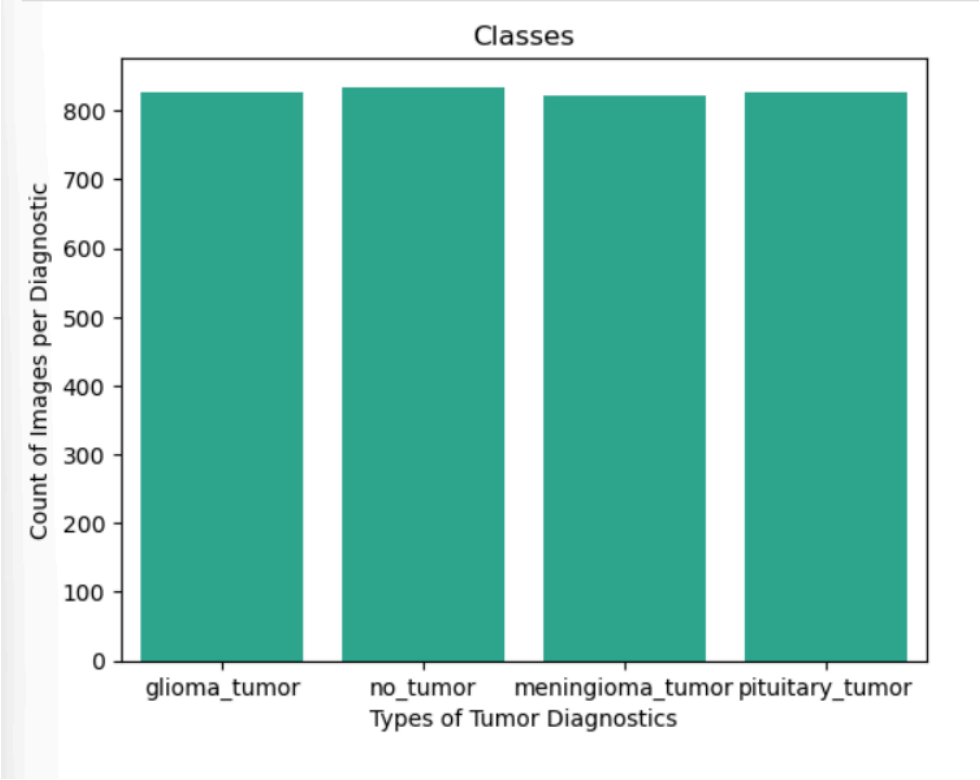
After completing all preprocessing steps, the effects of CLAHE are clearly visible. It enhances the image by selectively increasing brightness in low-contrast regions, which helps emphasize finer details without overexposing brighter areas. This localized contrast adjustment ensures

that both dark and light regions are better defined, preserving important structural features in the MRI scan.

Next I got to know my data a bit more by seeing the shape of each variable was. The training images were a total of 3310 images all 224x224 pixels. The test data was 394 images totaling all 224x224 pixles.

Class Balances

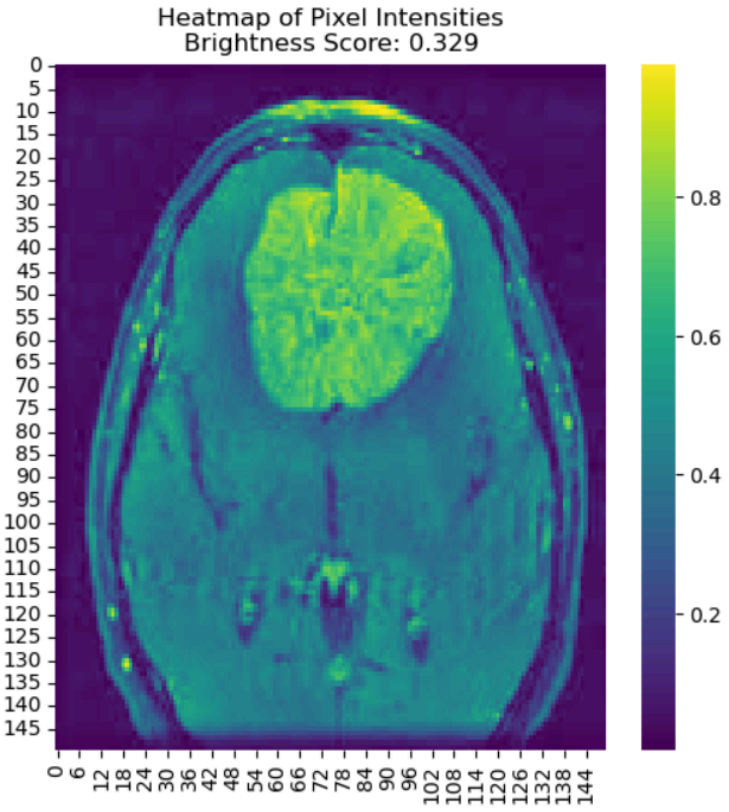
When building a classification model, it's essential to ensure the dataset is class balanced. If one class has significantly more samples than the others, the model can become biased favoring that majority class during training and leading to inaccurate predictions. To address this, I checked the distribution of labels in my training data. All four categories 'glioma', 'meningioma', 'no_tumor', and 'pituitary' had counts within a similar range. This confirms that the dataset is already balanced, so no resampling or augmentation is needed at this stage. Below is a visual representation of the tumor class distributions in the training set.

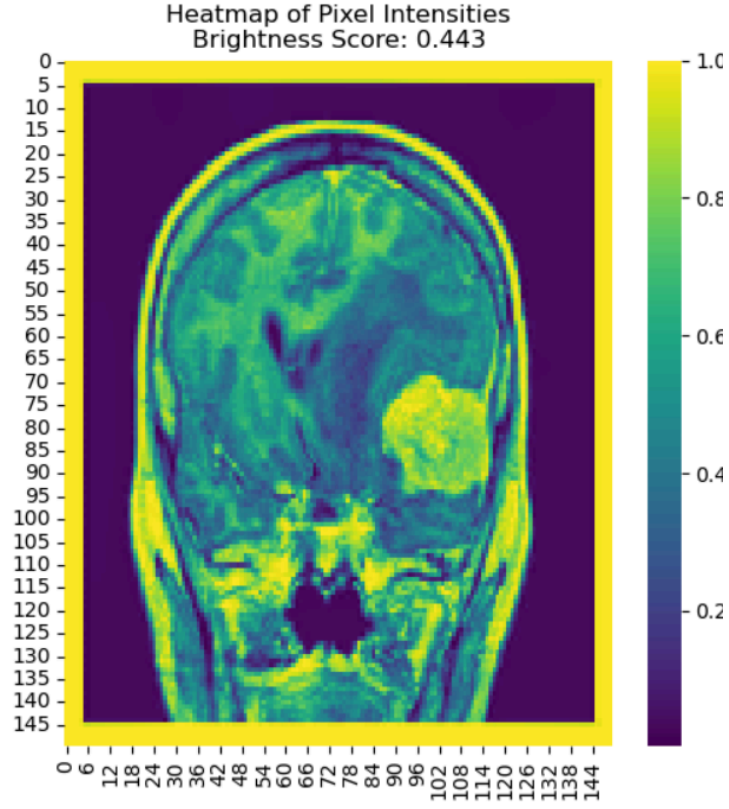


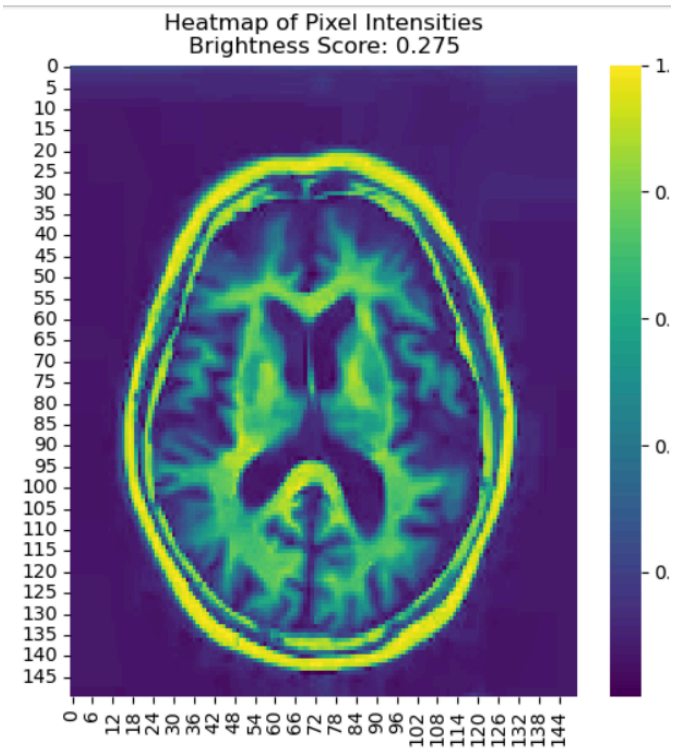
Upon evaluating the testing dataset, I noticed a significant class imbalance. Since the data was pre-split and provided in its current form, I couldn't recombine and redistribute it without introducing bias or compromising experimental integrity. Instead of attempting to rebalance the test set, I'll rely on evaluation metrics that are more robust to class imbalance such as precision, recall, and the F1-score. These metrics provide a more accurate picture of model performance across all categories, especially when the number of samples per class is uneven.

Pixel Intensity Heatmap

Before training my model, I wanted to explore the pixel intensity patterns in the preprocessed images to better understand the data my model will learn from. To do this, I generated pixel intensity heat maps that visually highlight contrast patterns across the brain scans. In these heatmaps, brighter regions represent higher pixel intensities and may correspond to tumor structures, while darker areas typically align with background tissue or non-tumorous regions. This kind of visual inspection helps me interpret how the model might differentiate between classes based on pixel level features. If the model underperforms later in evaluation, I can revisit this step to investigate whether further image enhancement or preprocessing adjustments might improve its performance. Below are three sample heatmaps. In general, higher brightness scores indicate a greater likelihood of tumor presence, making this a valuable lens for understanding model input quality.


Heatmap of Pixel Intensities
Brightness Score: 0.329
0.8
0.6
0.4
0.2


Heatmap of Pixel Intensities
Brightness Score: 0.443
1.0
0.8
0.6
0.4
0.2



As you can see the third image has the lowest heat score and it's obvious that it does not have a tumor. The other two images are higher in the heat score and have an obvious tumor.

Image Transformations

Next, I convert the image data from single channel grayscale to three-channel RGB format. While the original MRI scans are grayscale, most deep learning models especially those pretrained on large datasets like ImageNet expect input images with three color channels (Red, Green, and Blue). This conversion ensures compatibility with those models and maintains consistency in input dimensions during training. I saved these transformed images in new variable names X_train_rgb and X_test_rgb. I also transformed the labels from text to numbers using LabelEncoder() and saved both the encoded and one-hot versions as y_train_encoded, y_train_cat, y_test_encoded, and y_test_cat.

Model Selection & Training

The next step involved selecting the models I would use to train the dataset. I chose three well-established convolutional neural network (CNN) architectures: ResNet50, DenseNet121, and Xception, each known for strong performance in image classification tasks. CNNs are especially effective for image-based problems because they begin with convolutional layers, which scan the input to detect patterns, textures, and shapes. These are followed by fully connected (dense) layers, where the network combines the extracted features to learn more complex representations. This is the core learning phase, where the model refines its

understanding of the data. Finally, the architecture ends with an output layer, which generates a set of classification probabilities indicating the model's confidence in each possible class. These models were previously trained on large scale datasets like ImageNet, allowing them to learn a wide range of visual features such as edges, textures, and shapes. Through transfer learning, I can leverage this pre-learned knowledge and apply it to my own MRI classification task. While the original training data differs from medical images, the lower level features still generalize well, providing a strong foundation for detecting tumor patterns in brain scans.

ResNet50

I began with ResNet50 because my dataset contains only 3,310 images, which isn't a large amount by deep learning standards. With this in mind, I chose a model that's deep enough to learn meaningful features, but not so complex that it risks overfitting. ResNet50 strikes that balance well. As mentioned earlier, this model was originally pretrained on the ImageNet dataset, which includes a wide variety of natural images, everything from zebras to airplanes. While these classes have nothing to do with brain tumors, the low level features it learned (such as edges, textures, and shapes) can still be extremely useful. To adapt ResNet50 to my classification problem, I set include_top=False, which removes the model's final fully connected classification layer. In essence, I'm telling the model, "Don't make predictions based on what you were originally trained to learn,instead classify based on my MRI images".

I then added a custom classification head, beginning with a GlobalAveragePooling2D layer. This layer compresses the 3D output of the convolutional blocks into a 1D vector, allowing the model to summarize each feature map into a single meaningful value which helps focus on the most important regions of each scan. To improve generalization, I used Dropout(0.5), a regularization technique where 50% of neurons are randomly deactivated during each training step. This forces the model to rely on different combinations of features during training, helping it perform better on unseen data. I followed that with a Dense layer containing 128 neurons, activated by ReLU who introduces non-linearity into the model, allowing it to learn complex patterns that wouldn't be possible with just linear transformations. Next, I added a second Dropout(0.25) layer, this time with a dropout rate of 25%. Layering multiple regularization techniques helps prevent overfitting by reducing the model's reliance on specific neurons and encouraging more generalized learning. The final component of my custom classification head is a Dense layer with 4 neurons, where each neuron corresponds to one of the four tumor categories: glioma, meningioma, pituitary, and no tumor. I used a softmax activation function, which transforms the raw outputs from those neurons into probabilities that sum to 1. This makes the output interpretable whichever neuron has the highest probability becomes the model's predicted class for a given MRI scan.

Finally, I instantiated the model and configured it for training by specifying an optimizer, a loss function, and a performance metric, all essential components of any deep learning workflow. I used the Adam optimizer, which adapts the learning rate during training and is well suited for image classification tasks. To measure how well the model is learning, I chose the categorical cross entropy loss function, which is ideal for multi class classification problems like mine.

Categorical cross entropy calculates the difference between the model's predicted probabilities and the actual labels essentially telling the model how far off its predictions are. During training, the optimizer uses this signal to update the network's weights and gradually improve performance. I also tracked accuracy as a metric to get a quick sense of how often the model is classifying tumor types correctly. Then I finally fit and trained the model on the training data X_train_rgb and y_train_cat.

```
res_net_model = ResNet50(include_top =False, weights = 'imagenet',
                         input_shape=(224,224,3))
# Freeze base layers
res_net_model.trainable = False

# Add custom classifier
x = res_net_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation = 'relu')(x)
x = Dropout(0.25)(x)
output = Dense(4, activation='softmax')(x)

model_res_net = Model(inputs=res_net_model.input, outputs=output)

model_res_net.compile(
    optimizer='adam',
   loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

DenseNet121

The next model I explored was DenseNet121 because I wanted to see how a deeper architecture would perform compared to my previous model, ResNet50. While ResNet50 has 50 layers, DenseNet121 contains 121 layers, allowing for potentially richer feature learning. I followed a similar structure to my ResNet based model, but I was more intentional in tuning its parameters. Once again, I set include_top=False to remove the model's default classification head and replace it with a custom classifier tailored to my four tumor categories. In this new classifier, I added a Dense layer with 64 neurons fewer than the 128 I used in the previous model and used the same ReLU activation function. I also included the same Dropout(0.5) rate of 0.5 to help reduce overfitting. The final layer was identical: a Dense layer with 4 neurons (one for each tumor class) and a softmax activation to output class probabilities. I compiled the model using the adam optimizer, the categorical cross-entropy loss function, and accuracy as the performance metric. Then I trained it using the X_train_rgb images and the one-hot encoded

labels in y_train_cat. The purpose of this model was to test how a model with more layers and less neurons would perform compared to a model with less layers and more neurons.

```
dense_model = DenseNet121(
    include_top=False,
    weights='imagenet',
    input_shape=(224, 224, 3),
    pooling='avg'
)

# Freeze base layers
dense_model.trainable = False

# Add custom classifier
x2 = dense_model.output
x2 = layers.Dense(64, activation='relu')(x2)
x2 = layers.Dropout(0.5)(x2)
outputs_2 = layers.Dense(4, activation='softmax')(x2)

model_densenet = models.Model(inputs=dense_model.input, outputs=out

# Compile the model
model_densenet.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Xception

The last model I tried out was Xception, mostly because I was curious to see how something with a different style of learning would do compared to the other two. Unlike ResNet and DenseNet, Xception doesn't follow the usual step-by-step structure. Instead, it takes big image-processing tasks and breaks them into smaller, faster ones. For my MRI scans, that means it looks at each color channel red, green, and blue separately first, then combines everything it finds to make a final call. It’s like it studies the details in pieces, then pulls it all together for the big picture. The way I built this model was still similar to the other two, again I turned off the original classification layer using include_top=False and added my own custom one instead. I used GlobalAveragePooling2D so the model could focus on shapes and textures in the MRI scans while keeping things lightweight. Then I added a Dense layer with 64 neurons and used the ReLU activation again to keep things consistent. I also included a Dropout(0.3) layer with a 30% rate. I wanted to see how it would perform with a bit of regularization, but still keep the number of parameters lower than in the previous models. After that, I wrapped it up

with the same final Dense layer with 4 neurons and softmax activation for multi-class classification. I compiled the model using adam, with categorical cross entropy as the loss and accuracy as the metric to get a quick feel for how it was doing. Then I trained it on X_train_rgb and y_train_cat.

```
xception_model = Xception(
    weights='imagenet',
    include_top=False,
    input_shape=(224, 224, 3))

# Freeze base layers
xception_model.trainable = False

# Add custom classifier
x3 = xception_model.output
x3= GlobalAveragePooling2D()(x3)
x3_= layers.Dense(64, activation='relu')(x3)
x3 = Dropout(0.3)(x3)
output_3 = Dense(4, activation='softmax')(x3)

model_xception = Model(inputs=xception_model.input, outputs=output_3)
model_xception.compile(optimizer= 'adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Model Predictions and Evaluations

Predictions Function

Now the next step is to make predictions with each of the models to evaluate those predictions and see what model is the best performing one. To be more productive and efficient without having to repeat code unnecessarily I created a function that would take a model and training data as inputs. It uses the model to generate prediction probabilities and returns the most likely class labels by applying np.argmax.

Metric Evaluation Function

Then I created a new function called evaluate to help me review how each model was performing. This function took in the model, the true labels, and the predicted labels. I used the classification_report from scikit-learn in it because it gives a breakdown of precision, recall, and F1-score. Precision tells me how many of the model's predictions were actually correct. Recall shows how many of the actual cases the model was able to identify correctly. F1 score gives a balance between precision and recall. Since this project involves classifying medical images

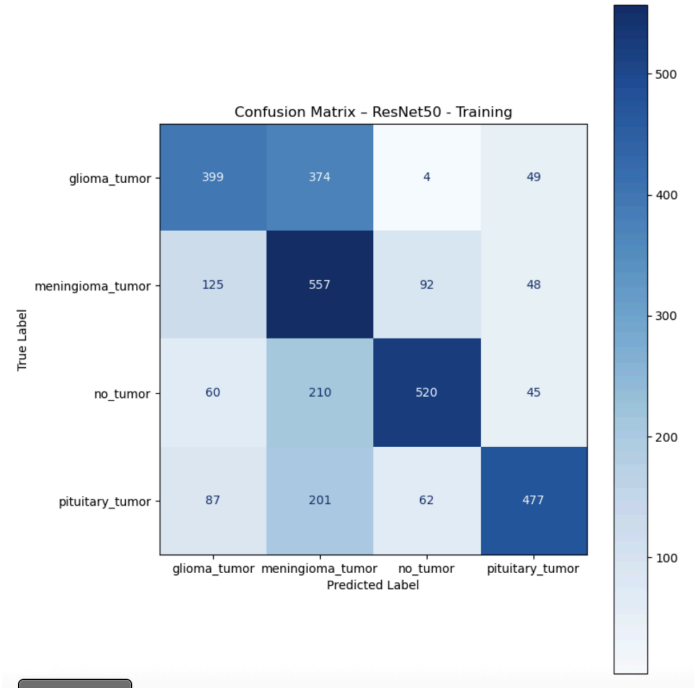
where the outcomes could impact real people and their health I'm especially focused on models that score well high across all three metrics. Once the function has a score I times it by 100 to get a percentage score for all 3 scores.

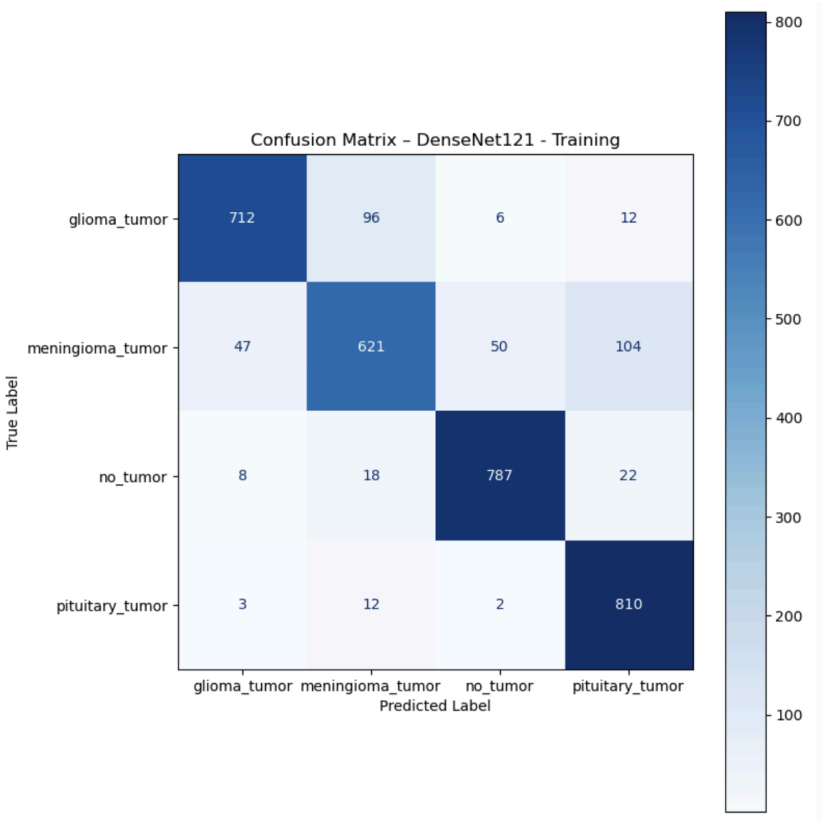
ResNet50, DenseNet121 and Xception Evaluation scores

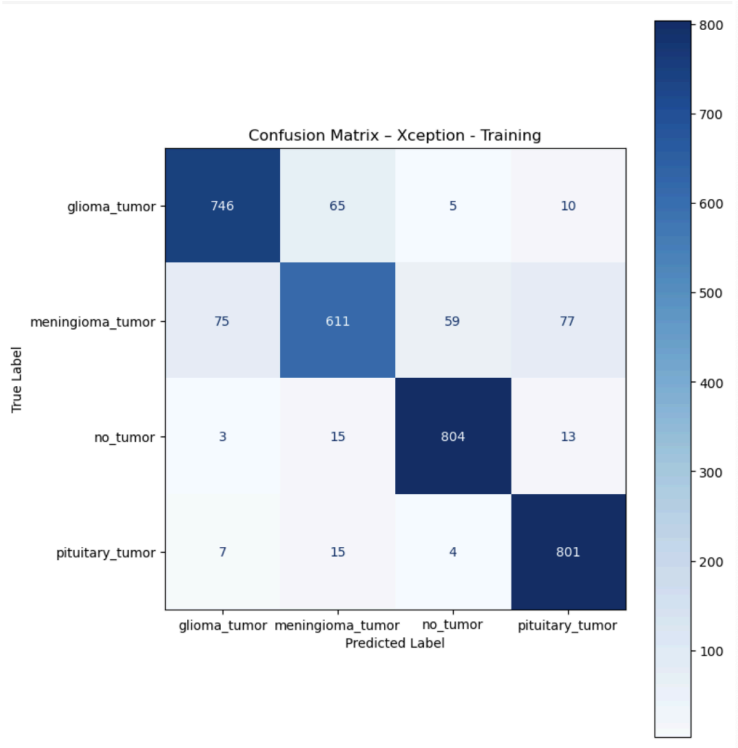
The first model I tested was ResNet50, which had a precision score of 63.68%, a recall score of 59%, and an F1 score of 59.87%. I wasn't happy with those numbers. This kind of task requires accurate predictions, and clearly, using fewer layers with more neurons wasn't the best approach. Next, I evaluated DenseNet121, which gave a much better result: 88.54% precision, 88.49% recall, and an F1 score of 88.34%. I felt good about these scores. They showed the model was capturing patterns a lot more reliably. Then I ran Xception, which performed slightly better than DenseNet121 with 89.35% precision, 89.45% recall, and an F1 score of 89.23%. The difference wasn't massive, but it was noticeable. I also calculated overall accuracy by writing a function that compared the true labels to each model's predictions. The results were: 59% for ResNet50, 88% for DenseNet121, and 89% for Xception. Again, DenseNet121 and Xception performed well and were very close to each other. Based on these results, I'm confident that either model could be fine-tuned further to perform even better.

Confusion Matrix

After I evaluated precision, recall, F1 score and accuracy, I also created a confusion matrix for each model to get a clearer picture of how well they were actually predicting each class.







In each confusion matrix, there's a clear diagonal line running from the top left to the bottom right, those diagonal squares represent the number of images that were correctly classified for each tumor type. The boxes off the diagonal are where the model made mistakes; they show

the images that were misclassified into the wrong categories. While Xception had the strongest overall performance, a closer look revealed that DenseNet121 actually did slightly better at identifying meningioma and pituitary tumors, correctly classifying around 9–10 more images in those classes compared to Xception. The difference wasn't huge, but it was enough to make me curious about whether Xception could match or beat that performance. That's what led me to start fine-tuning the Xception model to see if I could push it even further.

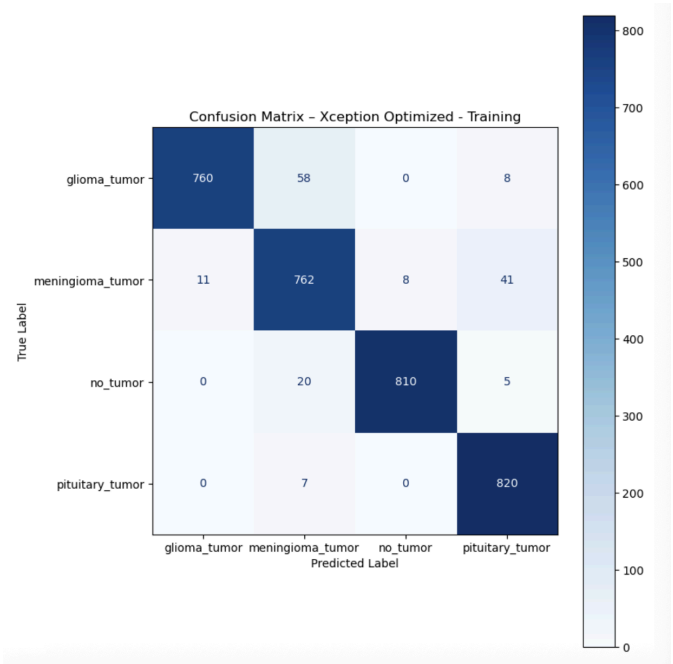
Model Optimization

Manual Optimization

Earlier, I trained my Xception model with just 64 neurons in the first Dense(0.3) layer and a 30% dropout rate. This time, I made some key adjustments, I increased the number of neurons to 256 and dropped the dropout rate to 10%. I chose to manually tune these hyperparameters instead of using something like cross-validation, because I wanted more control over how individual changes impacted performance, especially since the dataset is relatively focused and the model isn't too heavy to retrain.

Aside from those tweaks, everything else in the model stayed the same. I generated a new set of predictions using this updated Xception model and evaluated the results and they were seriously impressive. The model achieved a precision of 95.23%, recall of 95.22%, F1 score of 95.23%, and an accuracy of 95.23% across all classes. Below is a new confusion matrix that clearly shows how the model is predicting more images correctly for all 4 classes.

Confusion Matrix for Exception Model



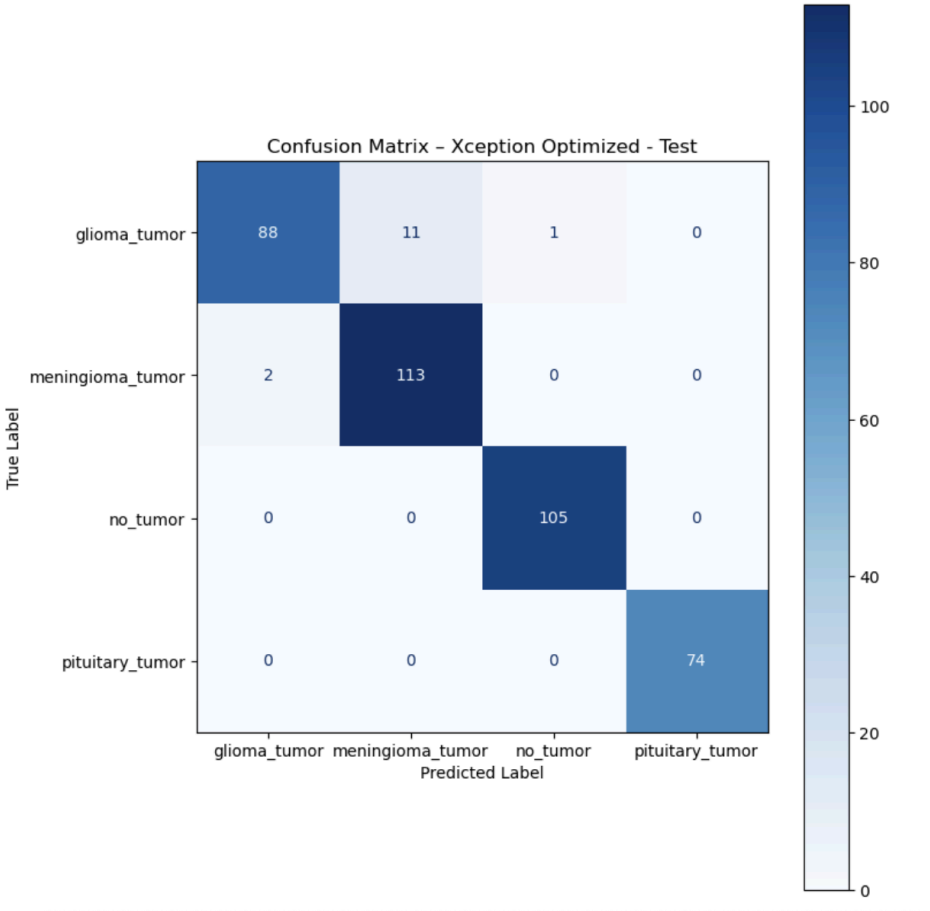
Testing the Model Final Model

Optimized Xception Evaluation scores for Test Data

It’s exciting to see the model performing well, but what really matters is whether it can keep that performance up on unseen data. That’s where my test set of 394 images comes in, which I carefully cleaned and transformed the same way as the training data to keep things consistent. When I evaluated the optimized Xception model on this test set, the results were very close to the training metrics: 97% precision, 96.5% recall, 96.7% F1 score, and 96% accuracy.

These scores are a strong sign that the model is generalizing well and can be trusted to make solid predictions on new data. The confusion matrix for the test set shown below backs this up. It shows the same diagonal trend of correct classifications across all four tumor types, with few misclassifications scattered off the diagonal. This pattern confirms the model isn’t overfitting or underfitting, it's staying consistent on data it hasn’t seen before.

Confusion Matrix for Optimized Xception Model on Test Data



Recommendations

NeuroScan Diagnostics can integrate this deep learning classification model into their workflow to analyze patient brain MRI scans with higher accuracy and efficiency. By letting the model handle the classification, doctors can focus more of their time on developing personalized treatment plans instead of spending hours manually reviewing scans.

Beyond diagnosis, this model could also support research into patient demographics and tumor patterns, helping the team explore why certain individuals are more prone to specific tumor types. Additionally NeuroScan Diagnostics could use the model in a screening tool for early detection, especially in high risk patients.