Brain Tumor Detection and Classification

Rohan Sharma, Saahas Kohli, Shuwang Zhang
University of California, Los Angeles
rohansharma03@g.ucla.edu, skohli2@g.ucla.edu, shuwangzhang@ucla.edu

Abstract

In this project, we aim to evaluate the performance of different deep-learning architectures for the detection and classification of brain tumors in MRI images. We compared a CNN model with transfer learning and a Visual Transformer model. We processed the MRI images, trained both models, and evaluated their performance. By analyzing metrics like accuracy and visualizing model predictions, we aimed to determine which approach works best for this critical medical task. Our findings offer practical insights into choosing the right deep learning technique for brain tumor analysis in MRI images.

1. Introduction

1.1. Background

Brain tumor classification is a critical task in medical image analysis, aiming to accurately differentiate between various types of brain tumors based on imaging data. Accurate classification is essential for timely diagnosis, treatment planning, and patient prognosis.

In recent years, deep learning techniques, particularly Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), have demonstrated remarkable success in various computer vision tasks, including medical image analysis. CNNs have been widely used for brain tumor classification due to their ability to automatically learn hierarchical features from raw image data. These networks excel at capturing local patterns and spatial dependencies, making them well-suited for analyzing medical images. The motivation behind using both CNNs and ViTs for brain tumor classification stems from their complementary strengths. CNNs excel at capturing local patterns and spatial dependencies, while ViTs can effectively capture long-range dependencies and global context.

The goal of this project is to investigate the effectiveness of CNNs and ViTs for brain tumor classification, comparing their performance, and identifying their strengths and limitations. By leveraging the power of deep learning and exploring different architectural choices, we aim to develop an accurate and reliable brain tumor classification system that can assist medical professionals in the diagnosis and treatment planning process. The dataset that we used for this model was Kaggle Brain MRI Dataset [7]. In the following sections, we will delve into the details of preprocessing techniques, model architectures, training procedures, and evaluation metrics used in this project.

1.2. Overview of Model Architectures

The VGG16 architecture utilizes deep convolutional neural networks (CNNs) to extract hierarchical features from input images, which are then processed through fully connected layers for classification. With its proven effectiveness in image recognition tasks, VGG16 offers a strong foundation for feature extraction and classification in medical image analysis, such as brain tumor classification. Our model comprises 16 convolutional layers followed by max-pooling layers, culminating in three fully connected layers. The input layer accepts images of size 256x256 pixels with three color channels (RGB). Convolutional blocks consist of multiple layers with 3x3 filters, followed by max-pooling for downsampling. Global average pooling reduces feature map dimensions to a vector. Dense layers with ReLU activation perform classification, while dropout regularization prevents overfitting [2].

On the other hand, the Visual Transformer (ViT) architecture introduces a novel approach by applying self-attention mechanisms to capture global dependencies among image patches, thereby enabling effective feature extraction and classification without relying on predefined convolutional structures. By leveraging ViT, we can explore the potential benefits of attention-based models in medical imaging tasks, potentially improving classification accuracy and inter-operability [4].

1.3. Motivation Behind the Models

The motivation to use CNN and ViT architectures in brain tumor classification lies in their ability to effectively capture and learn relevant features from medical images. VGG16, with its deep convolutional layers, has demon-

strated remarkable performance in various image recognition tasks, making it a reliable choice for medical image analysis. Additionally, ViT introduces a novel paradigm shift by leveraging attention mechanisms to capture long-range dependencies in images, offering a promising alternative for feature extraction and classification. By employing these architectures, we aim to enhance the accuracy and interpretability of brain tumor classification models, ultimately contributing to improved diagnostic accuracy and patient care in medical imaging [5].

1.4. Model Characteristics to Optimize

In a vision transformer, the input image is divided into smaller patches, each of which is treated as a token. The patch size determines the dimensions of these patches. Larger patch sizes (patch_size) allow the model to capture more global information from the image but may reduce the spatial resolution of the input. Smaller patch sizes enable the model to capture more fine-grained details but may require more computational resources.

Self-attention mechanisms in vision transformers are often implemented with multiple attention heads. Each attention head attends to different parts of the input tokens, allowing the model to capture diverse patterns and relationships. The number of attention heads (n_heads) controls the level of parallelism and the diversity of attention patterns captured by the model.

Vision transformers consist of multiple transformer layers, each containing self-attention and feedforward neural network sublayers. The number of transformers (n_transformers) in the model determines its depth and capacity to learn complex representations from the input image.

Dropout is a regularization technique commonly used in neural networks to prevent overfitting. In vision transformers, the dropout (transformer_dropout) is applied after the self-attention and feedforward sublayers within the transformer layers. It may help to randomly drop out units during training, forcing the model to learn more robust and generalized representations.

Similar to transformer dropout, MLP dropout (mlp_dropout) is applied to the hidden layers of the MLP heads to help regularize the model during training.

The learning rate is a hyperparameter that determines the step size at which the model parameters (weights) are updated during training via optimization algorithms like gradient descent. It controls the magnitude of the updates to the parameters in each iteration of training. A high learning rate may cause the optimization algorithm to overshoot the optimal solution, leading to instability or divergence, while a low learning rate may result in slow convergence or getting stuck in suboptimal solutions. Tuning the learning rate is crucial for achieving optimal training performance and model generalization, so we focus on optimizing it.

The number of epochs (num_epochs) hyperparameter refers to the number of complete passes through the entire training dataset during training. Training for too few epochs may result in underfitting, where the model fails to capture the underlying patterns in the data. Conversely, training for too many epochs may lead to overfitting, where the model memorizes the training data without generalizing well to unseen data. The number of epochs is typically chosen based on the trade-off between underfitting and overfitting, often determined through experimentation or model validation on a separate validation dataset. Thus, this is also something we focus on in our optimization [2].

2. Results

The dataset we used for this project consists of brain MRI images divided into four classes: ”no tumor”, ”pituitary”, ”meningioma”, and ”glioma”, each class describing a certain type of brain tumor shown on the MRI, or no presence of any tumors. This dataset is split into 4571 training samples, 1142 validation samples, and 1311 testing samples.

To address potential issues of class imbalance, we created a helper class called BalancedSparseCategoricalAccuracy(), which helped us to more accurately gauge the performance of our models through weighting accuracy calculations over each class (so that smaller classes still have equal importance in the overall accuracy calculation). From now on, when we mention ”accuracy,” we are referring to this customized metric.

For our CNN model, we decided to use a pre-trained VGG16 model on Imagenet with a few additional custom layers for fine-tuned learning on our dataset, as shown in Section 4.2. After optimizing hyperameters and architectural configurations, we trained a model that obtained a validation accuracy of 91.4%. We then used it to obtain a testing accuracy of 93.2%.

Using the vision transformer architecture and after fine-tuning the hyper-parameters, we were able to train a model that achieved a validation accuracy of 96.4%, and we then used it to obtain a testing accuracy of 94.5%.

The ViT was significantly faster than the CNN model in terms of training time, and yet it still managed to attain better accuracy on the testing dataset of MRI images. It is also a less memory-intensive model than the CNN, as we will discuss later.

3. Discussion

3.1. The Preferred Model

From our empirical results, we can conclude that the ViT model is superior to the CNN model for brain tumor classification. ViT only uses 8,677,412 total parameters compared

to the 15,766,852 total parameters required for CNN, while also providing higher validation and testing accuracy.

3.2. Reasons for Difference in Performance

We must first note the primary architectural difference between the two neural network paradigms. Specifically, vision transformers utilize attention mechanisms, which allow them to capture long-range dependencies in the input images. This enables them to effectively learn global image representations without being limited by the fixed-size receptive fields of convolutional layers, as in CNNs like VGG16. The attention mechanism helps ViTs capture both local and global context, which can be beneficial for understanding complex relationships within images.

Additionally, vision transformers apply self-attention mechanisms across the spatial dimensions of the input images. This self-attention mechanism allows ViTs to focus on relevant parts of the image while aggregating information from all image regions. In contrast, CNNs like VGG16 use convolutional layers that operate on local receptive fields, which may limit their ability to effectively capture global dependencies.

Vision Transformers can certainly be more parameter-efficient compared to CNNs like VGG16, as evidenced by our results. ViTs typically require fewer parameters to achieve comparable performance due to their self-attention mechanism, which allows them to capture long-range dependencies without needing to resort to excessive parameter layers.

However, it's worth noting that the performance of ViTs and CNNs like VGG16 can vary depending on the specific dataset, task, and computational resources available. Additionally, ViTs are relatively newer compared to CNNs and may still require further exploration and refinement to fully exploit their potential in various computer vision tasks.

3.3. Hyperparameter/Architectural Optimization

While fine-tuning the hyper-parameters for ViT, it is notable that increasing the patch size and the number of transformer blocks and heads for the multi-head attention layer seems to yield the most improvement to model performance. Increasing the dropout rate seems to worsen model performance on the validation dataset. This is likely because excessively high dropout rates can hinder the model's ability to learn useful representations from the data. If too many units are dropped out, the model may struggle to generalize well to unseen data, leading to decreased performance on validation or test datasets.

References

[1] K. Simonyan, A. Zisserman (2014) *Very Deep Convolutional Networks for Large-Scale Image Recognition*, https://arxiv.org/abs/1409.1556.

[2] K. He, X. Zhang, S. Ren, J. Sun (2016) *Deep Residual Learning for Image Recognition*, https://arxiv.org/abs/1512.03385.

[3] A. Sharif Razavian, H. Azizpour, J. Sullivan, S. Carlsson (2014) *CNN Features Off-the-Shelf: An Astounding Baseline for Recognition*, https://arxiv.org/abs/1403.6382.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, et al. (2020) *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, https://arxiv.org/abs/2010.11929.

[5] A. Touvron, M. Cord, M. Douze, et al. (2021) *Training data-efficient image transformers distillation through attention*, https://arxiv.org/abs/2012.12877.

[6] M. Caron, P. Bojanowski, A. Joulin, et al. (2021) *Emerging Properties in Self-Supervised Vision Transformers*, https://arxiv.org/abs/2104.14294.

[7] Masoud Nickparvar(Kaggle) *Brain MRI Dataset*, Kaggle. Available at: https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset.

4. Appendix

4.1. Performance of Different Algorithms

Model Type	Hyperparameters/Architecture	Validation Accuracy
VGG-16 Pretrained CNN (1)	learning_rate: 0.0001, num_epochs: 30, mlp_units: [1024, 512], dropout_rate : 0.50,	Validation Accuracy: 89.9%
VGG-16 Pretrained CNN (2)	learning_rate: 0.0002, num_epochs: 10, mlp_units: [1024, 1024], dropout_rate : 0.35,	Validation Accuracy: 90.3%
*VGG-16 Pretrained CNN (3) (obtained highest validation accuracy during optimization and was therefore used to evaluate the test dataset)	learning_rate: 0.0002, num_epochs: 30, mlp_units: [1024, 512], dropout_rate : 0.20,	Validation Accuracy: 91.4% (Testing Accuracy: 93.2%)
Vision Transformer (1)	patch size: 6, n_heads: 2, n_transformers: 1, transformer_dropout: 0.1, mlp_dropout: 0.1 learning rate: 0.001 n_epochs: 10	Validation Accuracy: 94.6%
Vision Transformer (2)	patch size: 10, n_heads: 3, n_transformers : 2, transformer_dropout : 0.4, mlp_dropout : 0.4 learning rate: 0.002 n_epochs: 10	Validation Accuracy: 90.9%
*Vision Transformer (3) (obtained highest validation accuracy during optimization and was therefore used to evaluate the test dataset)	patch size: 10, n_heads: 3, n_transformers : 2, transformer_dropout : 0.1, mlp_dropout : 0.1 learning rate: 0.0005 n_epochs: 20	Validation Accuracy: 96.4% (Testing Accuracy: 94.5%)

4.2. Model Architectures

Model: "model_13"

Layer (type)	Output Shape	Param #
input_14 (InputLayer)	[(None, 64, 64, 3)]	0
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 32, 32, 128)	73856
block2_conv2 (Conv2D)	(None, 32, 32, 128)	147584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
block3_conv1 (Conv2D)	(None, 16, 16, 256)	295168
block3_conv2 (Conv2D)	(None, 16, 16, 256)	590080
block3_conv3 (Conv2D)	(None, 16, 16, 256)	590080
block3_pool (MaxPooling2D)	(None, 8, 8, 256)	0
block4_conv1 (Conv2D)	(None, 8, 8, 512)	1180160
block4_conv2 (Conv2D)	(None, 8, 8, 512)	2359808
block4_conv3 (Conv2D)	(None, 8, 8, 512)	2359808
block4_pool (MaxPooling2D)	(None, 4, 4, 512)	0
block5_conv1 (Conv2D)	(None, 4, 4, 512)	2359808
block5_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block5_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block5_pool (MaxPooling2D)	(None, 2, 2, 512)	0
global_average_pooling2d_13 (GlobalAveragePooling2D)	(None, 512)	0
dense_39 (Dense)	(None, 1024)	525312
dense_40 (Dense)	(None, 512)	524800
dropout_13 (Dropout)	(None, 512)	0
dense_41 (Dense)	(None, 4)	2052

Total params: 15766852 (60.15 MB)
Trainable params: 1052164 (4.01 MB)
Non-trainable params: 14714688 (56.13 MB)

The above image displays the architecture for the VGG-16 pretrained CNN model. "Relu" activation functions were used in the Dense layers (the last Dense layer was simply linear). The Adam optimizer was also used, as well as categorical crossentropy for the loss function.

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 64, 64, 3)]	0	[]
patches (Patches)	(None, None, 108)	0	['input_1[0][0]']
patch_encoder (PatchEncoder)	(None, 100, 32)	6688	['patches[0][0]']
layer_normalization (LayerNormalization)	(None, 100, 32)	64	['patch_encoder[0][0]']
multi_head_attention (MultiHeadAttention)	(None, 100, 32)	8416	['layer_normalization[0][0]', 'layer_normalization[0][0]']
add (Add)	(None, 100, 32)	0	['multi_head_attention[0][0]', 'patch_encoder[0][0]']
layer_normalization_1 (LayerNormalization)	(None, 100, 32)	64	['add[0][0]']
dense_1 (Dense)	(None, 100, 64)	2112	['layer_normalization_1[0][0]']
dropout (Dropout)	(None, 100, 64)	0	['dense_1[0][0]']
dense_2 (Dense)	(None, 100, 32)	2080	['dropout[0][0]']
dropout_1 (Dropout)	(None, 100, 32)	0	['dense_2[0][0]']
add_1 (Add)	(None, 100, 32)	0	['dropout_1[0][0]', 'add[0][0]']
layer_normalization_2 (LayerNormalization)	(None, 100, 32)	64	['add_1[0][0]']
flatten (Flatten)	(None, 3200)	0	['layer_normalization_2[0][0]']
dropout_2 (Dropout)	(None, 3200)	0	['flatten[0][0]']
dense_3 (Dense)	(None, 2048)	6555648	['dropout_2[0][0]']
dropout_3 (Dropout)	(None, 2048)	0	['dense_3[0][0]']
dense_4 (Dense)	(None, 1024)	2098176	['dropout_3[0][0]']
dropout_4 (Dropout)	(None, 1024)	0	['dense_4[0][0]']
dense_5 (Dense)	(None, 4)	4100	['dropout_4[0][0]']

Total params: 8677412 (33.10 MB)
Trainable params: 8677412 (33.10 MB)
Non-trainable params: 0 (0.00 Byte)

The above image displays the architecture for the ViT model. "Gelu" activation functions were used in the intermediate dense layers (not at the end). The Adam optimizer was also used. The Adam optimizer was also used, as well as categorical crossentropy for the loss function.