Brain Tumor Classification

Learning Machines - Cheng Zhengxing, Ma Yuhan, Sun Chang, Zhang Tianyu, Zhou Runbing

Academic Year 2021/2022, Semester I

Abstract

Brain tumor has long been recognized as one of the most vicious diseases faced by mankind. Accurate classification of tumor type is vital for patients' recoveries. This paper aims at deriving robust Machine Learning models serving as better alternatives of the error-prone manual classification. In the experiment, we trained 5 machine learning models on 2870 MRI scans with 4 predefined tumor classes. Judging on their performances on 394 testing images, ResNet34 yielded the highest accuracy of 80.4%, while K-Nearest Neighbor, Support Vector Machine, Random Forest and Vision Transformer methods have accuracies 79.4% 71.1%, 78.7% and 60.2% respectively. Methodologies and models are justified accordingly, and further improvements are discussed within the conclusion.

Contents

1 Literature Review

Brain tumor has long been recognized as one of the most vicious diseases faced by mankind. According to National Cancer Institute of USA, around $250,000$ people worldwide suffer from primary brain tumor annually, and the average 5-year survival rate of malignant brain cancers is merely 33% [14]. The disease has posed great threats to all age groups, from children to elderly citizens. International Agency for Research on Cancer reported the mortality rate due to brain cancer is the highest across the Asian continent [21].

Many different types of brain tumors exist, while medical treatments vary for different types of tumor [13]. If a brain tumor is misdiagnosed, the narrow window of treatment would easily be missed, leading to fatal consequences. Therefore, the classification of brain tumors is particularly essential for society.

Currently, the most effective and well-referenced diagnostic tool is Magnetic Resonance Imaging (MRI)[8]. However, the traditional manual classification of MRI scans is error-prone owing to the complex characteristics of brain tumors and the susceptible subjectivity of neurosurgeons. To make matters worse, the lack of clinical experience adds challenges for doctors to interpret MRI reports and identify tumor categories correctly [1].

The past decade has witnessed a rise in utilizing Machine Learning (ML) tools to classify brain tumors due to its priority in accuracy and time efficiency [2]. In this project, we focused on constructing robust and reliable ML models as practical alternatives for brain tumor classification. A total of five ML models are investigated: K-Nearest Neighbours(KNN), Support Vector Machine(SVM), Random Forest(RF), Residual Neural Network(ResNet) and Vision Transformer(ViT). The details and performances of the model will be discussed in following sections.

2 Data Features

2.1 Dataset Description

Our choice of data is the Brain Tumor Classification (MRI) dataset from kaggle [3]. The dataset comprises pre-splited 2870 pieces of training data and 394 pieces of test data. Each piece of data is an RGB image of MRI scan in JPEG format. The detailed split of data are listed below:

Class	Training images	Testing images
Glioma	826	100
Meningioma	822	115
Pituitary	827	74
None	395	105

Table 1: Images amount for each class

2.2 Feature Engineering

Our response variable is the categorical variable *class*, with four possible outcomes: *glioma tumor*, *meningioma tumor*, *no tumor* and *pituitary tumor*. Predictors are the pixels of each image. For traditional models, numbers of predictors selected are unified to 224 pixels by 224 pixels for computational efficiency and easy comparison of performances. We used grey mode in traditional models and RGB mode in modern models.

2.3 Exploratory Data Analysis

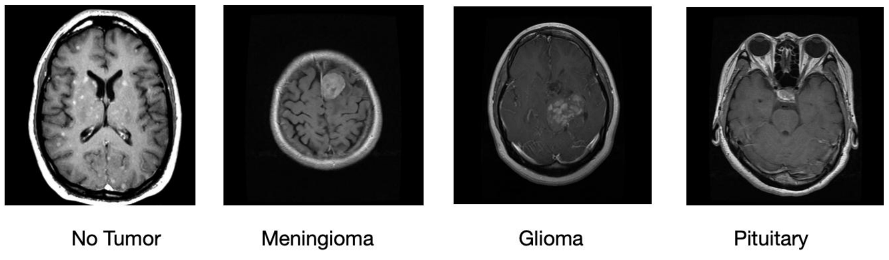


Figure 1: Sample Brain MRIs with Tumors

A brief understanding about the left three tumors:

- Gliomas are tumors in the central nervous system (brain or spinal cord) and peripheral nervous system that form out of various types of glial cells (neuroglia).
- Meningioma develops on the brain's meninges and is a tumor of leptomeningeal origin.
- Pituitary tumor occurs in the anterior body of the pituitary gland. This is a pea-sized gland that is located behind the bridge of your nose.

2.4 Data Processing

For KNN, SVM and RF, images are processed as follows:

- Center-cropped pixels with dimension (224, 224).
- Assigned each image with an integer in range [0, 3], representing the tumor class it belongs to.
- Flattened each image into a 1-D array of length 50176.
- For KNN and SVM, we also applied standardization on predictors to normalise and center the data.

For ResNet34 and ViT, in order to cater to transfer learning requirements, we transformed our input images in the same way as the transformation in the pre-trained model [9]. We center-cropped pixels with dimension (224, 224) and normalize the data according to the pre-trained model.

3 Traditional Approaches

In this section, we considered two measurements of model performance: F1 weighted score and accuracy. Accuracy focuses only on the true positives and true negatives while F1 score takes into account of the false negatives and false positives. F1 score is calculated as:

$$F1\_Score = \frac{2 \times (precision \times recall)}{precision + recall}.$$

By calculating the harmonic mean between precision and recall, F1 score ensures considerations of incorrect classifications and penalizes extreme values. The weighted F1 score, which is the weighted average of the F1 scores of all classes, is adopted due to the imbalanced class distribution of our dataset.

For each model, the parameter-tuning was done twice for both measurements, and the results are compared to ensure the optimal model to be well-rounded.

3.1 K-Nearest Neighbours (KNN)

3.1.1 Definition

KNN is a non-parametric machine learning method available for both regression and classification. With an observation x, its neighbourhood $N(x)$ is the set of K data points closest to x or the set of all data points within a fixed distance K from x. The classsified object that is assigned to the most common class among its k nearest neighbors.

3.1.2 Model Building and Performance

We first normalized our data before training. We performed dimension reduction with principal component analysis (PCA) to reduce the number of features and avoid overfitting before training the actual model, but we considered the proportion of variance explained as a hyperparameter and tuned it by looping through candidate values (0.9 to 0.99 with step 0.01 for the parameter *n_components* in the *sklearn.decomposition.pca* function). We set the lower bound of *n_components* as 0.9 to avoid using too few features which might result in underfitting. Within each loop, we used grid-search to exhaustively search all possible combinations of other hyperparameters using 5-fold cross validation to come up with the best values. We repeated the tuning process twice using weighted F1 score and accuracy. Finally, we obtained the best model by comparing the best models under each value of variance explained and the optimal hyperparameters are shown in Table 2.

Hyper-parameters	Value
n_component in PCA	0.9
n_neighbors	1

Table 2: Optimal Hyperparameters (KNN) Obtained from Grid Search

n_components in PCA=0.9 and *n_neighbors*=1 were chosen with the highest accuracy and highest weighted F1 score. This optimal KNN model achieved a weighted F1 score of 0.7741 and an accuracy of 0.7944 on test data.

KNN is a simple instance-based learning algorithm. KNN finds the nearest neighbours by Euclidean distance which requires scaling of data. Training KNN requires large memory to store the training data set for prediction tasks and its testing phase is also relatively slow and costly.

3.2 Support Vector Machine (SVM)

3.2.1 Definition

SVM is a supervised machine learning algorithm for classification. We applied multi-class SVM classification model and trialed different kernels to optimize model performance.

3.2.2 Model Building and Performance

Similar to KNN, we first normalised the data and performed a looping process (*n_components* from 0.9 to 0.99 with step 0.01). Within each loop, we exhaustively searched all combinations of SVM hyperparameters using 5-fold validation. Finally, we obtained the best model by comparing the best models under each value of variance explained. We then used the function *sklearn.svm.SVC* to implement the “one-versus-one” approach for multi-class classification, where in total $\frac{(n\ classes)\times(n\ classes-1)}{2}$ classifiers are constructed and each classifier trains data from two classes.

The most crucial step of model tuning is the selection of the kernel function (Linear, Polynomial, Gaussian, Radial Basis Function (RBF), Sigmoid in the cross validation stage) and hyperparameters. This step largely determines the smoothness and efficiency of the class separation.

The selection process regarding hyperparameters is then discussed. A variety of parameters could be embedded into the SVC function. For instance, the most significant kernel function, the regularization parameter, the degree of the function if the kernel was chosen as polynomial, the gamma choice as either ‘auto’ or ‘scale’ which normally comes together with RBF kernel, etc.

After some preliminary trials, we chose the following hyperparameters for tuning: the choice of kernel function, the regularization parameter C addressing overfitting, and the degree of the polynomial kernel function which would be automatically ignored if other kernels are used. The reason for not including the parameter ‘gamma’ which normally comes with kernel RBF is that after testing on the performance of the model with and without this parameter which by default will be ‘scale’, we found that there is almost no difference in between. In contrast, we included the parameter ‘degree’ is to avoid the situation that polynomial kernel function was chosen whereas no degree was clearly indicated.

The appropriate selection of parameters is crucial to the performance of SVM. Specifically, lower values of C make the decision surface smooth whereas higher C try to correctly classify all the training data. Also, the more accurate of the degree subject to the polynomial kernel, the more fitted our model would be. We applied grid-search to obtain good hyperparameter values:

Hyper-parameters	Value
n_component in PCA	0.9
Kernel	RBF
C (Regularisation)	3
Degree	4

Table 3: Optimal Hyperparameters (SVM) Obtained from Grid Search

We finalized our model with the above hyperparameters, and the SVM model obtained 0.708 weighted F1 score and 0.711 accuracy on test data.

3.3 Random Forest(RF)

3.3.1 Definition

Random Forest is an ensemble method available for both regression and classification. To construct RF, each tree uses a subset of m predictors on each split and is trained on a bootstrapped sample. A subset of m predictors is randomly selected for the next split and plurality vote is used to combine each tree’s classification into the final classification.

The hyperparameters used in our model are defined as:

- $n\_estimators$: define how many trees were there in the forest.
- $min\_samples\_split$: the fewest samples needed to split an internal leaf node.
- $min\_samples\_leaf$: the least number of samples that were needed at a leaf node.
- $min\_impurity\_decrease$: split would happen to a node if this split made the impurity decrease more than or equal to this number.
- max_depth: the maximum depth of the tree. If none, expansion will be happened to the node until all leaves become pure or less than min_samples_leaf.
- $max\_features$: max_features features at each split if it is an integer; $round(max\_features * n\_features)$ features at each split if it is a float; $max\_features = sqrt(n\_features)$ if it is "auto"; $max\_features = sqrt(n\_features)$ if it is "sqrt"; $max\_features = log_2(n\_features)$ if it is "$log2$"; $max\_features = n\_features$ if it is none.

3.3.2 Model Building and Performance

To save time, we did a two-phase model tuning. In the first phase, we set the $n\_components$ of PCA as 0.9 and trained an RF using grid-search with out-of-bag (OOB) error. The optimal hyperparameters are shown below:

Hyper-parameters	Value
$n\_estimators$	800
$min\_samples\_split$	2
$min\_samples\_leaf$	1
$min\_impurity\_decrease$	0
max_depth	None
$max\_features$	auto

Table 4: Optimal Hyperparameters (RF) Obtained from Grid Search

Theoretically, the test error will decrease if $n\_estimators$ increases. However, in practice, we found that the validation score decreased as $n\_estimators$ increased after a threshold. We obtained $n\_estimators = 800$ as the optimal value by grid-search. With the optimal hyperparameters taking values in the Table 4, our model achieved a weighted F1 score of 0.7390 and an accuracy of 0.7868.

The second phase is to loop the PCA hyperparameter ($n\_components$) from 0.9 to 0.99 with step 0.01, and within each loop we adopted fixed hyperparameter values in the above table. We observed that both the weighted F1 score and the accuracy decreased as the PCA $n\_components$ value increased. As a result, we chose $PCA = 0.9$ for our final model.

Random forest employs a rule-based method and does not require data normalization. It also relieves the over-fitting problem resulted from decision trees. However, it requires massive computational power and time since it builds numerous trees and combines their outputs to determine the classes.

3.4 Miscellany

As observed from the results for all three models above, the weighted F1 scores are slightly lower than the normal accuracy for all cases, which makes sense because the normal accuracy does not stress on false positive and false negative.

In our attached python code, the range for looping PCA hyperparameter $n\_components$ is not completely alined with our description in the above sections. This is because we performed several preliminary model building rounds before this final version of code which include the full range as stated above. However, in order for the code to run smoothly in one run with our limited computing resources, we reduced the range to a smaller value.

4 Modern Approaches

4.1 Residual Neural Network(ResNet)

4.1.1 Introduction to Convolutional Neural Network (CNN)

LeCun et al. [11] first constructed neural network layers with convolutional kernels, which allows the network to be fed directly with three-dimensional images rather than flattened feature vectors. In the following years, several classic CNN structures (LeNet, AlexNet, VGGNet, InceptionNet, ResNet, etc.) have been developed and shown promising prospects in image classification tasks.

Generally, the core module of CNN consists of four parts - convolution, batch normalization, activation and pooling. A number of basic modules with different designs stack together, followed by fully connected layers leading to the output. The convolutional layers are essentially feature extractors [11]. A kernel with a uniform set of weights slides through the three-dimensional image, attempting to extract particular features. Multiple feature maps obtained from multiple kernels are stacked together and then used for further feature extraction in subsequent layers. The feature maps from the last convolutional module are flattened and become the input features of the fully connected layers [11].

Compared with dense neural network, CNN not only significantly reduces the number of parameters by using the shared kernels, but also preserves the relative location information of the raw images. With this feature, CNN is also promising in the field of object detection. Recent development includes the Fast R-CNN by Girshick[5] and advancements proposed by other groups. However, CNN also has some shortcomings. Firstly, the pooling layers will cause loss of some details and information between the part and the whole image. Secondly, CNN only has translation invariance, which is determined by the design of convolutional layer and the pooling layer, but if the feature to be detected is slightly tilted in orientation, the trained kernel might not be able to discern it. These problems could be alleviated by using larger training dataset, but could not be eradicated.

4.1.2 Introduction to Residual Network (ResNet)

The only difference between the structure of a ResNet and that of a generic CNN is that, in ResNet, several core modules are grouped together, and one copy of the input identity is added to the output before the activation layer of the last module [6]. In this way, this group of modules would learn the function of $F(x) - x$ instead of the original task $F(x)$ [6].

He et al.[6] suggests that the current theoretical and empirical development of CNN both reveals a positive correlation between the depth of the convolutional network and its feature extracting capacity. However, very deep convolutional networks may suffer a degradation problem, where training accuracy culminates at a certain depth and then deteriorates with additional layers [6]. He et al.'s residual learning framework [6] empirically resolves this issue by lightening the optimization task for each residual module which attempts to learn the underlying function from a baseline of identity rather than from nothing. With this structure, it is feasible to increase the depth of the network to a large extent and thus improve CNN's ability to extract different level of features.

He et al.[6] introduced 5 ResNet structures of depth 18, 34, 50, 101 and 152 respectively, with the former two having a different residual structure from the latter three. Specifically, the structure of ResNet34, which we exploited for our classification task, is shown as follows [6]:

layer name	output size	kernel dim, no., stride	mods/grp	grp/layer
conv1	112×112	$7 \times 7, 64, 2$	1	*nil*
conv2prelim	56×56	$3 \times 3, maxpool, 2$	1	*nil*
conv2	56×56	$3 \times 3, 64, 1$	2	3
conv3	28×28	$3 \times 3, 128, 1$	2	4
conv4	14×14	$3 \times 3, 256, 1$	2	6
conv5	7×7	$3 \times 3, 512, 1$	2	3
endings	average pool, fc 1000, softmax			

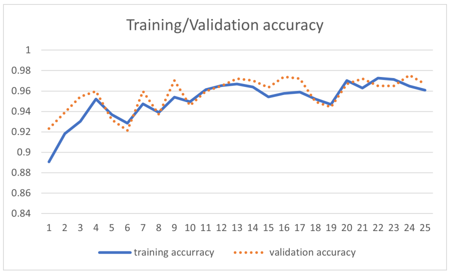
Table 5: ResNet34 Structure (kernel dim: the dimension of the kernel; no.: the number of kernels; mods/grp: the number of basic convolutional module per group; grp/layer: the number of residual groups each layer and the residual structure happens once each group; fc 1000: a fully connected layer containing 1000 units)

The core module of CNN includes a Batch Normalization (BN) layer between the convolutional layer and the activation layer [6]. It normalizes the output of the convolutional layer according to the mean and variance of every individual value of the output matrices in one batch of data. By minimizing the covariate shift in each intermediate layer, BN not only increases the training speed [7], but also alleviates the vanishing or exploding gradient problem [6]. In addition, BN with random allocation of samples into batches results in different normalization for that layer per training epoch. This regularization effect obviates the need for Dropout layers [7].

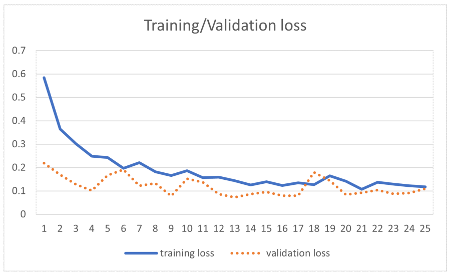
4.1.3 Model Building and Performance

We adopted the ResNet34 model developed by PyTorch Team [17] and trained it on our training data. Our model utilized transfer learning and initialized the weights in the convolutional layers as the pre-trained path [19] generated on ImageNet data [9]. We transformed our input images in the same way as the transformation in the pre-trained model [18] (different transformations for training and validation images).

We used a batch size of 16 in consideration of our computing resources and a small learning rate of 0.0001 since we adopted transfer learning where the pre-trained model already converged. The model converged fast and achieved a good validation accuracy of 0.954625 at the third epoch. We iterated 25 epochs on the training data, among which the validation maximized on the 24th epoch at 0.975567. The training loss decreased and displayed a converging trend towards 0.1. We finalized our model with the weights generated from the 24th epoch, with which the accuracy on our test set achieved 0.8046.



(a) Training/Validation Accuracy



(b) Training/Validation Loss

Figure 2: ResNet34 Model Training Data on 25 epochs

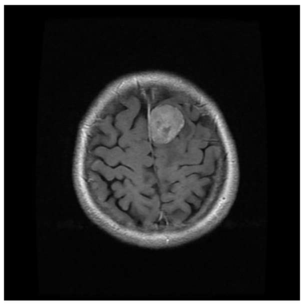
Predicted/Observed	glioma	meningioma	pituitary	none
glioma	37	0	0	0
meningioma	48	115	8	0
pituitary	0	0	60	0
none	15	0	6	105

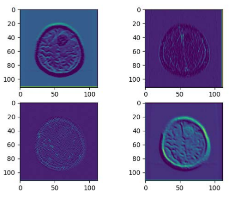
Table 6: Confusion Matrix for ResNet34 Model

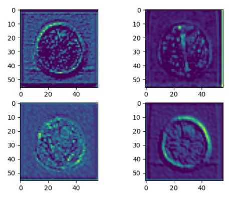
We noticed a disparity between validation accuracy and test accuracy. From the confusion matrix of test data, we could observe errors mainly arised from the class of glioma_tumor. Since our downloaded training and test dataset were pre-separated, this disparity might result from the heterogeneity of data between training and test set. Since our test data is visible, we observed that there is a considerable portion of MRI images in test data for glioma_tumor that are relatively bright, which is a lacking feature of the training set of glioma_tumor and pituitary_tumor but present in that of both meningioma_tumor and no_tumor. In addition, from a medical point of view, MRI images glioma_tumor have more features similar to meningioma_tumor. All of the above analysis would explain the disparity in validation and test accuracy and the model's relative poor performance in predicting glioma_tumor.

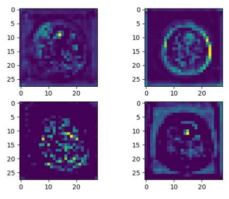
4.1.4 Visualization of model structure

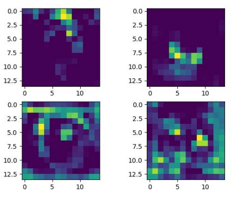
In this section we visualize the feature map produced by convolutional layers taking a meningioma_tumor image from our training set as an example. From the figures, we could see that shallow layers generally learn the texture of the image, while deep layers further extract more abstract features.


(a) input image


(b) conv1 output


(c) conv2 output


(d) conv3 output


(e) conv4 output

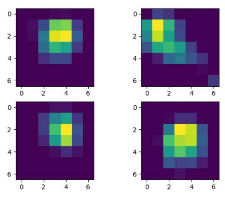

(f) conv5 output

Figure 3: Four Feature Maps from conv1-conv5 after Activation

4.2 Vision Transformer(ViT)

Vision Transformer (ViT) is a state-of-the-art in computer vision field and it is commonly used in image classification tasks. Transformer was originally proposed for natural language processing (NLP) and has been a great success in the NLP field. Therefore, Alexey Dosovitskiy and his team, managed to apply self-attention-based transformer to computer vision for image recognition and, Vision Transformer model was created in 2020 [4]. ViT performs well if pre-trained with sufficiently large datasets on image classification tasks. Therefore, we applied the ViT model to our classification task.

4.2.1 Introduction to Transformer

As ViT was created on the basis of Transformer, we need to introduce the concept of Transformer first. Visual Attention was firstly proposed in 2014 by Google Mind[12] that used recurrent neural network (RNN) and it was popular in deep learning fields. However, RNN in Visual Attention had restrictions in parallel processing capability and information could be lost during sequential processing. In 2017, Transformer was published by Google [20] mainly targeting for sequence modeling and NLP. The traditional RNN architecture was abandoned in the Transformer, and the entire network structure was composed mostly of Attention mechanisms (Multi-Head-Attention). The Transformer model architecture is illustrated below.

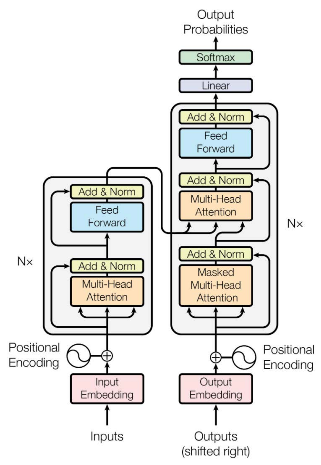


Figure 4: Transformer model architecture[4]

The Transformer consists of the Encoder and the Decoder. First, inputs go into an embedding layer and each word is mapped to the continuous value which represents the word. To capture the meaning of sequential sequences, the Position Embedding part adds the positional information into the embedding after Input Embedding. In the network block of Encoder, it consists of a multi-head-Attention sub-layer and a fully-connected neural network sub-layer. The whole Encoder is stacked N times in Transformer repeatedly. The Output Embedding uses the previous output from the Encoder followed by a Positional Embedding. The Decoder part is similar to the Encoder part, except that the Decoder part has one more masked multi-headed attention sub-layer. The whole Encoder is also stacked N times in Transformer. The output passes through a linear layer. After that, the final predictions are made through the Softmax function. In addition, there are residual connections and normalization of layers (Add & Norm) to improve the performance.

One of the most significant compositions of the Transformer is the multi-headed attention sub-layer. The attention mechanism will learn from the input sequences and find the important parts of the sequence. Multi-headed attention was proposed based on Self-Attention. The formula of Self-Attention is:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}V)$$

Q represents Query which will subsequently match with Key, K represents Key and, V represents Value that extracted from the input value. d_K represents the dimension of each row vector in Key. Dividing by d_K is to scale down the value of dot products among three matrices. If the value of dot products becomes too large, the gradient will be too small after passing through the Softmax function. The matrices of Query, Key, and Value are all derived from the input X which is a sequence.

Multi-Head Attention is a more advanced version of Self-Attention. Multi-Head Attention learns information from different representation subspaces by using different $Q/K/V$ weight matrices. Then we can get multiple attention matrices relating to different $Q/K/V$ weights and then concatenate them with a new weight matrix W^0 to produce the final output. The Multi-Head Attention has more comprehensive understandings as different combinations of $Q/K/V$ contain information in different perspectives. The formula of Multi-Head Attention is:

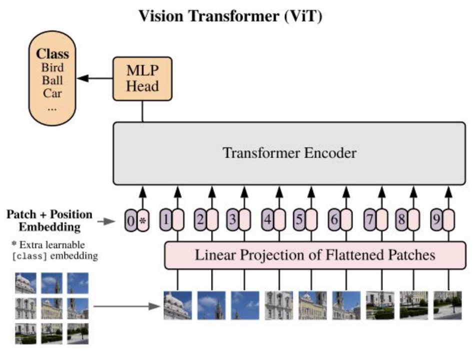
$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^0$$

$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

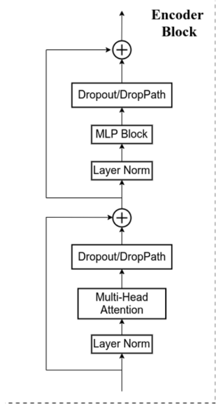
The Transformer can be used for translation or summarizing text in NLP field. One of the outstanding advantages of the Transformer is the ability of parallel processing. Through repeatedly stacking Encoder and Decoder, Transformer learns from different combinations of attentions. Therefore its prediction ability is improved.

4.2.2 Introduction to Vision Transformer Architecture

Vision Transformer consists of three parts which are Linear Projection of Flattened Patches, Transformer Encoder and MLP Head. The overall architecture graphs are shown in the following graphs.


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

(a) ViT model architecture



(b) Transformer Encoder

Figure 5: ViT [4] [16]

Linear Projection of Flattened Patches: First, the images should be transformed to a suitable form that can enter the Transformer Encoder. For a standard Transformer, the input should be a token sequence containing flattened 2-Dimension patches. However, the dimension of the data matrix of the graph is $H * W * C$ which is three. In the previous dimension formula, (H, W) represents the resolution of input images and C represents the number of channels of the graph (RGB graph has 3 channels), the patch embedding is performed to transform the data. First, the algorithm divides the picture into a bunch of smaller patches of a given size and then maps each patch to a one-dimensional vector by a linear mapping. Before the sequence of embedded patches is transferred into the Transform Encoder, class token and position embedding should be added.

Transformer Encoder: It is similar to the Encoder in the Transformer that we introduced in the previous chapter. It will map the input sequences into abstract continuous representations of the same size. The continuous representations contain information that learns from the entire sequence. The detailed flowchart of Transformer Encoder was shown in Figure 5(b). Transformer Encoder consists of a Multilayer Perceptrons (MLP) sub-layer and a Multi-Headed Attention sub-layer. The Encoder block is repeatedly stacked L times in ViT. Layer Norm was a normalization method mainly proposed for the NLP field. Layer Normalization can improve the training speed and accuracy of the model, making it more robust. To reduce overfitting, Dropout or DroupPath was added in the Transformer Encoder. Moreover, Multi-headed attention has been introduced in the previous chapter. Multilayer Perceptrons (MLP) Block was a type of neural network composed of a fully connected linear layer, GELU activation function, and dropout.

MLP Head: MLP head mainly consists of a pre-logit function and a linear layer. The shape of the output remained the same as the input shape after passing through the Transformer Encoder. As a result, the final classification result is obtained through the MLP head.

Vision transformer succeeds in classifying the images through going through three parts in order. There is a macro integration of image features through using the transformer encoder during the learning process.

4.2.3 Model building and performance

There were three kinds of ViT model called ViT-Base, ViT-Large, ViT-Huge. They were mainly different in the complexity of model and parameters' number. We chose the ViT-Large which had better performance than ViT-Base through trail and error. Moreover, ViT-large had a more acceptable training time and less potential of over-fitting compared to ViT-Large. We adopted the VIT-Large model which has 16*16 patch size, 24 times repeated Encoder blocks in Transformer Encoder, 4096 MLP sizes, 16 heads in Multi-Head Attention, and 307M parameters. We used Pytorch to implement VIT-Large in Python. The dropout and DropPath with suitable dropout rates are added in our ViT model to reduce the problem of overfitting. In the Encoder Block, we used DropPath which was faster and more effective [10]. In addition, we added a dropout layer in the Linear Projection of Flattened Patches part after the embedding was formed to prevent the model from overfitting in another way. The layer normalization was also utilized to regularize our model. There are some hyperparameters in our model:

- *Epochs*: max epochs of training the model
- *Batch_size*: the size of batches
- *patch_size*: the size of flattened patches
- *Optimizer*: gradient descent optimization algorithm
- *learningratio*: Initial Learning rate
- *drop_ratio*: dropout ratio in Linear Projection of Flattened Patches part
- *attn_drop_ratio*: dropout ratio in Multi-Headed Attention
- *drop_path_ratio*: DropPath ratio in Transformer Encoder Block

Because Grid Search was hard to implement and its processing times were too long, we chose optimal hyperparameters through trial and error among several combinations of hyperparameters. We chose the optimal epoch with the highest validation set accuracy. Our model utilized transfer learning and initialized the weights as the pre-trained model generated by Google Research on ImageNet-21k dataset[15]. By using the pre-trained model, the converging time could be faster and much training time was saved. The final accuracy of the test set was 0.602. From the confusion matrix, we found that the prediction performance of glioma tumors was also worse than the other three types.

5 Conclusion

5.1 Summary of Results

In a nutshell, our results are summarized as follows, from the highest accuracy to the lowest:

Model	Accuracy	F1-score
Residual Neural Network (ResNet)	80.4%	78.2%
K-Nearest Neighbour (KNN)	79.4%	77.4%
Random Forest (RF)	78.7%	73.9%
Support Vector Machine (SVM)	71.1%	70.8%
Vision Transformer (ViT)	60.2%	57.5%

Table 7: Model performances

5.2 Discussion

In the experiment, we trailed three traditional models (KNN, SVM, RF) and two modern approaches (ResNet, ViT) on the classification of brain tumors.

For the traditional models, KNN and RF have similar accuracies, while SVM failed to perform as well. This might be due to the complexity of data features that is non-separable by hyperplanes. Limited by the high time complexity and the poor run-power of our personal laptops, we have to narrow the optimal parameter range step by step rather than brute force searching all possible combinations (only the final attempt was shown in the code). Hence, our result might not reflect the absolute global optimum. Further improvement could be made by looping more combinations of parameters.

For the modern models, ResNet34 yields the highest accuracy of all attempted models. Possible improvements would be more tuning and including more layers instead of 34. In comparison, ViT performed unexpectedly bad. This might be due to the insufficient training data size for transformer encoding or insufficient time of tunning the hyperparameters. ViT would perform much better in extremely large data set compared to small data set[4].Possible improvements would be increasing the size of training data, or using hybrid ViT models which is more advanced.

6 Code Availability

All code for exploratory data analysis and model building for our project is available at https://github.com/yuhan0205/MH4510-Group-Project.

7 Acknowledgement

We would like to extend our special thanks to an uploader on bilibili (a Chinese website similar to Youtube), whose homepage is https://space.bilibili.com/18161609. We have gained understanding of ResNet and ViT model largely through his video tutorials, and our ResNet and ViT model training, testing and visualization were accomplished by adopting and modifying the various scripts from his github at https://github.com/WZMIAOMIAO/deep-learning-for-image-processing/tree/master/pytorch_classification.

References

[1] Parnian Afshar, Konstantinos N. Plataniotis, and Arash Mohammadi. "Capsule Networks for Brain Tumor Classification Based on MRI Images and Coarse Tumor Boundaries". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 1368–1372. DOI: 10.1109/ICASSP.2019.8683759.

[2] Milica M. Badža and Marko Č. Barjaktarović. "Classification of Brain Tumors from MRI Images Using a Convolutional Neural Network". In: *Applied Sciences* 10.6 (2020). ISSN: 2076-3417. DOI: 10.3390/app10061999. URL: https://www.mdpi.com/2076-3417/10/6/1999.

[3] Sartaj Bhuvaji et al. *Brain Tumor Classification (MRI)*. 2020. DOI: 10.34740/KAGGLE/DSV/1183165. URL: https://www.kaggle.com/dsv/1183165.

[4] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[5] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[6] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[7] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[8] Michael Iv et al. “Current Clinical State of Advanced Magnetic Resonance Imaging for Brain Tumor Diagnosis and Follow Up”. In: *Seminars in roentgenology* 53.1 (Jan. 2018), pp. 45–61. ISSN: 0037-198X. DOI: 10.1053/j.ro.2017.11.005. URL: https://doi.org/10.1053/j.ro.2017.11.005.

[9] Stanford Vision Lab, Stanford University, and Princeton University. *ImageNet*. 2021. URL: https://www.image-net.org/index.php (visited on 11/01/2021).

[10] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. “FractalNet: Ultra-Deep Neural Networks without Residuals”. In: *CoRR* abs/1605.07648 (2016). arXiv: 1605.07648. URL: http://arxiv.org/abs/1605.07648.

[11] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[12] Volodymyr Mnih et al. “Recurrent Models of Visual Attention”. In: *CoRR* abs/1406.6247 (2014). arXiv: 1406.6247. URL: http://arxiv.org/abs/1406.6247.

[13] Antonio Nicolato et al. “Prognostic factors in low-grade supratentorial astrocytomas: A uni-multivariate statistical analysis in 76 surgically treated adult patients”. In: *Surgical Neurology* 44.3 (1995), pp. 208–223. ISSN: 0090-3019. DOI: https://doi.org/10.1016/0090-3019(95)00184-0. URL: https://www.sciencedirect.com/science/article/pii/0090301995001840.

[14] SEER Survival Statistics. *SEER 18 2011–2017*. 2018. URL: https://seer.cancer.gov/statfacts/html/brain.html.

[15] Andreas Steiner et al. “How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers”. In: *CoRR* abs/2106.10270 (2021). arXiv: 2106.10270. URL: https://arxiv.org/abs/2106.10270.

[16] Taiyanghuadexiaolvdou. *Vision Transformer detailed explanation.* 2021. URL: https://blog.csdn.net/qq_37541097/article/details/118242600?spm=1001.2014.3001.5501 (visited on 11/12/2021).

[17] PyTorch Team. *Deep residual networks pre-trained on ImageNet*. URL: https://pytorch.org/hub/pytorch_vision_resnet/ (visited on 10/21/2021).

[18] PyTorch Team. *ImageNet image transformation*. 2020. URL: https://github.com/pytorch/examples/blob/master/imagenet/main.py (visited on 10/21/2021).

[19] PyTorch Team. *ResNet34 pre-trained path*. URL: https://download.pytorch.org/models/resnet34-333f7ec4.pth (visited on 10/21/2021).

[20] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[21] International Agency for Research on Cancer World Health Organizaiton. *The Cancer Survival in High-Income Countries (SURVMARK-2) project*. URL: https://gco.iarc.fr/survival/survmark/.