BRAIN TUMOR DETECTION BASED ON ENSEMBLE LEARNING

DONGHYUN (ETHAN) KIM

Gyeonggi Suwon International School, 451 YeongTong-Ro, YeongTong-Gu, Suwon-Si, Gyeonggi-Do, Republic of Korea

Abstract. In this paper, we propose methods for brain tumor detection in MRI images based on ensemble learning. We build upon prior research on ensemble methods by testing the concatenation of pre-trained models: features extracted via transfer learning are merged and segmented by classification algorithms or a stacked ensemble of those algorithms. The proposed approach achieved accuracy scores of 0.98 , outperforming a benchmark VGG-16 model. Considerations to granular computing are given in the paper as well.

1. Introduction

Brain tumors are defined as the growth of abnormal cells in the human brain. Brain tumors are either benign (non-cancerous) or malignant (cancerous) and are generally classified based on the afflicted region; common brain tumors include meningioma, glioma, and pituitary tumors.

Brain tumors pose a major public health issue as according to the American Cancer Society, the total death count from brain tumors is predicted to be 18,020 in 2020 and 23,890 people are expected to be diagnosed with malignant tumors in 2020.

Treatments for brain tumors include chemotherapy, radiation therapy and surgery. However, before such treatments can begin, initial evaluation of the tumors must take place. Typically, the brain is assessed either by Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) scans.

Though medical images play a crucial role in patient diagnosis and treatment, analyzing such images is a time-consuming and costly task. As a result, computational and mathematical methods have been introduced to this field. Notably, various machine learning methods and techniques such as convolutional neural networks and support vector machines have been utilized to analyze medical images. In this paper, we expand upon past studies by testing the concatenation of pre-trained models.

First, sections 2,3, and 4 will present relevant background information, including an overview of algorithms and ensembles. Next, section 5 will present the proposed approach and sections 6 and 7 will show and discuss its results. Finally, section 8 will conclude the paper.

2. Ensemble Methods

2.1. **Bootstrap Aggregation (Bagging) Ensemble Method ([2]).** The Bootstrap Aggregation method, more commonly known as the bagging method, involves homogeneous weak learners. The learners are trained in parallel and independently from one another and the results are aggregated either by voting or averaging to arrive at a final prediction. Often, the homogeneous learners are base classifiers fitted on random subsets of the dataset.

Given L bootstrap samples, $s_L(.)$ as the model, and $w_1(.), w_2(.), ..., w_i(.)$ as weak learners, the 2 aggregation methods are given by:

$$\text{Averaging: } s_L(.) = \tfrac{1}{L}\textstyle\sum_{l=1}^{L} w_l(.)$$

$$\text{Voting : } s_L(.) = \underset{k}{\operatorname{argmax}}[card(l|w_l(.) = k)]]$$

2.2. Boosting Ensemble Method ([20], [11], [10]).

The Boosting method involves homogeneous weak learners that are trained in a sequential manner. Each subsequent model attempts to correct the errors from the previous model, hence reducing the bias of the overall classifier before a prediction is made. There are 2 main types of boosting methods that differ in how shortcomings of weak learners are identified : adaptive boosting and gradient boosting.

Adaptive boosting identifies such shortcomings by giving higher weight to misclassified input data and lower weight to correctly classified input data. On the other hand, gradient boosting identifies the shortcomings by utilizing gradients to minimize the loss function.

Given $s_l(.)$ as the model, c_l as coefficients, w_l as weak learners, $E(.)$ as the fitting error of the model, and $e()$ as the loss/error function, the following are true for adaptive boosting.

$$s_l(.) = s_{l-1}(.) + c_l \times w_l(.)$$

$$(c_l, w_l(.)) = \underset{c,w(.)}{\operatorname{argmin}}\, E(s_{l-1}(.) + c \times w(.)) = \underset{c,w(.)}{\operatorname{argmin}} \sum_{n=1}^{N} e(y_n, s_{l-1}(x_n) + c \times w(x_n))]$$

Similarly, given ∇ as the gradient, the following is true for gradient boosting.

$$s_l(.) = s_{l-1}(.) - c_l \times \nabla_{s_{l-1}} E(s_{l-1})(.)$$

2.3. Stacking Ensemble Method ([23]).

The Stacking method involves heterogeneous weak learners that are trained in parallel and independently from one another. The results are then aggregated through a meta model that makes a prediction based on the predictions from weak learners.

The meta model for classification tasks is usually logistic regression and for regression tasks, it is usually linear regression. Their respective equations are given below.

$$\text{Linear Regression: } y = a + bx$$
$$\text{Logistic Regression: } p = \tfrac{1}{1+e^{-(a+bx)}}$$

3. Classification Algorithms

Classification algorithms are supervised machine learning techniques that maps given input to categorical target variables.

3.1. **Support Vector Machines ([7]).** A support vector machine is a classifier that constructs a hyperplane in the feature space to separate the data (input) into classes. The classifier aims to maximize the distance between the hyperplane and the nearest data-point of any class.

If the data set is not linearly separable, it can be projected to higher dimensions through a kernel function.

There are 4 main types of kernel functions–linear, polynomial, radial basis function (RBF), and sigmoid–and their equations are given below.

$$\begin{gathered}\text{Linear Kernel Function: } k(X,Y) = (X \cdot Y)\\ \text{Polynomial Kernel Function: } k(X,Y) = (\gamma(X \cdot Y) + C)^d\\ \text{RBF kernel function: } k(X,Y) = -\gamma||X-Y||^2\\ \text{Sigmoid Function: } K(X,Y) = \tanh(\gamma X \cdot Y + C)\end{gathered}$$

(where γ, d, and C are kernel parameters and $||X-Y||^2$ is the square euclidean distance)

The regularization parameter (often referred to as the C parameter) controls the extent to which mis-classification should be avoided. On the other hand, the gamma parameter (γ) defines the extent of influence of a single data point when calculating the hyperplane.

3.2. **K-Nearest Neighbors ([8]).** K-Nearest Neighbors is a classifier that separates the data into classes by examining the distance between data points and the current given point. The algorithm selects k closest data points and classifies the given point based on votes.

There are several distance functions available: Euclidean, Manhattan, and Minkowski.

Euclidean Function:

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan Function:

$$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski Function:

$$(\sum_{i=1}^{k}|x_i - y_i|^q)^{\frac{1}{q}}$$

Furthermore, note that cross-validation is often used to select the value k.

3.3. **Random Forest ([3]).** The Random Forest classifier is a bootstrap aggregation (boosting) ensemble of decision tree classifiers. The decision trees are fitted on random subsets of the dataset and are aggregated either by voting or averaging.

There are a number of heuristics (known as attribute selection measures) that are used to define how data points on certain levels of the tree will be split. 2 heuristics are given below.

$$\text{Information gain (entropy): Info(D)} = -\sum_{i=1}^{n} p_i \log_2(p_i)$$
$$\text{Gini Index: Gini(D)} = 1 - \sum_{i=1}^{n} p_i^2$$

(where p_i is the probability that an arbitrary tuple in D belongs to a class C_i)

3.4. **XGBoost** ([6])**.** XGBoost is an ensemble of gradient boosted decision trees. A key aspect of XGBoost is that it uses more regularized model formalization in order to control overfitting.

4. Convolutional Neural Networks

Convolutional neural networks (CNN) are a class of deep learning neural networks, often used to analyze and classify images. A CNN works by extracting features from an input image (array of pixels). The image passes through a series of layers (usually consisting of convolutional, ReLU, and Pooling layers) before the final fully connected layer classifies the image based on "voting".

4.1. **Convolutional Layer** ([16])**.** A convolutional layer uses filters to extract features from previous layers while preserving corresponding spatial information.

A feature map O_s is calculated as shown below.

$$O_s = b_s + \sum_r W_{sr} * X_r$$

(where b_s is the bias term, W_{sr} is the sub-filter for this feature map, $*$ is the convolution operation, and X_r is the rth inputted feature map)

4.2. **Activation Functions** ([12],[4])**.** Activation functions define the output of neurons given a set of inputs. The function introduces non-linear properties into the network by calculating the 'weighted sum' of inputs before determining which neurons will push forward values into the next layer.

Some common activation functions are given below.

$$\text{Sigmoid: } f(x) = \frac{1}{1+e^{-x}}, f'(x) = f(x)(1-f(x))$$
$$\text{Tanh: } f(x) = \frac{e^x+e^{-x}}{e^x+e^{-x}}, f'(x) = 1 - f(x)^2$$
$$\text{ReLU: f(x)} = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}, \text{ f'(x)} = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$
$$\text{Softmax: } f(x_j) = \frac{e^x j}{\sum_{k=1}^{d} e^x k}, \frac{\partial f(x_j)}{\partial x_i} = f(x_j)(\delta_{ij} - f(x_i))$$

4.3. **Pooling Layers** ([19], [17])**.** Pooling layers reduce the spatial size of activation maps while maintaining important structural elements (without unnecessary detail). The 2 most common methods are max pooling and average pooling and they are given below.

$$\text{Max Pooling: } y_{i^{l+1},j^{l+1},d} = \max_{\{0\leq i<H, 0\leq j<W\}} x^l_{i^{l+1}} \times H + i, j^{l+1} \times W + j, d$$

$$\text{Average Pooling: } y_{i^{l+1},j^{l+1},d} = \frac{1}{HW} \sum_{\{0\leq i<H, 0\leq j<W\}} x^l_{i^{l+1}} \times H + i, j^{l+1} \times W + j, d$$

(where x is the input to the layer, y is the output of the l-th layer, and $H^l \times W^l \times D^l$ is the size of the l-th layer)

4.4. **Transfer Learning.** In the field of deep learning, transfer learning is a technique in which pre-trained models are selected and applied on a different but related problem. This technique is often used because these models have already been trained on huge data-sets like imageNet.

4.4.1. *VGG-16 (*[21]*).* VGG-16 is a 16 layer convolutional neural network model introduced by K. Simonyan and A. Zisserman in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model consists of several convolutional and max-pooling layers and the ReLU activation function. VGG-16 improved upon prior models by replacing large kernel-sized filters (11 in the first convolutional layer and 5 in the second convolutional layer) with multiple 3×3 kernel-sized filters in a serialized fashion. This model achieved 92.7% accuracy on the ImageNet dataset (a dataset containing over 14 million images and 1000 classes).

4.4.2. *Resnet-50 (*[15]*).* Resnet-50 is a 50 layer residual neural network model introduced by Kaiming He et al at Microsoft Research. The model consists of shortcut (skip) connections that contain nonlinearities (ReLU) and batch normalization. As a result of shortcut connections, the model has lower complexity when compared to VGG models.

4.4.3. *Inception-v3 (*[22]*).* Inception-v3 is a 48 layer convolutional neural network model introduced by the Google Brain Team. The model consists of batch normalization, image distortions, RMSprop and utilizes numerous small convolutions to greatly decrease the number of parameters.

4.5. **Granular Computing (**[18]**).** The model proposed in this paper is an application of the granular computing paradigm. Granular computing is concerned with the processing of information units or information granules. These information granules are collections of entities that are arranged together based on similar aspects. In the context of machine learning and ensemble learners, each ensemble can be thought of as a granule because they combine multiple learning algorithms. The proposed model and ensemble consists of multiple levels, where each level corresponds to a distinct level of granularity. For example, in a stacked ensemble, the collection of base level classifiers is at the bottom level of granularity and the entire ensemble (including both base level classifiers and the meta classifier) is at the highest level of granularity.

Two granular computing concepts–granulation and organization–can be applied to the proposed model as well. Granulation involves the process of decomposing an object into parts and organization involves the process of integrating certain parts into a complete object. The extraction of feature vectors with convolutional neural networks is essentially granulation, and combining different classifiers and learners through a stacking ensemble is organization.

Granular computing concepts can be applied to the random forest classifier as well. The bootstrap aggregation technique (bagging ensemble) found in a random forest classifier involves integrating numerous decision trees into 1 ensemble; it involves the organization concept.

5. Proposed Method

5.1. **Dataset.** The proposed method was tested on the "Brain MRI Images for Brain Tumor Detection dataset", which contains a total of 253 images: 155 of the images contain tumors and 98 do not. Before the images were used to train and test the model, the dataset was pre-processed and augmented.

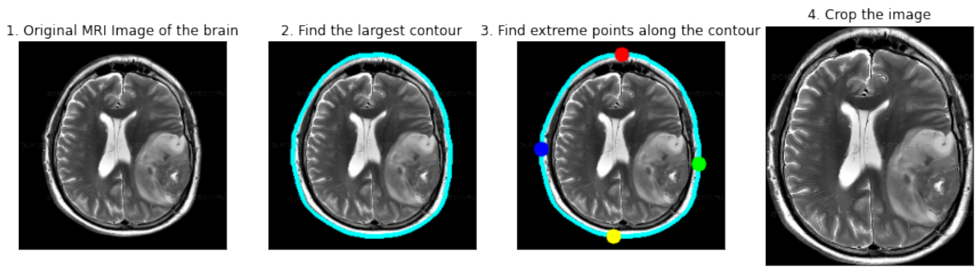
The dataset was created by Navoneel Chakrabarty and it can be found here: `https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection/data`([5]),

5.2. **Data Pre-processing.** Data Pre-processing involved 3 main steps: splitting the dataset, cropping the images, and resizing the images.

5.2.1. *Dataset Splitting.* The dataset was split up into 3 distinct sets: the training set, validation set, and test set. The training set is used to train and fit the model, the validation set provides both an unbiased evaluation of the trained model and tunes the model's hyper-parameters, and the test set provides an unbiased evaluation of the final model. The ratio used to split the training, validation, and test set was 60/20/20 respectively, resulting in 152 images in the training set, 50 images in the validation set, and 51 images in the test set.

5.2.2. *Cropping.* The brain was cropped out of the MRI images through a 3 step method. The method finds the largest contour of the brain, finds extreme points along the contour, and crops the image. An example is given below in Figure 1.

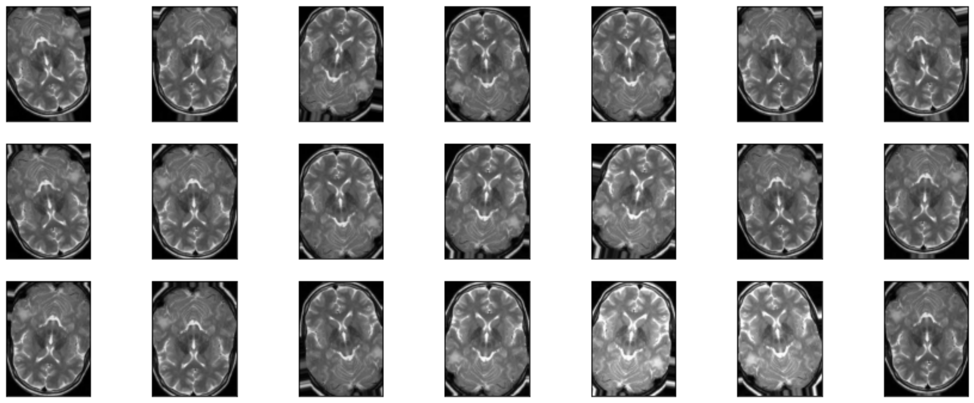
Figure 1. Cropping


1. Original MRI Image of the brain
2. Find the largest contour
3. Find extreme points along the contour
4. Crop the image

5.2.3. *Resizing and additional pre-processing.* Because the dataset contains images of differing dimensions and aspect ratios, they were resized to match the input size of the pre-trained models (dimensions $224 \times 224 \times 3$). Additional pre-processing (including interpolation and subtracting the mean RGB channels of the data-set) was done to prepare the images for the pre-trained model as well.

5.3. **Data Augmentation.** To increase the size of the datasets, the data was augmented through numerous random transformations. Through data augmentation, possibilities of over-fitting were reduced and the model was able to generalize better. Chosen augmentation options include a range of rotation of 15 degrees, width and height translation range of 0.1, shearing transformation range of 0.1, brightness range between 0.5 and 1.5, and horizontal and vertical flips. An example of data augmentation is given below in Figure 2.

FIGURE 2. Augmented Images



5.4. **Architecture.** The proposed model consists of 2 distinct levels. First, modified pre-trained convolutional neural network models are used to extract features from the pre-processed MRI images. The extracted feature tensors are then merged and segmented either by classification algorithms or a combination of those algorithms (ensemble methods).

5.4.1. *Modified Pre-trained convolutional neural network models.* 3 different pre-trained convolutional neural network models are used in the model: VGG-16, Inception-v3, and Resnet-50.

Because segmentation is done by separate classifiers, the final fully connected layers of the pre-trained models are not included. Instead, a custom flatten layer, dropout layer, and dense layer with the sigmoid activation function are added at the end of the models to prepare the tensors for the merging step.

5.4.2. *Merging extracted features.* The feature vectors extracted from the pre-trained convolutional neural network models are merged via concatenation. Concatenation was chosen instead of other merging options (eg. average, add) to ensure that none of the input was discarded. The feature vectors are merged to utilize the distinct features extracted from different models and to improve classification.

Once the features vectors have been merged, the resulting vector passes through another dense layer with the sigmoid activation function.

The merging of pre-trained models takes place with the following combinations: VGG16 + Resnet-50, VGG16 + Inception-v3, Resnet-50 + Inception-v3, and VGG16 + Resnet-50 + Inception-v3.

5.4.3. *Classification Algorithms.* The classification algorithms that are used in the model include support vector machines, k-nearest neighbors, random forest classifiers, XGBOOST, and a fully connected dense layer (with the sigmoid activation function). Before classification takes place, the hyper parameters of all these algorithms (including choice of kernel for support vector machines) are selected using grid search with cross-validation.

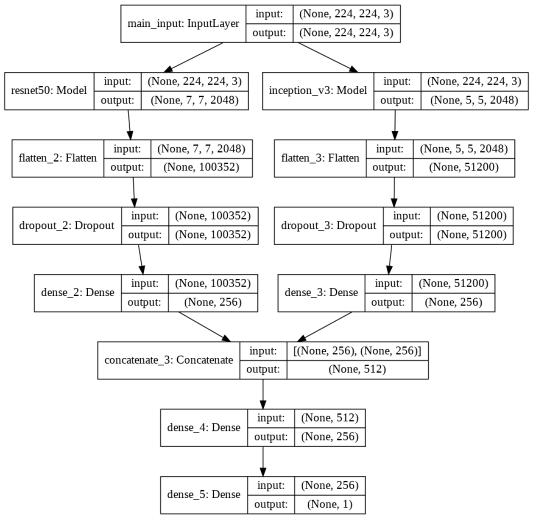
doi:10.20944/preprints202008.0641.v2

5.4.4. *Ensemble.* The ensemble method used to combine the heterogeneous collection of classification algorithms is model stacking. Through model stacking, the predictions of various models can be combined to potentially obtain better predictive performance.

The structure of the proposed model with Resnet-50 and Inception-v3 as the feature extractors and a fully connected dense layer (with the sigmoid activation function) as the classifier is shown below in Figure 3.

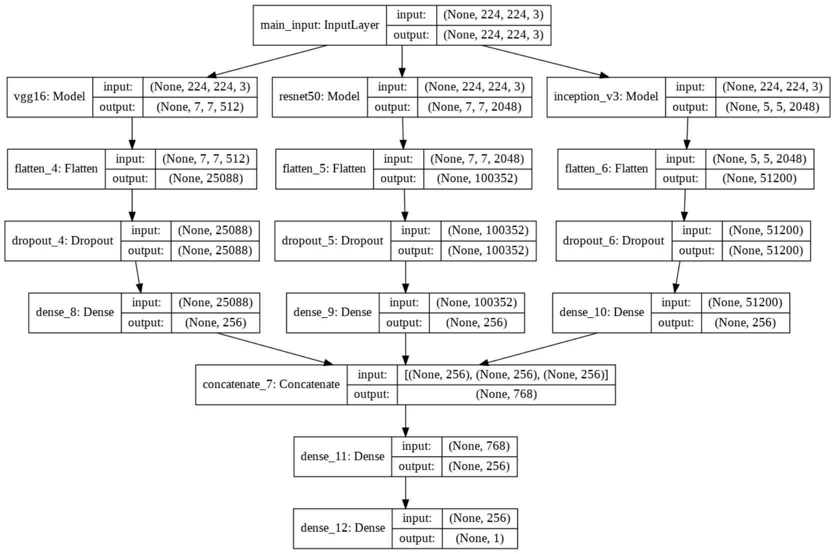
For other variations of the model, different combinations of feature extractors (pre-trained models) are used and the final dense layer is simply switched with a classification algorithm or a stacked ensemble of classifiers.

FIGURE 3. Proposed model with Resnet-50 and Inception-v3 as the feature extractors and a fully connected dense layer (with the sigmoid activation function) as the classifier


main_input: InputLayer | input: (None, 224, 224, 3) | output: (None, 224, 224, 3)
resnet50: Model | input: (None, 224, 224, 3) | output: (None, 7, 7, 2048)
inception_v3: Model | input: (None, 224, 224, 3) | output: (None, 5, 5, 2048)
flatten_2: Flatten | input: (None, 7, 7, 2048) | output: (None, 100352)
flatten_3: Flatten | input: (None, 5, 5, 2048) | output: (None, 51200)
dropout_2: Dropout | input: (None, 100352) | output: (None, 100352)
dropout_3: Dropout | input: (None, 51200) | output: (None, 51200)
dense_2: Dense | input: (None, 100352) | output: (None, 256)
dense_3: Dense | input: (None, 51200) | output: (None, 256)
concatenate_3: Concatenate | input: [(None, 256), (None, 256)] | output: (None, 512)
dense_4: Dense | input: (None, 512) | output: (None, 256)
dense_5: Dense | input: (None, 256) | output: (None, 1)

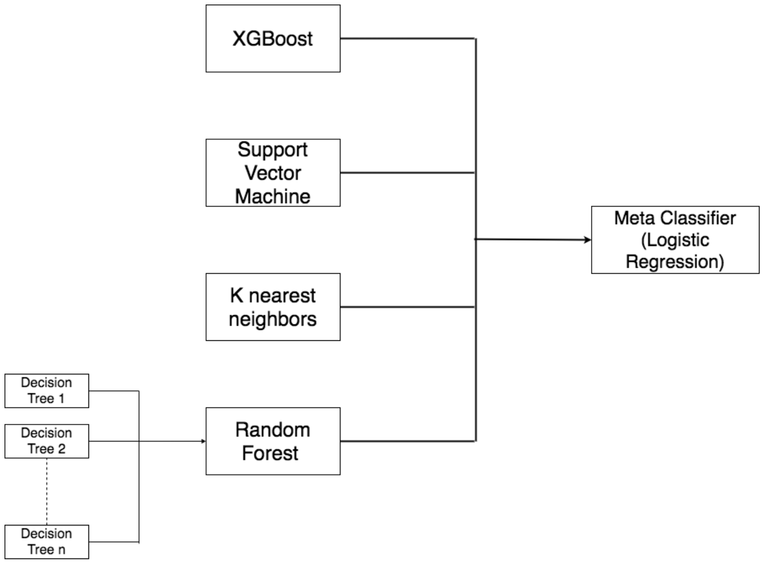
A similar model with Resnet-50, Inception-v3 and VGG-16 as the feature extractors and a fully connected dense layer (with the sigmoid activation function) as the classifier is shown below in Figure 4.

FIGURE 4. Proposed model with Resnet-50, Inception-v3, and VGG-16 as the feature extractors and a fully connected dense layer (with the sigmoid activation function) as the classifier


main_input: InputLayer | input: (None, 224, 224, 3) | output: (None, 224, 224, 3)
vgg16: Model | input: (None, 224, 224, 3) | output: (None, 7, 7, 512)
resnet50: Model | input: (None, 224, 224, 3) | output: (None, 7, 7, 2048)
inception_v3: Model | input: (None, 224, 224, 3) | output: (None, 5, 5, 2048)
flatten_4: Flatten | input: (None, 7, 7, 512) | output: (None, 25088)
flatten_5: Flatten | input: (None, 7, 7, 2048) | output: (None, 100352)
flatten_6: Flatten | input: (None, 5, 5, 2048) | output: (None, 51200)
dropout_4: Dropout | input: (None, 25088) | output: (None, 25088)
dropout_5: Dropout | input: (None, 100352) | output: (None, 100352)
dropout_6: Dropout | input: (None, 51200) | output: (None, 51200)
dense_8: Dense | input: (None, 25088) | output: (None, 256)
dense_9: Dense | input: (None, 100352) | output: (None, 256)
dense_10: Dense | input: (None, 51200) | output: (None, 256)
concatenate_7: Concatenate | input: [(None, 256), (None, 256), (None, 256)] | output: (None, 768)
dense_11: Dense | input: (None, 768) | output: (None, 256)
dense_12: Dense | input: (None, 256) | output: (None, 1)

The structure of an ensemble classifier stacking a XGBoost classifier, a support vector machine, k nearest neighbors classifier, and a random forest classifier is shown below in Figure 5.

FIGURE 5. Stacked Ensemble combining XGBoost, Support Vector Machines, K nearest neighbors, and Random Forest.



6. RESULTS

All experiments were conducted via Google Colaboratory and the results are presented in this section.

Early stopping was used when training the models to avoid overfitting and selected parameters include monitoring for validation accuracy in 'max' mode with a minimum of 10 epochs.

Note that a GPU was used during model training.

A benchmark VGG-16 model will be tested to compare results as well. The benchmark VGG-16 model consists of the pre-trained VGG-16 model without the final fully connected layers. Instead, a dropout layer, flatten layer, another dropout layer, and a dense layer with the sigmoid activation function are added in that order.

Training Accuracy	Validation Accuracy	Test Accuracy
0.993	0.88	0.843

TABLE 1. Table of Model Accuracy for the Benchmark VGG-16 model

Classifier	Training Accuracy	Validation Accuracy	Test Accuracy
Sigmoid	0.947	0.9	0.922
SVM	0.987	0.88	0.961
KNN	0.980	0.86	0.961
XGBOOST	1.0	0.84	0.941
Random Forest	0.993	0.84	0.941
SVM + KNN	0.987	0.86	0.961
SVM + XGBOOST	0.993	0.84	0.941
SVM + Random Forest	0.993	0.86	0.941
KNN + XGBOST	1.0	0.84	0.961
KNN + Random Forest	0.987	0.84	0.961
XGBOOST + Random Forest	1.0	0.84	0.941
SVM + KNN + XGBOOST	0.993	0.86	0.961
SVM + KNN + Random Forest	0.993	0.86	0.961
SVM + XGBOOST + Random Forest	0.993	0.84	0.941
KNN + XGBOOST + Random Forest	0.993	0.84	0.961
SVM + KNN + XGBOOST + Random Forest	0.993	0.84	0.961

TABLE 2. Table of Model Accuracy with Resnet-50 and Inception-v3 as the feature extractors

Classifier	Training Accuracy	Validation Accuracy	Test Accuracy
Sigmoid	0.974	0.88	0.863
SVM	1.0	0.9	0.98
KNN	0.987	0.9	0.961
XGBOOST	1.0	0.88	0.922
Random Forest	1.0	0.9	0.98
SVM + KNN	1.0	0.9	0.98
SVM + XGBOOST	1.0	0.9	0.98
SVM + Random Forest	1.0	0.9	0.98
KNN + XGBOST	1.0	0.88	0.922
KNN + Random Forest	0.993	0.9	0.98
XGBOOST + Random Forest	1.0	0.88	0.922
SVM + KNN + XGBOOST	1.0	0.9	0.98
SVM + KNN + Random Forest	1.0	0.9	0.98
SVM + XGBOOST + Random Forest	1.0	0.9	0.98
KNN + XGBOOST + Random Forest	1.0	0.9	0.941
SVM + KNN + XGBOOST + Random Forest	1.0	0.9	0.98

TABLE 3. Table of Model Accuracy with Resnet-50 and VGG-16 as the feature extractors

Classifier	Training Accuracy	Validation Accuracy	Test Accuracy
Sigmoid	0.98	0.9	0.824
SVM	0.974	0.92	0.843
KNN	0.98	0.88	0.804
XGBOOST	1.0	0.9	0.882
Random Forest	1.0	0.9	0.863
SVM + KNN	0.98	0.88	0.804
SVM + XGBOOST	0.993	0.9	0.882
SVM + Random Forest	0.993	0.9	0.863
KNN + XGBOST	0.993	0.9	0.863
KNN + Random Forest	0.993	0.88	0.824
XGBOOST + Random Forest	1.0	0.9	0.863
SVM + KNN + XGBOOST	0.993	0.9	0.863
SVM + KNN + Random Forest	0.98	0.88	0.804
SVM + XGBOOST + Random Forest	0.993	0.9	0.863
KNN + XGBOOST + Random Forest	1.0	0.9	0.863
SVM + KNN + XGBOOST + Random Forest	0.993	0.9	0.863

TABLE 4. Table of Model Accuracy with VGG-16 and Inception-v3 as the feature extractors

Classifier	Training Accuracy	Validation Accuracy	Test Accuracy
Sigmoid	0.987	0.9	0.863
SVM	0.993	0.86	0.882
KNN	0.993	0.9	0.882
XGBOOST	1.0	0.86	0.902
Random Forest	1.0	0.86	0.863
SVM + KNN	0.993	0.9	0.882
SVM + XGBOOST	1.0	0.86	0.902
SVM + Random Forest	1.0	0.88	0.863
KNN + XGBOST	1.0	0.86	0.902
KNN + Random Forest	0.993	0.86	0.863
XGBOOST + Random Forest	1.0	0.86	0.902
SVM + KNN + XGBOOST	1.0	0.86	0.882
SVM + KNN + Random Forest	0.993	0.86	0.863
SVM + XGBOOST + Random Forest	1.0	0.86	0.882
KNN + XGBOOST + Random Forest	1.0	0.86	0.882
SVM + KNN + XGBOOST + Random Forest	1.0	0.86	0.882

TABLE 5. Table of Model Accuracy with Resnet-50, Inception-v3, and VGG-16 as the feature extractors

7. Discussion

From the results, we see that the proposed models achieved high accuracy scores in segmenting the MRI images. Though exceptions did exist, accuracy greater than 0.85 for the test data-set was achieved for all combinations of feature extractors. In particular, the model that combined VGG-16 and Resnet-50 achieved test accuracy of 0.98 for most of its classifiers/ensembles.

Moreover, most of the proposed models achieved higher accuracy scores when compared to the benchmark VGG-16 model which achieved an accuracy rate of 0.843 for the test dataset.

8. Conclusion

In this paper, methods based on ensemble learning were proposed for detecting brain tumors in MRI images. The proposed models were successful in the given task, reaching scores of 0.98 in the case of the stacked ensemble model combining VGG-16 and Resnet-50. Though exceptions did exist, most of the proposed models in this paper outperformed the benchmark VGG-16 model.

The author hopes that research into merging model outputs and stacking ensembles continues, especially in the context of medical research.

Python programs used for data pre-processing and experimentation can be found in this Github repository: `https://github.com/ethank11k/Brain-Tumor-Detection-Models/`.

References

1. Braverman, M. (2011). Poly-logarithmic independence fools bounded-depth boolean circuits. *Communications of the ACM, 54*(4), 108-115.
2. Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2), 123-140.
3. Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5-32.
4. Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing* (pp. 227-236). Springer, Berlin, Heidelberg.
5. Chakrabarty, Navoneel. (2019). *Brain MRI Images for Brain Tumor Detection.* Retrieved from Kaggle: https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection
6. Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. *In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).
7. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3), 273-297.
8. Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1), 21-27.
9. Delalleau, O., & Bengio, Y. (2011). Shallow vs. deep sum-product networks. In *Advances in neural information processing systems* (pp. 666-674).
10. Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780), 1612.
11. Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.
12. Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., & Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789), 947-951.
13. Hastad, J. (1986, November). Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing* (pp. 6-20).
14. Hastad, J., & Goldmann, M. (1991). On the power of small-depth threshold circuits. *Computational Complexity*, 1(2), 113-129.

15. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
16. LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. *In Advances in neural information processing systems* (pp. 396-404).
17. Lin, M., Chen, Q., & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400.*
18. Pedrycz, W., Skowron, A., & Kreinovich, V. (Eds.). (2008). *Handbook of granular computing.* John Wiley & Sons.
19. Ranzato, M. A., Huang, F. J., Boureau, Y. L., & LeCun, Y. (2007, June). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In 2007 IEEE conference on computer vision and pattern recognition (pp. 1-8). IEEE.
20. Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, 5(2), 197-227.
21. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv*:1409.1556.
22. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818-2826).
23. Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2), 241-259.
24. Yao, A. C. C. (1985, October). Separating the polynomial-time hierarchy by oracles. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)* (pp. 1-10). IEEE.

Gyeonggi Suwon International School, 451 YeongTong-Ro, YeongTong-Gu, Suwon-Si, Gyeonggi-Do, Republic of Korea
Email address: ethank11k@gmail.com