Brain Tumor Classification

Beth Crisler
Natalie Elenz
Myra Tran
Patrick Xiao
Group 2 - Distance Section

Abstract

Brain tumor classification is of the utmost importance in modern cancer research and patient care. Oftentimes, the speed of which diagnosing a cancerous tumor is a matter of life and death for a patient. Cancerous brain tumors have a high mortality rate, and account for a large number of nervous system tumors. The faster a patient can obtain that diagnosis, the faster they can seek treatment that may ultimately save their life. The following paper is meant to outline our attempt at creating a convolutional neural network (CNN) that will accurately classify MRI images of brain tumors. This will give healthcare providers a tool that will quicken the diagnosis process for multiple patients at a time, instead of reviewing the images individually.

We developed a multi-classification CNN that takes JPG images of MRI brain scans, and produces labels identifying the types of brain tumors in those images. We started with an open sourced code with binary classification, and modified it to handle multiclass classification.

Potential near-term impact of this model includes improved diagnostic accuracy, increased efficiency, enhanced accessibility, and standardization of diagnosis. Longer term impact can potentially include: early intervention, personalized treatment planning, advanced research and development, and reduced healthcare costs. This model could also potentially provide a positive impact to data collection on brain tumors, more transparency between doctors and patients, and ideally integration into healthcare systems.

1. Introduction

In Siegel, Giaquinto, and Jemal's publication for the American Cancer Society, they estimated that for both male and female patients in the United States in 2024, there would be 25,400 new cases of cancer in the brain and other parts of the nervous system (14,420 in males and 10,980 in females). Also for the year 2024, they estimated that the number of deaths caused by brain and nervous system cancers would reach 18,760 for both sexes (10,690 males and 8,070 females). These statistics alone show that finding fast and accurate ways of diagnosing brain tumors is imperative to the future of our healthcare systems.

According to the American Cancer Society's article, *Key Statistics for Brain and Spinal Cord Tumors*, malignant tumors are more likely to develop in male patients than in females, however benign tumors develop more often in female patients than in males. Though there is less than a 1% chance that a patient will develop a malignant brain or spinal cord tumor, early diagnosis is still key in recovery and survival.

Magnetic Resonance Imaging (MRI) has been an incredibly useful tool in diagnosing brain tumors. Doctors are able to see whether they are malignant, based on whether there is dead tissue around the tumor, and they can see what the best course of treatment would be. Most commonly, MRIs would be reviewed by a radiologist who would classify the tumors to the best of their abilities. This task is rather time consuming and requires a skilled and experienced radiologist. This process is also liable to human error where it is possible to misconstrue a malignant tumor for a benign one, and vice versa.

In our research for this report, we seek to build a convolutional neural network that is capable of applying diagnosis labels to JPG images of MRI brain scans. Our multi-classification system identifies the images as four different labels: Pituitary Tumor, Meningioma Tumor, Glioma Tumor, and No Tumor. We have based our CNN code on an open sourced code that used binary classification originally. We thought, with how nuanced brain tumors can be, that multi-classification would be a better avenue, and we hoped to improve our model to handle labeling images in this way.

2. Related Work

According to ZainElden et al., the diagnosis of a brain tumor is a lengthy and expensive endeavor. It requires a radiologist who is highly skilled and experienced, and it takes time to correctly classify both benign and malignant tumors. Older methods of diagnosis are becoming

obsolete, as tumors become more prevalent. This makes deep learning ideally a faster and more accurate method of classification and diagnosis. In ZainElden et al.'s publication, *Brain Tumor Detection and Classification Using Deep Learning and Sine-Cosine Fitness Grey Wolf Optimization*, they sought to research and improve a brain tumor classification model (BCM) using a convolutional neural network (CNN) and optimizing their hyperparameters with an adaptive dynamic sine-cosine fitness grey wolf optimizer (ADSCFGWO) algorithm. Our research in this project is similar, in that we want to further the research concerning BCM-CNN's, and improve upon work that has already been completed in medical image classification.

In another related work, *Brain Tumor Detection and Classification Using Convolutional Neural Network and Deep Neural Network*, by Choudhury et al., a convolutional neural network was used to determine whether a brain tumor was malignant or benign, using binary classification. Their model consisted of three 2D convolutional layers, which took in image inputs with the shape defined as 366, 310, and 1. This is related to the image's width-height-grayscale nature. They also used very little pre-processing of their input images, and managed to achieve 96.08% accuracy with only 35 epochs. Though we pre-process our images quite a bit more than Choudhury et al., we also have three 2D convolutional layers. The difference is that we implemented transfer learning with a Visual Geometry Group (VGG) CNN, a pre-trained CNN that we fine-tuned on the brain image data set. Building on a pre-trained CNN model improved our accuracy overall.

3. Data

Our data set is a set of brain MRI images, sourced from Kaggle.com's library, uploaded by Kaggle user Sartaj. This data set has already been separated into four different labels: Pituitary Tumor, Meningioma Tumor, Glioma Tumor, and No Tumor. We have also separated them into Training and Testing sets for the purpose of our experiments.

4. Methods

4.1 Data Splitting

Downloading the required libraries was our first step. These libraries included 'os' for file operations, 'numpy' for numerical calculations, 'cv2' for image processing, and 'train_test_split' from 'sklearn.model_selection' for dataset splitting. The 'load_images' method was then defined after that. Every folder in our dataset directory is iterated over by this program. It then detects whether the item is in a directory for each folder, and if it is, it uses the folder name to produce the label for the photographs. The label 'no_tumor' is assigned if the folder name begins with 'no'; if not, the tumor type is extracted from the folder name. It loaded each image in the folder using 'cv2.imread' and then converted each to RGB format. Next, it was downsized to 224 by 224 pixels and appended the image and corresponding label to lists. We set it up so that if loading or processing an image encountered an error, it would print an error message. If the item was not in our directory, it would also print an error message.

After running our code, it returned the loaded images and their corresponding labels as 'numpy' arrays.

The paths to the training and testing dataset directories ('training_path' and 'testing_path') were specified using 'os.path.join'. We called the 'load_images' function to load images and labels for both the training and the testing datasets. The loaded images and labels were stored in the 'training_images', 'training_labels', 'testing_images', and 'testing_labels' variables. Then the 'train_test_split' function was used to split the training dataset into training and validation sets ('X_train', 'X_val', 'y_train', 'y_val'). The validation set size was set to 20% of the training dataset, and a random seed of 42 was used so we could reproduce it later if needed.

4.2 Data Visualization

Next we wanted to visualize the data we were using. First we imported 'matplotlib.pyplot' (as 'plt') for data visualization, and 'seaborn' (as 'sns') for enhanced data visualization. Our first step in exploring the dataset was to print out the shapes of the training and testing datasets. This information helped us understand the size and structure of the dataset, which helped with building our model and performing evaluation. The shape of the training images was (2296, 224, 224, 3), and the shape of the training labels was (2296,). The shape of the validation images was (574, 224, 224, 3), and the labels shape was (574,).

Next we wanted to visualize the class distribution. We used the 'sns.countplot' function to create a bar plot showing the count of each unique class within the 'y_train' array. The 'y_train' array contained the labels corresponding to the training images. Each bar in the plot represents the frequency of a particular class within the training dataset. This provided a visual representation of the class distribution within the training data for us, allowing for a quick assessment of class balance or imbalance. This helped us understand the dataset's characteristics and potential biases. [Fig. 1]

Figure 1. Class Distribution in Training Data

We also wanted to see a subset of sample images from the training dataset along with their corresponding labels. We set each figure size to 10x10, and set up our loop to produce 9 random sample images from our training dataset along with its corresponding label. The images were arranged in a 3x3 grid, with each subplot showing one sample image. [Fig. 2]

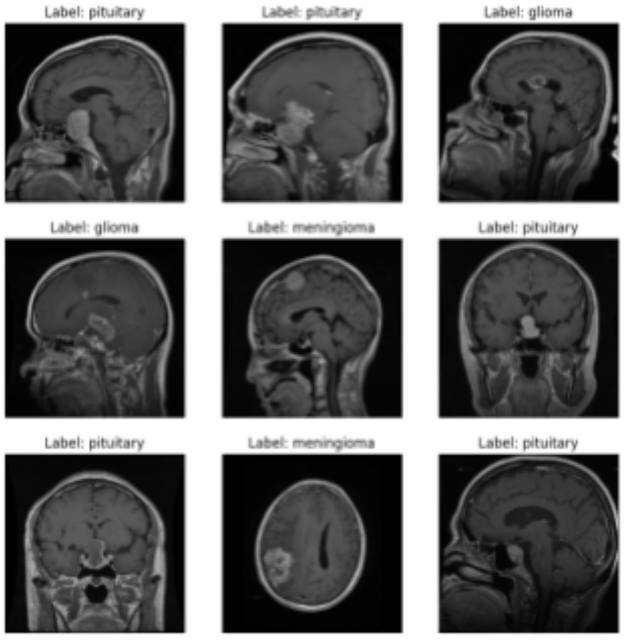

Figure 2: Sample Images with Corresponding Labels

We also wanted to explore the image properties, so we set up code that would print out each image's shape and its pixel intensity range. The image shape that was printed read as (224, 224, 3), and the pixel intensity range was 0 255. This meant that each image had a resolution of 224 pixels in height and 224 pixels in width. The '3' at the end of the shape reading indicated to us that the image is in color (RGB), with three channels corresponding to red, green, and blue. The pixel intensity values in the image range from 0 to 255. This is standard for images represented in an 8-bit format, where each channel (red, green, blue) can have intensity values ranging from 0 (black) to 255 (fully saturated color).

4.3 Data Preprocessing

Our next step was to normalize the pixel values so we can fully analyze them with our convolutional neural network. First we normalized the pixel values of the training images ('X_train') by converting them to floating-point format ('float32') and then dividing each pixel value by 255.0. We did the same to the validation images ('X_val').

Data normalization is a preprocessing technique commonly applied to input data before feeding it into machine learning models. In the context of image data, normalization typically involves scaling the pixel values to a range between 0 and 1. Dividing each pixel value by 255.0 achieves this normalization because the original pixel values range from 0 to 255 (with our dataset images being 8-bit per channel color), and dividing by 255.0 scales them to the range [0, 1]. Normalizing the data helps in stabilizing the training process and improving convergence speed, especially for optimization algorithms like gradient descent. It also helps in ensuring that all input features contribute equally to the learning process, preventing features with larger scales from dominating the learning process.

Before we started to build our model, we also wanted to analyze any relationships or correlations. We first calculated the mean pixel intensity for images belonging to class 0 (or the first class) and class 1 (the second class). We stored the calculations respectively in 'mean_intensity_class0', and 'mean_intensity_class1'.

This was in order to investigate whether there are any discernible differences in the mean pixel intensity values between different classes in our dataset. By calculating the mean pixel intensity separately for each class, it helped us understand potential differences or similarities in the visual characteristics of images across our different classes. The mean pixel intensity also provided insights into the overall brightness or darkness of images within each of our classes. Differences in mean pixel intensity could indicate variations in image composition, lighting conditions, or object characteristics between different classes. Analyzing correlations between pixel intensity and class labels is valuable for feature selection, model interpretation, and understanding the dataset's underlying structure.

4.4 Model Building

We started building our model by creating a baseline accuracy metric using random predictions. We imported 'accuracy_score' from the 'sklearn.metrics' library, and then from 'sklearn.preprocessing' we imported 'LabelEncoder'. The 'LabelEncoder' class from 'scikit-learn' is used to encode categorical labels into integer labels. We fitted the label encoder to our training labels and transformed them into integer labels. We then counted the occurrences of each class label in the encoded training labels and generated random predictions for each sample based on the class distribution. We calculated the accuracy of the random predictions by comparing the true encoded training labels ('training_labels_encoded') with the randomly generated predictions ('random_predictions'). Our final baseline accuracy was 0.2763066202090592.

Next, we provided a pipeline for loading, preprocessing, visualizing, and splitting image data for brain tumor classification tasks. To do this, we defined a function called 'get_data'. This function is responsible for loading and preprocessing images from our dataset. It takes two optional arguments: 1) 'test', which is a boolean flag indicating whether to load images from the testing directory with the default set to 'False', and 2) 'visualize' which is a boolean flag indicating whether to visualize the raw data with the default also set to 'False'. We also set a random seed to ensure reproducibility of results when generating random numbers.

Paths were set up to different subdirectories within the dataset directory based on the tumor types: Glioma, Meningioma, No Tumor, and Pituitary. Our code reads image files from each tumor type directory, resizes them to a common shape of (128, 128), and stores them in their own separate lists.

When the test argument was set to 'False' (indicating training data), our code flattened the images and prepared labels for the classification task. Each image was flattened into a 1D array and tumor images were labeled as 1, while images with no tumor were labeled as 0. Flattened images and labels were combined into feature (X) and target (Y) arrays. Our data was then split into training and validation sets using 'train_test_split'.

When the visualize argument was set to 'True', our code visualized the raw data by displaying a grid of images from each tumor type using 'matplotlib'. Each row in the grid represented a tumor type, and each column displayed a sample image. [Fig. 3]

Next we chose a pretrained model to set as our base model. We chose Visual Geometry Group with 16 layers (VGG16) with the arguments as: 'weights = imagenet', 'include_top = False', and 'input_shape = (244, 244, 3)'.

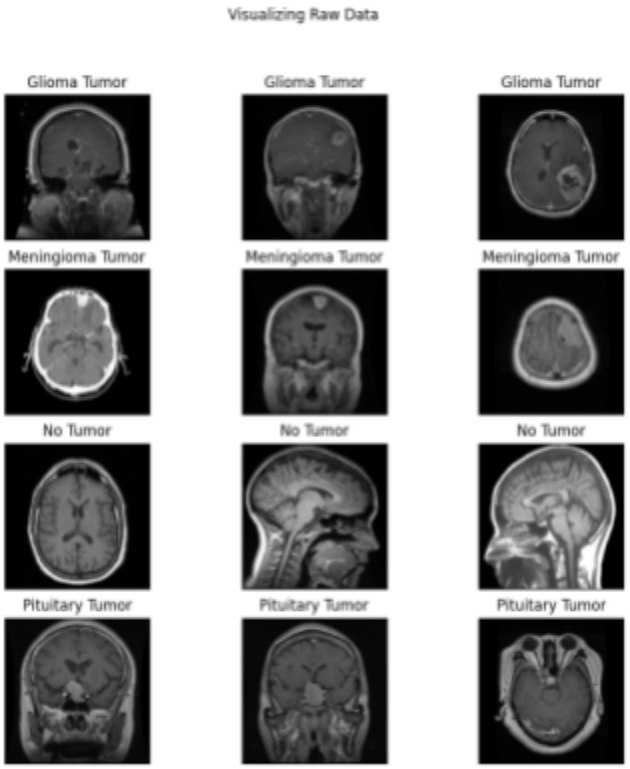

Figure 3: Visualization of Raw Data

We also imported 'tensorflow' and 'keras.layers' to use in building our CNN. 'LabelEncoder' was used to set up our integer labels for our training and validation datasets. The number of classes was determined by the length of unique classes obtained from label encoding. We also used 'to_categorical' to convert the integer labels into a one-hot encoded format for both training and validation sets.

We created the function 'create_multiclass_cnn_model' to define the architecture of our CNN model. Our model consisted of several different layers: 1) Rescaling, which normalized the input pixel values to the range [0,1]. 2) Convolutional layers (Conv2D) which had an increasing number of filters (16, 32, 64) and a kernel size of 3x3. 3) MaxPooling layers (MaxPooling2D) were added to down-sample the feature maps. 4) A flatten layer was used to convert 2D feature maps into a 1D feature vector. 5) Fully connected (Dense) layers, which brought in the ReLU activation. And 6) the output layer with softmax activation which was used for multiclass classification.

Our model was then compiled with the 'adam' optimizer, 'categorical_crossentropy' loss function (suitable for multiclass classification), and 'accuracy' metric. We then trained our model on the training data with the one-hot encoded labels we set up previously. Our validation data was also used with the one-hot encoded labels. We had a total batch size of 64, and ran through 15

epochs, where in the last epoch our training accuracy was 1.00, and our validation accuracy was 0.9059.

5. Empirical Applications, Experiments, and Results

5.1 Transfer Learning

Our first experiment with improving our model was to try transfer learning. We imported the VGG16 model from 'tensorflow.keras.applications' again. VGG16 is a pre-trained convolutional neural network architecture commonly used for image classification, so we hoped it would improve our model. We loaded VGG16 with pre-trained weights from the 'imagenet' dataset. We used the 'include_top=False' parameter so we could specify not to include the fully connected layers on top of the network, as we added our own. And we set the 'input_shape' parameter to (128, 128, 3), to indicate that the image sizes should be 128x128 pixels with 3 color channels (RGB).

The pre-trained layers of VGG16 were set to non-trainable ('layer.trainable = False'). This prevented their weights from being updated during training. We froze these layers in order to retain the knowledge learned from the original task with 'imagenet'. This way we could prevent them from being overwritten with new data.

We created a new 'Sequential' model to stack layers. The pre-trained VGG16 model ('vgg_base') was added as the first layer. We then added the 'flatten' layer to level the output from the convolutional base into a 1D vector. The two 'dense' layers were then added on top for custom classification. The first 'dense' layer was added with 64 units and ReLU activation function. And the output 'dense' layer was added with the units being the number of classes, and the softmax activation function to enable multiclass classification. Lastly, the model was compiled with the 'adam' optimizer, 'categorical_crossentropy' loss function (suitable for multiclass classification), and 'accuracy' metric, like our original model.

5.2 Data Augmentation

Our second experiment was using data augmentation. We used our previous model with the VGG16 transfer learning, and compiled it in the same way as with our original model.

We performed our data augmentation using Keras' 'ImageDataGenerator'. Augmentation techniques include rotation, width and height shifting, shear, zoom, and horizontal flipping. We set our rotation range to 20 to help make our model invariant to small variations in the orientation of objects in our images. Our width shift range parameter was set to 0.1. This allowed our model to learn spatial invariance and improve its robustness to object position variations. Our height shift range was also set to 0.1. Similar to our width shifting, this helped the model learn spatial invariance. Shear: We set the shear range to 0.1 as well, distorting the shapes of objects in our images, and helping our model to learn to recognize objects from different perspectives. We also set our zoom range to 0.1. This helped teach our model to focus on different parts of the image and improves its ability to detect objects at different scales. The horizontal flip parameter was set to 'True', which allowed our model to randomly flip images horizontally. This augmentation technique helped in increasing the diversity of our dataset by creating mirrored versions of some of the images. And lastly, our fill mode parameter was set to nearest, which fills new pixels in our images with the value of the nearest pixel in their original images. This helped in increasing the diversity of our training dataset and prevented overfitting.

Our augmented data was generated spontaneously during training using the flow method of 'ImageDataGenerator'. We set the batch size to 32. The original training dataset ('X_train_resized') was split into training and validation sets ('X_train_aug', 'X_val_aug') along with their corresponding labels ('y_train_aug', 'y_val_aug'). Finally, our transfer learning model was trained using our augmented data. The fit method was called on the model with the training generator ('train_generator') as its input. The 'steps_per_epoch' was calculated based on the length of the training set and batch size. The training was performed for 10 epochs with validation data specified for monitoring our model's performance during training. This was run a total of 10 epochs with an ending accuracy of 0.2819 and an ending validation accuracy of 0.2500.

5.3 Naive-Bayes, Random Forest, and K-Nearest Neighbors

We wanted a good mix of different types of classifiers, so we tried Naive-Bayes, Random Forest, and K-Nearest Neighbors. To start, we imported 'PCA' from scikit-learn's decomposition module, used for performing Principal Component Analysis, 'GaussianNB' from scikit-learn's 'naive_bayes' module, representing the Gaussian Naive Bayes classifier, 'RandomForestClassifier' from scikit-learn's ensemble module, representing the Random Forest classifier, and 'KNeighborsClassifier' from scikit-learn's neighbors module, representing the K-Nearest Neighbors classifier.

'PCA' is set up with 'n_components=100', indicating that we wanted to reduce the dimensionality of the dataset to 100 principal components. 'X_train_pca' and 'X_val_pca' were obtained by applying 'PCA'

transformations to the training and validation sets, respectively. The reshape method was used to flatten the input images before applying PCA. Then three classifiers were initialized: Gaussian Naive Bayes, Random Forest, and K-Nearest Neighbors. Each classifier was trained on the 'PCA'-transformed training dataset ('X_train_pca') along with the corresponding labels ('y_train') using the fit method. The trained classifiers were evaluated on the 'PCA'-transformed validation dataset ('X_val_pca') using the score method, which returned the mean accuracy of the classifier. Then the accuracy of each classifier was printed.

We then combined our efforts with the three different 'PCA' based classifiers with our transfer learning model. Our transfer Learning Model Accuracy ended up being 0.282. And our 'PCA' based classifiers' accuracies were as follows: our Naive Bayes Accuracy was 0.624, our Random Forest Accuracy was 0.8364, and our K-Nearest Neighbors Accuracy was 0.819.

6. Conclusion

Our final loss was 0.4424, and our final validation accuracy was 0.906. Comparing the 'PCA' based classifiers against our CNN model, it can be concluded that the CNN performs better, yielding a validation accuracy of 0.9059, which is notably higher than the accuracies from our 'PCA' based classifiers. The CNN outperforming the 'PCA' based classifiers indicates that the CNN effectively leverages hierarchical features found in images which may not be fully captured in 'PCA' based methods. Traditional machine learning methods such as Random Forest, PCA, and K-Nearest Neighbors rely on dimensionality reduction techniques to represent data. In brain tumor CT scans, these traditional methods could have a difficult time in capturing complex patterns that are often found in medical imaging. Additionally, Random Forest and K-Nearest Neighbors have fixed architectures whereas CNN's have adjustable parameters of convolutional layers during training, which allows them to be highly adaptable and flexible in learning complex representations from data.

One potential improvement that could be used to enhance the performance of traditional machine learning methods such as Random Forest, PCA, or K-Nearest Neighbors in a task such as classification of brain tumor CT scans could be to implement visualization filters such as Sobel or Hewitt filters. These filters highlight specific features within an image such as edges. Applying these filters to brain tumor CT scans can enhance certain features that are relevant for classification, making it easier for traditional machine learning methods to distinguish between different classes. These filters could also be used as a preprocessing step to extract relevant features from the images while reducing dimensionality, improving the performance of PCA or K-Nearest Neighbors which often struggle with higher dimensional data. Below is an example of Ed Sobel and Prewitt Horizontal/Vertical visualization filters done on our brain tumor CT scans dataset. [Fig. 4]

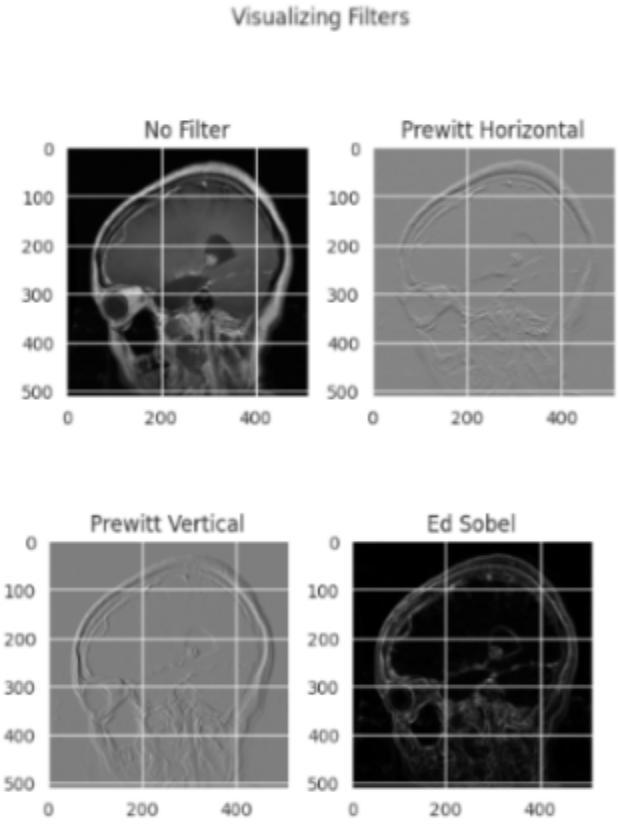

Figure 4: Sobel and Prewitt Visualization Filters

Some potential implications of brain tumor classification CNN models are: assistance in medical diagnosis or treatment of brain tumors, drug discovery and development in brain oncology, and enhancement in performance of brain computer interfaces.

Next steps for our CNN model could include additional improvements and refinement that could be made, validation and testing, and eventually, integration into clinical practice. Integration would require addressing ethical considerations.

In summary, the journey of our CNN model extends beyond its current state, with a trajectory set towards optimizing its efficacy, validating its performance, and navigating the ethical landscape to realize its full potential in revolutionizing the diagnosis and treatment of brain tumors.

References

Baiju Babu Vimala, Saravanan Srinivasan, Sandeep Kumar Mathivanan, Mahalakshmi, Prabhu Jayagopal & Gemmachis Teshite Dalu, (2023, Dec, 27), Detection and classification of brain tumor using hybrid deep learning models, *Scientific Reports*, Retrieved (2024, Apr, 30), from https://www.nature.com/articles/s41598-023-50505-6.

C. L. Choudhury, C. Mahanty, R. Kumar and B. K. Mishra, "Brain Tumor Detection and Classification Using Convolutional Neural Network and Deep Neural Network," 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), Gunupur, India, 2020, pp. 1-4, doi: 10.1109/ICCSEA49143.2020.9132874.

CodeLikeAGirl, (2024, Mar, 7), Revolutionize Dermatology: Skin Lesion Classification Project With CNN and Transfer Learning, *Code with C*, Retrieved (2024, Apr, 30), from https://www.codewithc.com/revolutionize-dermatology-skin-lesion-classification-project-with-cnn-and-transfer-learning/?amp=1.

David N Louis, Arie Perry, Pieter Wesseling, Daniel J Brat, Ian A Cree, Dominique Figarella-Branger, Cynthia Hawkins, H K Ng, Stefan M Pfister, Guido Reifenberger, Riccardo Soffietti, Andreas von Deimling, and David W Ellison, (2021, Jun, 29), The 2021 WHO Classification of Tumors of the Central Nervous System: a summary, *The National Library of Medicine*, Retrieved (2024, Apr, 30), from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8328013/.

Hanaa ZainEldin, Samah A. Gamel, El-Sayed M. El-Kenawy, Amal H. Alharbi, Doaa Sami Khafaga, Abdelhameed Ibrahim, Fatma M. Talaat, (2022, Dec, 22), Brain Tumor Detection and Classification Using Deep Learning and Sine-Cosine Fitness Grey Wolf Optimization, *National Library of Medicine*, Retrieved (2024, Apr, 30), from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9854739/.

Manish Bhobé, (2019, Aug 28), Cats-vs-Dogs — Kaggle Challenge — achieve 97% test accuracy!, *Medium*, Retrieved (2024, Apr, 30), from https://medium.com/@mjbhobe/cats-vs-dogs-kaggle-challenge-achieve-97-test-accuracy-3295636c77a6.

Multinomial Logistic Regression with PyTorch, Retrieved (2024, Apr, 30), from https://www.geeksforgeeks.org/multinomial-logistic-regression-with-pytorch/amp/.

OpenAI. (2024). ChatGPT (July 2021 version) [Large language model]. https://chat.openai.com.

Rebecca L. Siegel MPH, Angela N. Giaquinto MSPH, Ahmedin Jemal DVM, PhD, (2024, Jan, 17), Cancer statistics, 2024, *The American Cancer Society Journals*, Retrieved (2024, Apr, 30), from https://acsjournals.onlinelibrary.wiley.com/doi/10.3322/caac.21820.

Sartaj, (2020), Brain Tumor Classification (MRI), Retrieved (2024, Apr, 30) from https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri/data.

Soumick Chatterjee, Faraz Ahmed Nizamani, Andreas Nürnberger & Oliver Speck, (2022, Jan, 27), Classification of brain tumours in MR images using deep spatiospatial models, *Scientific Reports*, Retrieved (2024, Apr, 30), from https://www.nature.com/articles/s41598-022-05572-6.

The American Cancer Society medical and editorial content team, (2024, Jan, 17), Key Statistics for Brain and Spinal Cord Tumors, *The American Cancer Society Journals*, Retrieved (2024, Apr, 30), from https://www.cancer.org/cancer/types/brain-spinal-cord-tumors-adults/about/key-statistics.html.

Thota Sreenivas, P. Dhana Lakshmi, N.Veda Sravanthi, Sk. Sajeena, G. Raju, S. Praveen Kumar, (2024), A Machine Learning Approach for Brain Tumor Detection Using CNN Algorithm, *International Journal of Scientific Research in Science, Engineering and Technology*, *Volume 11,* (Issue 2), pp. 276-286, doi : https://doi.org/10.32628/IJSRSET.