Department of Electrical and Computer Engineering
North South University

Senior Design Project

Brain Tumor Detection from MRI Using 2D Convolutional Neural Network

Apu Das	**ID # 1620743042**
Meftaul Hafiz	**ID # 1620181042**
Md. Mohiuddin	**ID # 1712466642**

Faculty Advisor:
Mr. Zunayeed Bin Zahir
Lecturer
ECE Department

Spring, 2021

Declaration

This is to declare that no part of this report or the project has been previously submitted elsewhere for the fulfillment of any other degree or program. Proper acknowledgement has been provided for any material that has been taken from previously published sources in the bibliography section of this report.

....
Apu Das
ECE Department
North South University, Bangladesh

....
Meftaul Hafiz
ECE Department
North South University, Bangladesh

....
Md. Mohiuddin
ECE Department
North South University, Bangladesh

Approval

The Senior Design Project entitled "**Brain Tumor Detection from MRI Using 2D Convolutional Neural Network**" by Apu Das (ID#1620743042), Meftaul Hafiz (ID#1620181042) and Md. Mohiuddin (ID#1712466642) has been accepted as satisfactory and approved for partial fulfillment of the requirement of BS in CSE degree program on March 2021.

Supervisor's Signature

[illegible]

Mr. Zunayeed Bin Zahir
Lecturer
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh.

Department Chair's Signature

Dr. Mohammad Rezaul Bari
Associate Professor & Chair
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh.

Acknowledgement

First of all, we would like to express our profound gratitude to our honorable course instructor, **Mr. Zunayeed Bin Zahir,** for his constant and meticulous supervision, valuable suggestions, patience, and encouragement to complete the thesis work.

We would also like to thank the ECE department of North South University for providing us with the opportunity to learn Machine Learning (CSE445) and Pattern Recognition and Neural Network (CSE465) as part of our curriculum for the undergraduate program.

Finally, we would like to thank our families and everyone else who has supported us and provided us with guidance for the completion of this project.

Abstract

Brain Tumor are considered to be one of the most aggressive forms of tumors. But early detection of brain tumors can raise patient survival expectations significantly. That's why accurate and quick detection is a must for patient survival. The traditional way of Brain Tumor detection is doing an MRI of the brain. These MRI images reveal the abnormal cells, which suggest whether the patient is suffering from Brain Tumor or not. But detecting Brain Tumor in the traditional way is very costly, time-consuming, and complex. That's why we propose a Deep Learning based framework to automatically detect Brain Tumor. Deep Learning has already proven to be effective in diagnosing medical images. In this project, we used 2D Convolutional Neural Network (CNN) for the detection. It is a Deep Learning technique for image classification. For training and testing purpose, our proposed 2D CNN architecture used an online dataset named "Br35H :: Brain Tumor Detection 2020", which includes 1500 positive cases and 1500 negative cases of Brain Tumors MRI. These 3000 MRI images will be divided into 2 parts. 70% for the training set (2100 MRIs), 30% for the test set (900 MRIs). Here, we made sure train and test dataset are balanced, that means positive and negative case ratio is maintained. Then we preprocessed the dataset from MRI to tensor, so that we could feed the them as input. Upon preprocessing the images, we used the training portion to train some CNN models. We also train our dataset on some popular deep learning pre-trained models (AlexNet, VGG, ResNet, DenseNet). Then selected the best model based on the training score. Finally, we used the test set for the evaluation. As it is a binary classification, we use the accuracy and F-1 Score for the selection of the CNN model. With our ResNet50 pretrained model, we could able to achieve an accuracy score of 97.22% and F-1 score of 97.22% on the test set.

Table of Content

List of Figures

List of Tables

Chapter 1
Project Overview

1.1 Introduction

Brain is the most sensitive organ of our body, which controls the core functions and characteristics in the human body. But every year a huge number of people are diagnosed with brain tumor. Compared with other cancers such as breast cancer or lung cancer, a brain tumor is not more common but, still, a brain tumor is the number 10th leading cause of deaths worldwide. Record shows that Every year, around 11,700 people are diagnosed with brain tumor. And a large portion of this people die because of its long lasting and psychological impact on patient life. The Brain tumor is caused by tissue abnormality that develops within the brain or in the central spine, interrupting proper brain function. In fact, Brain tumors account for 85% to 90% of all primary Central Nervous System (CNS) tumors. A Brain tumor is marked as Benign and Malignant. Benign brain tumors do not contain cancer cells and grow gradually. They do not spread and commonly stay in one region of the brain, whereas malignant brain tumors contain cancer cells and grow quickly and spread through to other brain and spine regions as well. Therefore, a Malignant tumor is life-threatening and harmful.

The Brain tumor is diagnosed using several techniques such as CT scan, EEG, but Magnetic Resource Image (MRI) is the most effective and widely used method. MRI uses powerful and effective magnetic fields and radio waves to generate internal images of the organs within the body. MRI provides more detailed information on the internal organs and is, therefore, more effective than CT or EEG scanning. But manual examination of MRI images can be error-prone due to the level of complexities. That is why, Deep Learning based automated detection system for Brain Tumor is needed. And, since AI and Deep learning has advanced significantly in the medical science in the past few years, it is time that we build a system to automatically detect Brain tumors from MRI images.

1.2 Deep Learning, CNN & Brain Tumor Detection

With the advancement of the algorithms and as the power of our system has increased significantly, many sectors of bio- medical science are getting digitalized and automated. Thus, it is no wonder that Brain tumors are also getting automated. Brain MRI image is mainly used to detect the tumor and tumor progress modeling process. This information is mainly used for tumor detection and treatment process. As MRI image gives more information compared to the CT scan or ultrasound image, MRI image are best for the detection of Brain tumor. It provides detailed information about brain structure and anomaly in brain tissue.

As mentioned above, MRI images are most effective for the detection of tumors in brain. These MRI images are nothing but images. Therefore, we can handle it with Deep Learning, particularly by Convolutional Neural Network (CNN). CNN is mostly used for tasks that includes images or visual media. So, CNN is the perfect algorithm for detection of brain tumor from MRI images. The automatic brain tumor classification with CNN is very challenging task in large spatial and structural variability of surrounding region of brain tumor. Thus, many deep and complex architecture is needed for this task.

1.2.1 How This Brain Tumor Detection Works

First of all, the MRI images are transformed in a format so the CNN can work with the data. And as CNN or other Deep Learning models work only with numbers, we must convert the image to number so that machine can read values. As all the images are RGB, each image is transformed into 3 channeled matrix or tensor. Then the matrix gets resized into 256x256. After that the matrix gets flipped horizontally with a probability of 0.5. Then the matrix normalized and fed into the training phase. After training the model with all the images in training dataset for some epochs, it can learn to detect MRI images that are positive and negative.

1.2.2 Pros and Cons

Deep Learning based automatic detection of Brain tumors system has many advantages over the traditional approaches. Traditional manual examination can be error-prone due to the level of complexities while some of the models are proved useful. As detection are processed automatically it can remove the need of specially trained person or doctors. Also, as the process is instant, it can save time for patient. And time is valuable especially for patient diagnosed positive with Brain tumor. It can significantly increase the chance of survival for the patient.

On the contrast, it has some disadvantages too. For example, Deep Learning CNN model requires very large amount of data in order to perform better than other techniques. To process large amount MRI images, a very advanced system and GPU are needed. Thus, it is extremely expensive to train a Deep Learning based automatic detection of Brain tumors system due to complexity of required data and complex models. Patients may not get comfortable getting informed by a automated machine if they have Brain tumors. Rather they would prefer to visit a specialized doctor in this field. Also, deep-learning based system can be pretty pricey at the moment, although costs are anticipated to decrease as demand for the technology increases.

1.2.3 Performance

With our proposed model built from scratch, we could able to get an accuracy of 96.67% on the test set, which means our detection model could able to detect an MRI image as positive or negative correctly 96.67% of the times. Precision for this model is 96.65%, which means this model can detect MRI images as positive 96.65% of the time if an MRI image is actually positive. On the other hand, recall is 97.09%, which tells us this model can tell a negative MRI image as negative 97.09% of the time.

The model we built from pretrained deep learning models was even accurate than the model built from scratch. With this model, we could able to get an accuracy of 97.22% on the test set, which means our detection model could able to detect an MRI image as positive or negative correctly 97.22% of the times. Precision for this model is 97.54% and recall is 97.19%. Therefore, we can say our model can outperformed many other models. And with enough data and power, it may able to give a tough time for the specialized doctors in this field too.

1.3 Dataset and Data Partition

We have chosen Kaggle's "Br35H :: Brain Tumor Detection 2020" dataset for our project. It is a balanced dataset with 1,500 MRI images being positive and the same number of MRI images being negative. We have used 70% of the dataset as our training set and the rest of the data as our test set. We used scikit-learn train_test_split function for this task. While splitting the dataset we made sure positive-negative ratio is maintained.

1.4 Our proposed project

We are proposing two deep learning models for automatic detection of Brain tumors. One is built from scratched and other one is a fine-tuned version of one of the most popular and widely used pre-train model "ResNet50". The one built from scratched, has 3 ConV layers, with number of filters are 16, 32, 16 sequentially. Throughout the model, kernel size = 5x5, stride = 1, and padding = 2 are used. Activation function in these layers is ReLU and SoftMax is used in the output layer to get the probability of the MRI image being positive. The input size is (3, 256, 256), where 3 is the number of channel (each channel is for R, G, & B in a RGB image) and 256 is height and weight.

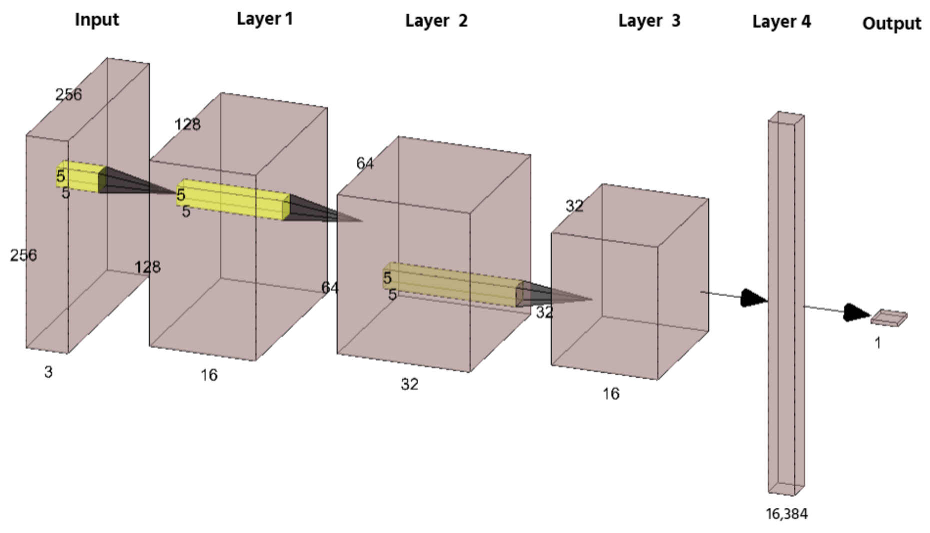


Figure 1.1: Proposed built from scratched model Architecture

In the pretrained model, the ResNet50 [7] has outperformed AlexNet, VGG16 [8], and DenseNet121 by all the evaluation matrices. We tried freezing some layers of the pre-trained models and using pre-trained models for feature extraction only (then use SVM/RandomForest/XGBoost to classify). The earlier one got higher accuracy, precision, and recall score. Therefore, we propose a model that is trained on ResNet50 pretrained version with some layers being frozen.

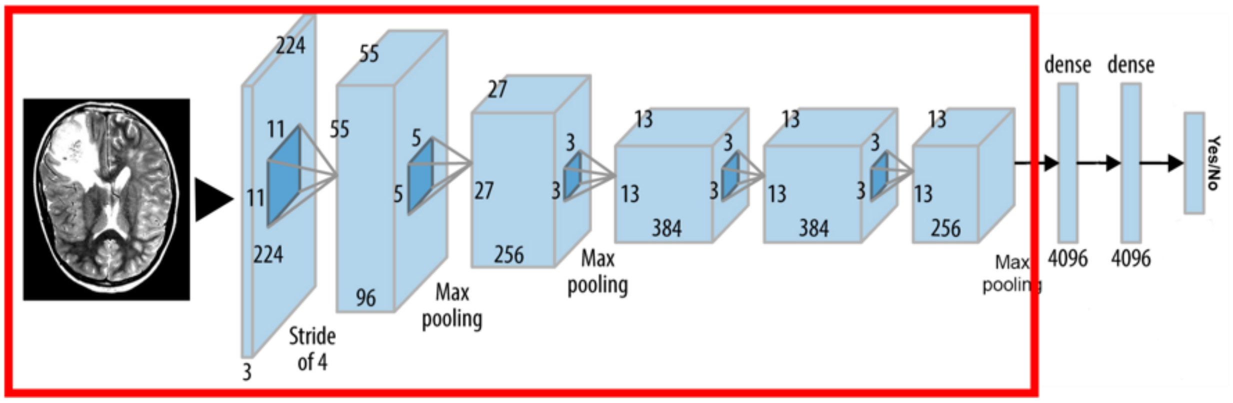


Figure 1.2: Proposed transfer learning Architecture (with ResNet50)

1.5 Motivation

According to researchers and doctors, Brain tumor is one of the most aggressive diseases. It accounts for 85% to 90% of all primary Central Nervous System (CNS) tumors. Every year a huge number of people are diagnosed with brain tumor and die a significant portion of them. But if they were detected timely, their life would have been save or live longer than they lived. As of now the best technique to detect brain tumors is MRI. Manual examination can be error-prone due to the level of complexities. And that is why we propose a deep learning based automated classification of brain tumor.

1.6 Summary

In this chapter, we have briefly described the basics of Deep Learning, our proposed models, advantages and disadvantages of our model, performance of our models, the main idea of our project, and the motivation behind the need of this project. The following chapters describe the related works on this field, theory and details of of the model and technology used, architectural designs and structure of the model, and many other aspects of the research.

Chapter 2
Related work

2.1 Introduction

The existing work related to Deep Learning based Brain Tumor detection that we had discovered and found useful, are described in this chapter. As we looked for similar systems, we realized that there are relatively few that combine both Deep Learning and Machine Learning algorithms. But as we looked at several similar works, they provided us some insight into what we were up against. For example, we got the idea of using of data augmentation, feature extraction, etc from these papers.

2.2 Existing work related to Deep Learning based Brain Tumor detection

Throughout our research period, we have read many scientific paper and research journal to get the idea of current position of the system. While reading these papers, we also get some insights and ideas to try on our project.

2.2.1 Identification of Brain Tumor using Image Processing and Neural Networks [1]

For detecting the brain tumor, this paper used the most famous and excessively used medical imaging technique - Magnetic Resonance Image (MRI). The intension of this project was to obtain results similar to the results given by a specialist. The methods they used for this paper are Thresholding, Contour and Shape method.

In Thresholding method, the pixels are separated and grouped together on the basis of their intensity values. Though this method is proved to be useful, but in some cases, where the tumor is in the very first stage, the intensity of the color contrast of the lump formed is very fade and the pixels forming the tumor cannot be grouped together. In Contour and Shape method, tumor boundaries and segments of lump were detected. But as this method can’t detect the irregular-sized tumor and the ones formed near the inner lining of the brain very well, accuracy was not good enough with this method. They preprocessed the MRI images to gray-scale with a dimension of “227*227”. To trained the network, they used MATLAB and a pre-trained CNN algorithm 'AlexNet’. Image processing has found its way in the biomedical stream and will continue to grow.

2.2.2 Brain Tumor Detection Using Shape features and Machine Learning Algorithms [2]

This paper proposes two approaches for brain tumor classification depended on machine learning algorithms. Shape features are extracted and used for classification. Numbers of shape features are considered in this paper include Major axis length, Minor axis length, Euler Number, Solidity, Area and Circularity. Here they used Adaptive Threshold, C4.5 decision tree and Multi-Layer Perceptron and they got maximum 95% precision from Multi-Layer Perceptron.

For the purpose of classification, C4.5 and Multi-Layer Perceptron are used. The maximum precession of about 95% is achieved by considering 174 samples of brain MR Images and using MLP algorithm. Weka tools are used for brain MR Images classification. Brain MR Images were classified using the C4.5 algorithm and Multi-Layer Perceptron (MLP) with 55% percentage split. In 55% percentage split, used 55% of the samples in the training process the rest of the samples have been used in the test. It is seen from the table (1) the C4.5 algorithm has the average TP rate and FP rate 0.897 and 0.017 respectively. The precision was of about 91% To increase this precession can use a large dataset and add other features such as texture and intensity-based features.

2.2.3 A Deep Learning Approach for Brain Tumor Classification and Segmentation Using a Multiscale Convolutional Neural Network [3]

In this paper, Francisco Javier Díaz-Pernas et al have presented a Deep Convolutional Neural Network that includes a multiscale approach that means input images are processed in three spatial scales along different processing pathways.

Their mechanism is inspired by Human Visual System. The proposed NN model can analyze MRI images without the need of preprocessing of input images to remove skull or vertebral column parts in advance. They were able to achieve an accuracy score of 97.3% with their proposed model.

2.2.4 MRI-Based Brain Tumor Classification Using Ensemble of Deep Features and Machine Learning Classifiers [4]

In their proposed framework, Kang et al have used several pre-trained deep CNNs to extract deep features from MRI. The extracted deep features are then evaluated by several machine learning classifiers.

Then the top three performers in extracting deep features are selected and concatenated as an ensemble of deep features which is then used as input feature into several machine learning classifiers to predict the final output. To extract the features, they used some very popular pre-trained CNN models like ResNet50, ResNet101, DenseNet121, DenseNet169, AlexNet, VGG16, VGG19, Inception V3, MobileNet v2, and SuffleNet v2. To classify, they used many powerful ML algorithm like SVM with RBF kernel, SVM with Sigmoid kernel, SVM with Linear kernel, k-NN, AdaBoost, and some others. The best score they got is 97.85% which is using SVM with RBF kernel.

2.2.5 Deep Learning-Based Brain Tumor Classification in MRI images using Ensemble of Deep Features [5]

In the proposed framework, three different deep features from brain MR images are extracted using three different pre-trained models (DenseNet-169, ShuffleNetV2, MnasNet). After that, they reduced the dimension of the extracted deep features by PCA algorithm.

Then for the classification part, these features are concatenated and fed into the classification module. This classification module is a fully connected (FC) layers. The final output layer was a Softmax layer. With their proposed model, they could able to achieve 94.18% accuracy.

2.2.6 Brain tumor classification in MRI image using convolutional neural network [6]

In this paper, the authors have used some pre-trained models to classify Brain Tumors. These models are VGG16, Inceptionv3, and ResNet50. They freeze one or multiple layers and then train with the whole training set. Here, freezing layers means not updating any weight in the layers.

They used convolutional neural network (CNN) along with Data Augmentation and Image Processing to categorize brain MRI scan images into cancerous and non-cancerous. Then using the transfer learning approach they have compared the performance of their scratched CNN model with pretrained models. This method requires very less computational power and has much better accuracy results, VGG-16 achieved 96%, ResNet-50 achieved 89% and Inception-V3 achieved 75% accuracy.

2.3 Summary

In summary, there’s been a lot of research on deep learning based automated detection of Brain Tumors. Most of the research was to find a model that use less computational power and has much better results. Some of the papers were able to achieve about 98% to 99% accuracy in the test set.

Chapter 3
Theory

3.1 Introduction

The human body is structured by many types of cells. Each cell has a different function. The cells in the body are growing day by day by maintaining an order. Then cell divide and by breaking a cell there make another cell. The newly growing cells helps to keep the human body healthy and working properly. But when some of the body cells lose their capability to control their growth they grow without any order. The extra cells formed from a mass of tissue which is called tumor. This tumor can grow in any part of the body. Any other part of the body is more recoverable than brain because brain is the most sensitive organ in human body. Therefore, if any kind of problems occur in brain it suffers us most.

Among all other tumors, Brain tumor is one of the most aggressive disease in the world. Most of the times it took the life of the patient. Brain tumor account for 85% to 95% of all primary Central Nervous System(CNS) tumors. Every year more than 12000 people are diagnosed with brain tumor. To detect this aggressive disease one of technique is doing MRI. This method is very time consuming. It is not appropriate for large amount of data. It also cause noise for the operator who operate the MRI machine which can lead to inaccurate classification. Large volume of MRI is to analyzed thus automated systems are needed as it they are most cost effective. The MRI human brain images are classified by using supervised technique like neural network. Deep Learning has already proven to be effective in diagnosing medical images. In this project, we used 2D Convolutional Neural Network (CNN) for the detection. It is a Deep Learning technique for image classification.

3.2 Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. It is a subset of machine learning, which is basically a neural network with three or more layers: one input layer, one output layer, and one or more hidden layers. These neural networks try to simulate the human brain, which allows it to learn from training data. Deep learning drives many Artificial Intelligence (AI) or automated applications and services that performs analytical and physical tasks without human intervention. This technology lies behind everyday products and services such as automated biomedical products, digital assistants, image or other media manipulation, voice-enabled TV remotes, and credit card fraud detection as well as emerging technologies such as self-driving cars. Deep learning is getting lots of attention lately and for good reason. It’s achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, sound, or any other files. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance particularly on programs that are done repeatedly by human. The models are trained by using a large set of labeled data and neural network architectures that contain many layers. We used deep learning in our research to diagnose the MRI images that are labeled as positive and negative. Here positive means existence of Brain tumor in the MRI and negative means there are no tumor in the brain MRI image.

3.3 Convolutional Neural Network

3.3.1 Theory

CNN means Convolutional Neural Network. It is a particular type of feed-forward neural network in Deep Learning. CNN is widely used in image recognition. CNN represent the input data in the form of multidimensional arrays. It works well for a large number of labeled data. Neural network is mostly about matrix multiplications but that is not the case with CNN. It uses a special technique called Convolution. It has applications mostly in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

CNN extract each and every spatial information of input image, which is known as receptive field. It assigns weights for each neuron based on the significant role of the receptive field. So that it can discriminate the importance of neurons from one another. Here is the image of CNN architecture. The architecture of CNN consists of three types of layers: Convolution layer, Pooling layer, and Fully connected Layer. Here, Convolutional layers are the major building blocks used in convolutional neural networks (CNN). The other two may or may not occur, depending on the need of them.

3.3.2 Architecture

A CNN architecture is formed by a stack of distinct layers that transform the input into an output through a differentiable function. A few distinct types of layers are commonly used. These are Convolution layer, Pooling layer, and Fully connected Layer.

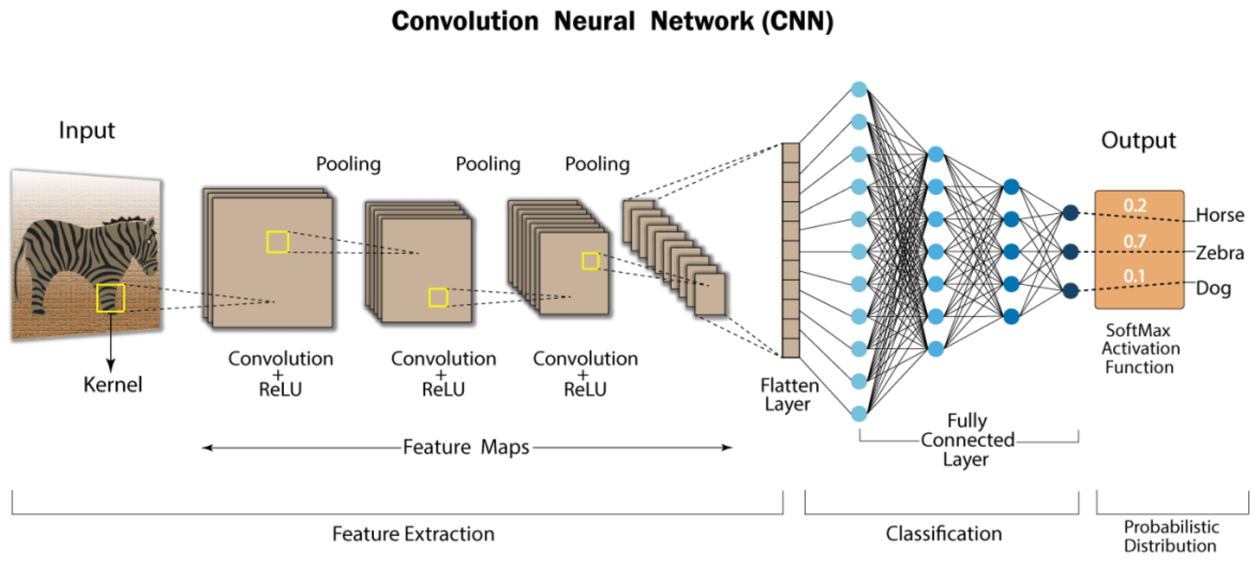


Figure 3.1: Convolutional Neural Network Architecture

3.3.3 How do convolutional neural networks work

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. As mention in the upper sections, It has three main types of layers: Convolutional layer, Pooling layer, and Fully-connected (FC) layer. The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. The FC layer is call classification layer, while the combination of other two is called feature extractor. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple spatial features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the filter entries and the input, producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input. A non-linearity function is mapped after the conv layer.

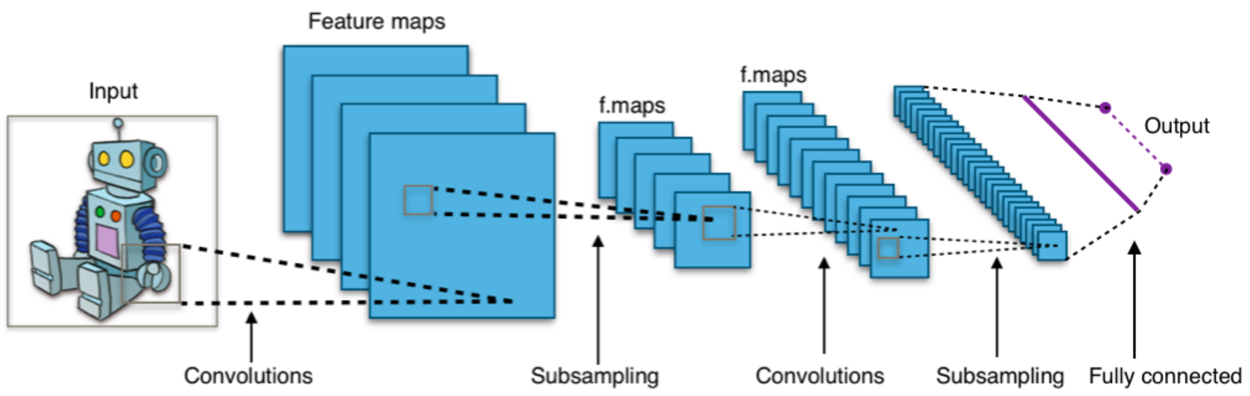


Figure 3.2: Convolutional layer, Pooling layer, and Fully-connected (FC) layer

Another important concept of CNNs is pooling. It is a form of non-linear down-sampling. There are several non-linear functions to implement pooling. Among them max pooling is the most common. It partitions the input image into a set of rectangles and, for each such sub-region, outputs the maximum. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network. Thus, it also controls the overfitting problem. This technique is also known as down-sampling as it reduces the spatial size. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture.

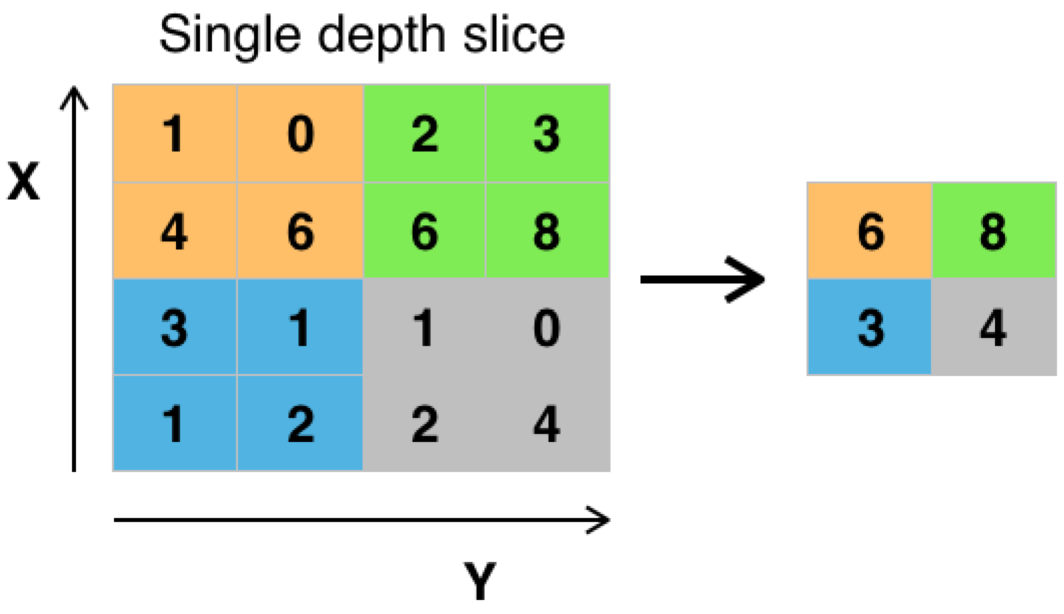


Figure 3.3: CNN Pooling layer

After several convolutional layers and pooling layers, the final classification is done by fully connected layers. After all the previous layers, the output got flattened and fed into the classification module. This classification module is the part that takes most of the memory, complexity, and parameters of a CNN model. Here, Neurons (represented as circle in the below image) in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (vector addition of a learned or fixed bias term).

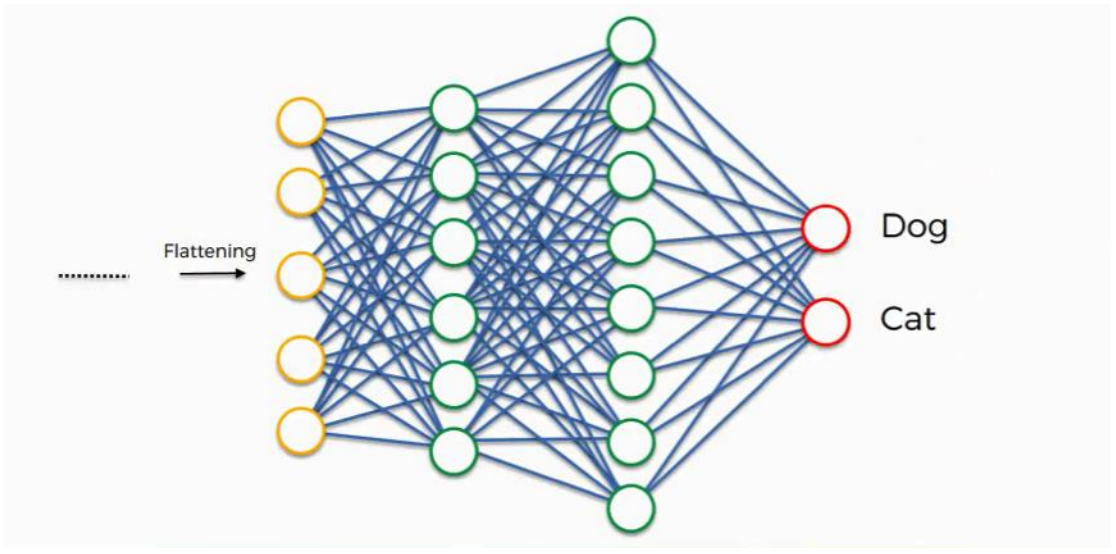


Figure 3.4: CNN Fully-connected (FC) layer

3.4 Transfer Learning

Transfer learning is a machine learning method where a model developed and specialized for a task is reused as the starting point for another model on another task. It is a popular approach in deep learning where pre-trained models are used as the starting point on image processing, computer vision and natural language processing tasks. Given that the vast computation and time resources required to develop convolutional neural network models from zero is a huge task, thus, transfer gives the models a huge jump in skill that they provide on related problems. And as the pre-trained models have already learned to do some tasks, it is considered that the model can certainly do some other tasks too.

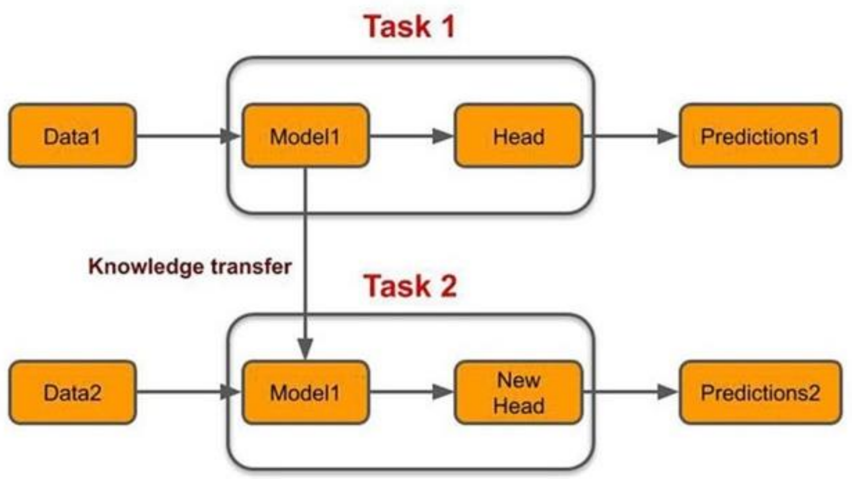


Figure 3.5: Transfer Learning Concept

3.5 Pre-trained Models

We have used some of the most popular, heavily used, and advanced pretrained models exist today. The pretrained models we used, are AlexNet, VGG, ResNet, and DenseNet. Here, VGG, ResNet, and DenseNet have different variation depending on the number of layers it has. We have used VGG16, ResNet50, and DenseNet121 as the higher version are too heavy for our system and require GPU to work properly.

3.5.1 AlexNet

AlexNet [10] is a classic convolutional neural network. It is the first convolutional neural network which used GPU to boost performance. There have many neural network models. But if comes to comfort and flexibility then AlexNet come first. AlexNet won the ImageNet large-scale visual recognition challenge in 2012. The model was proposed in 2012 in the research paper named ImageNet classification with Deep Convolution Neural Network by Alex Krizhevsky and his colleagues.

The AlexNet has eight layers with learnable parameters. The model consists of five layers with a combination of max pooling followed by 3 fully connected layers and they use Relu activation in each of these layers except the output layer. They found out that using the ReLU as an activation function accelerated the speed of the training process by almost six times. They also used the dropout layers, that prevented their model from overfitting. Further, the model is trained on the ImageNet dataset. The ImageNet dataset has almost 14 million images across a thousand classes. Some of the key features of AlexNet are ReLU Nonlinearity, Multiple GPUs, Overlapping Max Pooling, Data Augmentation, and Dropout. As a milestone in making deep learning more widely-applicable, AlexNet can also be credited with bringing deep learning to adjacent fields such as natural language processing and medical image analysis.

3.5.2 VGG

Visual Geometry Group is a classical convolutional neural network architecture. It was used to win ILSVR(Imagenet) competition in 2014. This architecture follows the arrangement of convolution and max pool layers consistently throughout the whole architecture. VGG- model was proposed by K. Simonyan and A. Zisserman in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. This architecture achieved top-5 test accuracy of 92.7% in ImageNet, which has over 14 million images belonging to 1000 classes.

The input is fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional layers, where the filters were used with a very small receptive field: 3×3. In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel. Spatial pooling is carried out by five max-pooling layers. Max-pooling is performed over a 2x2 pixel window, with a stride of 2. Three Fully-Connected (FC) layers follow a stack of convolutional: the first two have 4096 neurons each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. All hidden layers are equipped with the rectification (ReLU) non-linearity. The advantages of VGG includes accurately Identified, conveniency, efficiency, effectiveness and high accuracy.

3.5.3 ResNet

ResNet, short for Residual Network is a specific type of neural network that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper “Deep Residual Learning for Image Recognition”. ResNet won 1st place in the ILSVRC 2015 classification competition with a top-5 error rate of 3.57% and won the 1st place in ILSVRC and COCO 2015 competition in ImageNet Detection, ImageNet localization, Coco detection and Coco segmentation.

ResNet network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into the residual network. The ResNet has different versions, like ResNet18, ResNet34, ResNet50, ResNet101, etc. The numbers denote the number of layers in the CNN.

3.5.4 DenseNet

DenseNet [9] is one of the new discoveries in neural networks for visual object recognition. DenseNet is quite similar to ResNet with some fundamental differences. ResNet uses an additive method (+) that merges the previous layer (identity) with the future layer, whereas DenseNet concatenates (.) the output of the previous layer with the future layer. DenseNet was proposed in the paper “Densely Connected Convolutional Networks” by Gao Huang et el.

DenseNet was developed specifically to improve the declined accuracy caused by the vanishing gradient in high-level neural networks. In simpler terms, due to the longer path between the input layer and the output layer, the information vanishes before reaching its destination. The DenseNet has different versions, like DenseNet-121, DenseNet-160, DenseNet-201, etc. The numbers denote the number of layers in the CNN.

3.6 Magnetic Resonance Images (MRI)

MRI, or magnetic resonance imaging, uses strong magnetic fields to change the spin of atoms in our bodies. Radio signals detect these tiny changes. MRI computers process this information and construct images of soft tissues inside the body, from the brain to blood vessels. An advantage of MRI is it is virtually harmless to the patient because, unlike CT scanners, MRI does not generate radiation.

3.7 Loss Function

In our research, we have used Binary Cross Entropy Loss function. It compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value. If model's classification output in incorrect, this Binary Cross Entropy loss function penalizes the model heavily. Thus, it makes sure the model converges to the local minimum faster. The equation is:

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

3.7 Summary

As Deep Learning progressed and advanced new algorithms new idea emerged, thus now it it possible to use Deep Learning to build a model that can detect Brain tumors automatically. In this chapter, we have discussed what Deep Learning is, how it works, transfer learning, some of the pre-trained models that are most popular and used in our research.

Chapter 4
Workflow of the system

4.1 Introduction

Throughout the research, we have used some procedure and maintained a workflow for our project. It includes data partition phase to the classification phase.

4.2 Procedure and Workflow

Data preprocessing is same for both scratch model and transfer learning. But model workflow is different as they have very different nature and architecture.

4.2.1 Data Preprocessing

Our dataset is a balanced dataset with 1,500 MRI images being positive and the same number of MRI images being negative. We have used 70% (2,100 MRI images) of the dataset as our training set and the rest of the data as our test set. We used scikit-learn train_test_split function for this task. While splitting the dataset we made sure positive-negative ratio is maintained.

Then we used Deep Learning framework, Pytorch, to read images and resize them. The result of this transformation is a (3, 256, 256) tensor. As it is an RGB image it has 3 channels and we defined height-width in the transformation process. For the pre-trained phase, we cropped the image to (3, 224, 224) after resizing the images as the pre-trained models required this size. We also performed random horizontal flip with a probability of 0.5.

4.2.2 Model Workflow for Scratched Built Model

In our scrathed built model, we have an input layer, an output layer, and 4 hidden layers. The input layer has a dimension of (3, 256, 256) and the output is a single scalar value which is a probability of an MRI image being positive. This value is calculated by a sigmoid activation function. Among the 4 hidden layers, 3 layers are used for feature extraction and 1 layer for classification. Those feature extraction layers are stacked with 4 layers - 1 Conv layer, 1 batch normalization layer, an activation layer, and the last layer of the stack is a MaxPooling layer.

For the whole architecture, we have used ReLU as our activation function except the output layer. For the whole project, up until now, we have set batch size = 128 and learning rate = 0.001. Here learning defines the step size and batch size defines when we should update our model weights.

4.2.3 Model Workflow for Transfer Learning

We have used two different approaches for fine-tuning our model with transfer learning. One is “freeze some layers of the pre-trained models and later train for the training set” and the other one is “use pre-trained models for feature extraction only and later use SVM/RandomForest/XGBoost for classification”. For both cases, we have set batch size = 32 and learning rate = 0.001/0.0015. We used both learning rate simultaneously.

In the first case, we have frozen some of the layers in the pre-trained models. And the trained for 10 epochs. Here freezing means not updating any weights in the frozen layers. It will help the model to generalize well for both training and test set. For the other one we used the models to extract the spatial features from the image only. Then upon extracting features from the input image we used SVM, RandomForest, and XGBoost to classify whether the MRI image is positive or negative.

4.3 Model Diagrams and Workflow

Diagram and workflow for the model built from scratch: The diagram and workflow of the model which is built from scratch is given below.

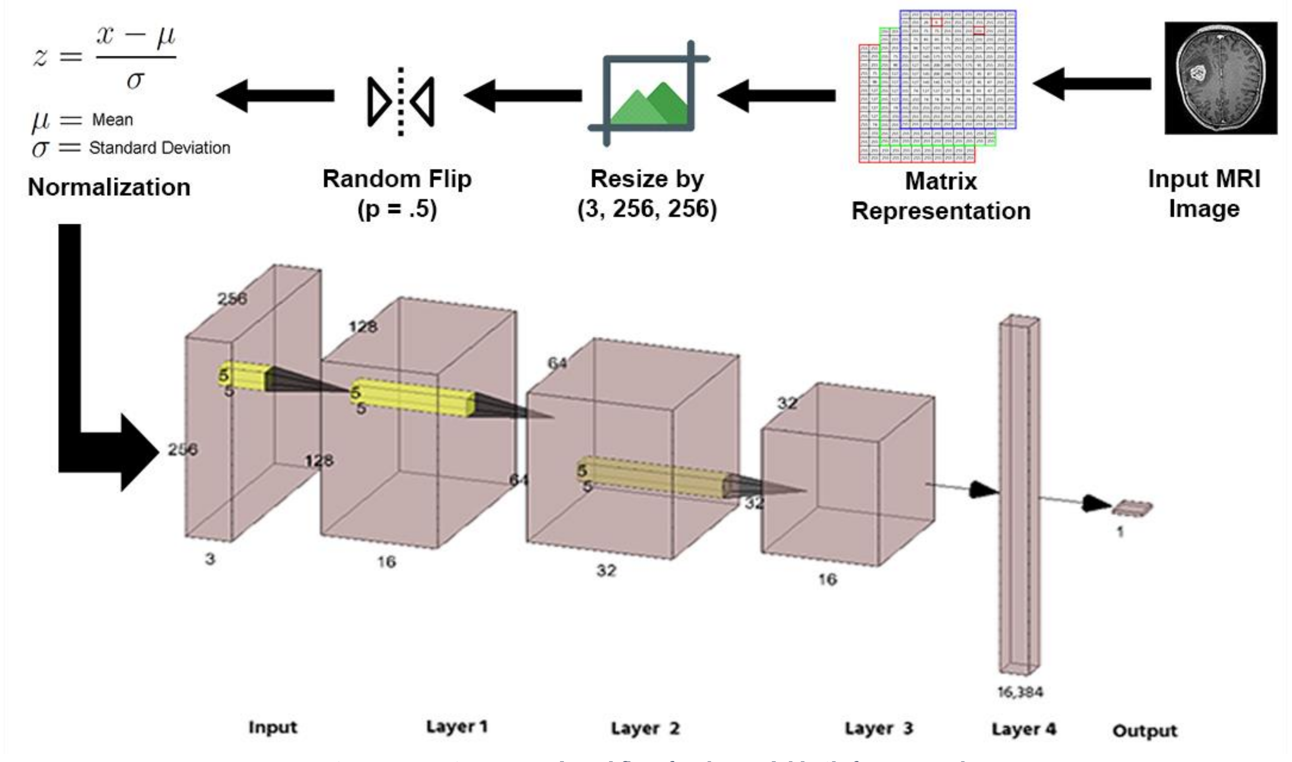


Figure 4.1: Diagram and workflow for the model built from scratch

Diagram and workflow for the pre-trained models: The diagrams and workflows for the most popular pre-trained models are given below. Here, one is for frozen layers and another one is for feature extraction only.

Pre-trained model built with frozen layers:

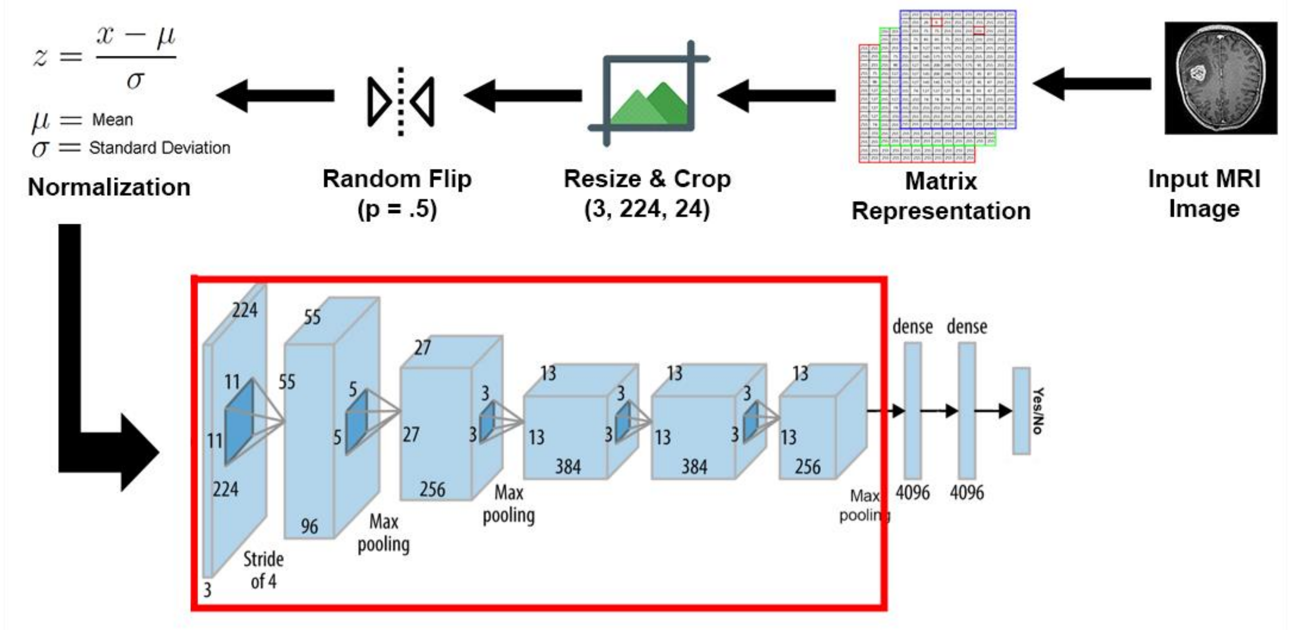


Figure 4.2: Diagram and workflow pre-trained model built with frozen layers

Diagram and workflow pre-trained model for feature extraction only:

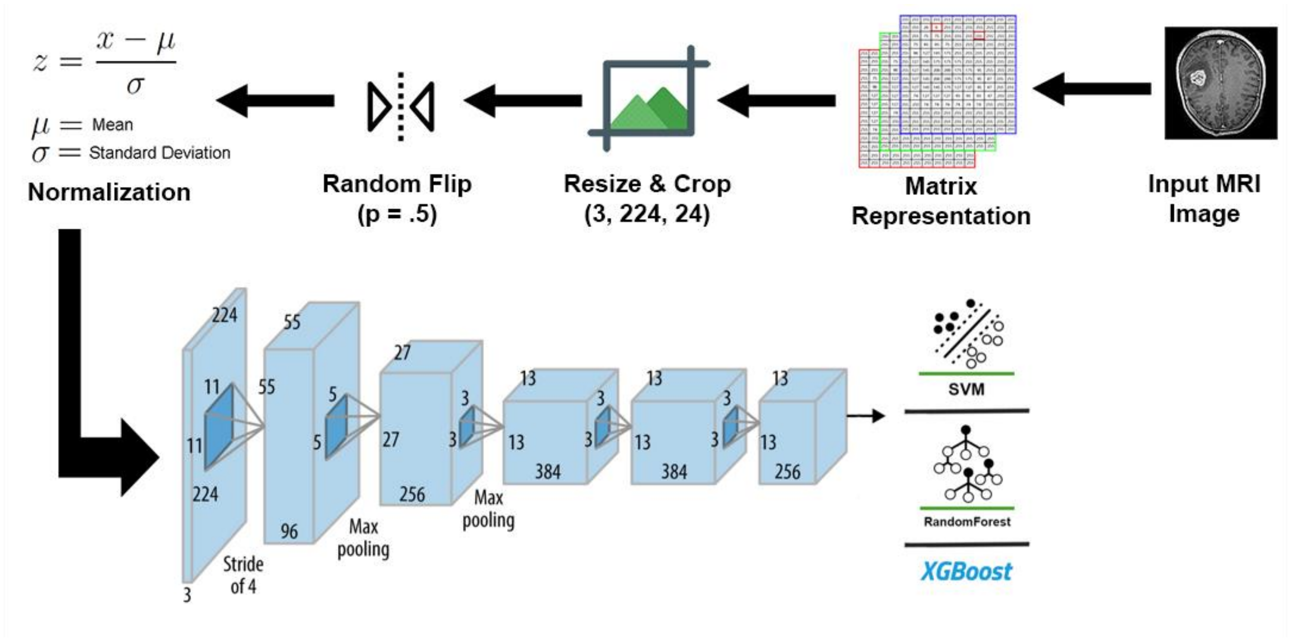


Figure 4.3: Diagram and workflow pre-trained model for feature extraction only

4.4 Summary

In this chapter, we have discussed the Diagrams and workflows for both scratch model and transfer learning models. Though the description of the models is simplified, it was an extensive amount of work to find the perfect diagram and workflow through trial and error.

Chapter 5
Frameworks and Libraries

5.1 Introduction

To conduct our research, we have used some frameworks and libraries. These frameworks and libraries have saved us a lot of time and effort. All of them are coded by Python Programming Language. Some of the frameworks and libraries we used are PyTorch, Scikit-learn, numpy, pandas, etc. As a code editor or interactive computational environment, we used jupyter notebook and Google Colaboratory. Jupyter Notebook is a web-based interactive computational environment for creating notebook documents. And Google Colaboratory (known as Google Colab) is a free Jupyter notebook environment that runs in the cloud.

5.2 Python Programming Language

Python is one of the most popular and post used programming languages in the world. It is an interpreted high-level general-purpose programming language. It is an object-oriented (OOP) language. Its design philosophy emphasizes on code readability with a use of significant indentation. It helps programmers write clean, fewer, and logical code for small and large-scale projects. It is dynamically-typed and garbage-collected. And It also supports multiple programming paradigms, including structured programming, object-oriented programming, and functional programming.

Python Programming Language is often described as a "batteries included" language due to its comprehensive collection standard library. It is regarded as one of its greatest strengths. It has a huge collection of libraries for scientists and researchers, which makes it easy for them to work fast and accurately. We have used some of the Python libraries for our research, including PyTorch, Scikit-learn, os, shutil, numpy, pandas, matplotlib, seaborn, random, and so on.

5.3 PyTorch

PyTorch is a free and open-source machine learning library, developed by Facebook's AI Research team (FAIR). It is based on the Torch library, is used for many applications including computer vision and natural language processing. Modified BSD is a free and open-source operating system. As a result, PyTorch has both a Python and a C++ interface. In addition to the Python interface, PyTorch provides a C++ version as well. There are two high-level functionalities that PyTorch offers. One is GPU-accelerated tensor computation (similar to NumPy) and the other one is Deep neural networks created with a type-based automated differentiation system. It has 3 modules: Autograd module, nn module, and optim module.

5.3.1 Autograd module

The first one is **Autograd module**. In this module, a mechanism called automatic differentiation is used in PyTorch's code. In order to compute gradients, a recorder captures what operations have been done and then repeats them backwards. To save time on a single epoch, this technique calculates the parameter differentiation at the forward pass.

5.3.2 nn module

Next one is, **nn module**. It's straightforward to construct computational graphs and calculate gradients using PyTorch autograd, however raw autograd can be a bit low-level for defining complicated neural networks. The nn module comes in handy here.

5.3.3 optim module

Last one is **optim module**. It implements several optimization methods used in the construction of neural networks. So, there is no need to start from scratch when it comes to the most widely utilized approaches.

5.4 Scikit-learn

Scikit-learn is a free machine learning library for the Python programming language. It has various classification, regression and clustering algorithms including support vector machines (SVM), random forests (RandomForest), various gradient boosting, k-means clustering, DBSCAN, etc. It is designed to work with the NumPy and SciPy, both of these libraries are hugely used for Python numerical and scientific works.

Particularly for our purpose, we have used its train_test_split method to split our dataset into train and test set. We have also used its SVM, RandomForest, XGBoost, OneVsRestClassifier etc algorithm to classify. We used some other of its methods to generate accuracy, precision, recall, f1-score, confusion matrix and many other evaluation metrics.

5.5 Jupyter Notebook

Jupyter Notebook is a web-based interactive computational environment for creating notebook documents. It was formerly known as IPython Notebooks. It is a browser-based REPL containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media. But underneath the interface, a notebook is a JSON document, following a versioned schema, usually ending with the ".ipynb" extension.

Many kernels may be connected to Jupyter Notebook, which allows programming in multiple languages. As of now, it supports Python, R, Julia and Haskell. There are several sorts of requests that may be handled by a Jupyter kernel, such as execution of code, completion of code, or inspection of code. Kernels in Jupyter are not aware that they are tied to a single document, thus they can be connected to several clients at once, unlike many other notebook-like interfaces that are. Usually a single-language kernel is supported, however there are certain exceptions. By default, Jupyter

Notebook ships by the IPython kernel installed on it. We mostly used this interactive computational environment in 499A, where we developed simpler models from scratch. But as model got complex and heavy, most of the pre-trained models, we switched to Google Colaboratory.

5.6 Google Colaboratory

Google Colaboratory is a free Jupyter notebook environment that runs in the cloud and stores its notebooks on Google Drive. It is alson known as Google Colab. Colab was originally an internal Google project; an attempt was made to open source all the code and work more directly upstream, leading to the development of the "Open in Colab" Google Chrome extension, but this eventually ended, and Colab development continued internally. The Google Colaboratory UI only allows for the creation of notebooks with Python 2 and Python 3 kernels; however, an existing notebook whose kernelspec is IR or Swift will also work, since both R and Swift are installed in the container.

5.7 Summary

Frameworks and libraries mentioned above are very popular and heavily used by researchers all over the world. Using these frameworks and libraries has saved us a lot of time and effort. And the Google colab has also offer us the opportunity to run our code on cloud with a better hardware system than our local system.

Chapter 6
Model Description

6.1 Introduction

We have developed multiple deep learning models for automatic detection of Brain tumors. One of the models is built from scratched with PyTorch. Other several models are fine-tuned version of most popular and widely used pre-train models AlexNet, VGG, ResNet, and DenseNet.

6.2 Proposed Model (built from scratch)

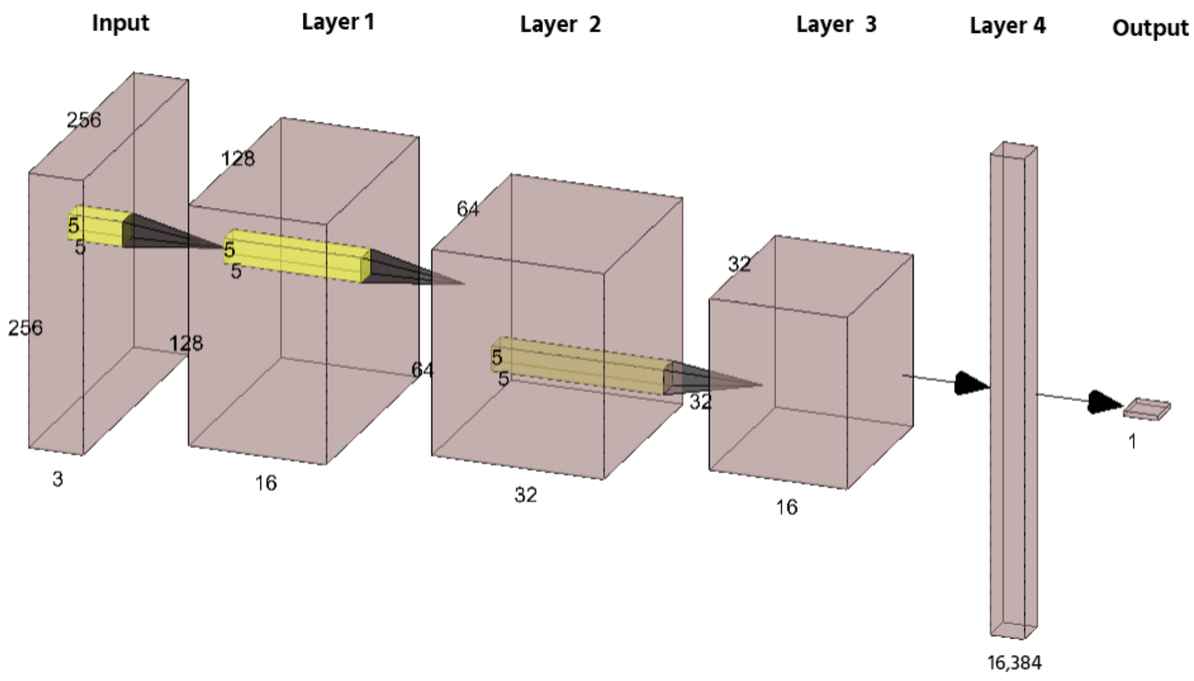


Figure 6.1: CNN Architecture for proposed Model

The above proposed model is generated on 499A. It is built from scratch and we achieved an accuracy of 96.67% and f1-score of 96.22%. Here, the input size is (3, 256, 256), which was generated and transformed by PyTorch torch.transforms module. The architecture has following layers:

TABLE- 6.1. Proposed Model Architecture

Layer Number	Layer Type	Details	Activation

1	Input Layer	RGB Image, 256x256 resolution. All pixel values are real numbers scaled between 0-1	None
2	Convolutional Layer	16 filters, kernel size = 5x5, stride = 1, padding = 2	ReLU
3	Batch Normalization	Number of channels = 16. The mean and standard deviation is calculated for each incoming channel. The output is a tensor where each channel is separately normalized (z-scores).	Not Applicable
4	Max Pooling Layer	Kernel size = 2, stride = 2	Not Applicable
5	Convolutional Layer	32 filters, kernel size = 5x5, stride = 1, padding = 2	ReLU
6	Batch Normalization	Number of channels = 32. The mean and standard deviation is calculated for each incoming channel. The output is a tensor where each channel is separately normalized (z-scores).	Not Applicable
7	Max Pooling Layer	Kernel size = 2, stride = 2	Not Applicable
8	Convolutional Layer	16 filters, kernel size = 5x5, stride = 1, padding = 2	ReLU
9	Batch Normalization	Number of channels = 16. The mean and standard deviation is calculated for each incoming channel. The	Not Applicable

		output is a tensor where each channel is separately normalized (z-scores).	
10	Max Pooling Layer	Kernel size = 2, stride = 2	Not Applicable
11	Flattening Layer	16,384 neurons	
12	Fully Connected Layer	Output a binary value (Probability of an MRI image being positive)	SoftMax

6.3 Pretrained Models using transfer learning

We have used two different approaches for fine-tuning our model with transfer learning. One is "freeze some layers of the pre-trained models and later train for the training set" and the other one is "use pre-trained models for feature extraction only and later use SVM/RandomForest/XGBoost for classification". For both cases, we have set batch size = 32 and learning rate = 0.001/0.0015. We used both learning rate simultaneously.

In the first case, we have frozen some of the layers in the pre-trained models. And the trained for 10 epochs. Here freezing means not updating any weights in the frozen layers. It will help the model to generalize well for both training and test set. For the other one we used the models to extract the spatial features from the image only. Then upon extracting features from the input image we used SVM, RandomForest, and XGBoost to classify whether the MRI image is positive or negative.

6.4 Summary

In this chapter, we have discussed the both scratch model and transfer learning models. We have selected these models with an extensive amount of work through trial and error.

Chapter 7
Design Impact

7.1 Introduction

The fatality rate of people owing to brain tumors was high a few years ago, but this rate has dramatically dropped due to early diagnosis of brain tumors. The chances of a patient living a long life are boosted when a brain tumor is diagnosed accurately in its early stages. We present a computationally efficient and accurate brain tumor segmentation approach in this study. The proposed approach is broken into three parts. If a brain tumor can be predicted, the tumor can be removed from the brain. Our project's aim is to find an effective method for detecting brain tumors in the early stage.

7.2 Economic impact

Brain tumor is one of the most severe and frequent neuro disorders in the population and has dramatic health effects as well as socio-economic implications. The direct costs of treating brain tumors are becoming increasingly outdated as treatment paradigms for gliomas evolve. Estimates of the present economics of brain tumors are required. These studies should contain in-depth analyses of direct and indirect costs, as well as empirical data on out-of-pocket spending. An updated detailed analysis of the costs borne by patients is required.

7.3 Environmental impact

Our project is free from all sorts of materials and chemicals that may harm the wildlife or the environment. Though our project may not aid the environmental development, it in no way harms it. So, it is safe to say that our project does not have any negative environmental impact.

7.4 Social impact

Our project can be put to a wide range of uses. The social impact of our project is very positive. The awareness among the people will grow up and people who get the accurate result of disease, thus more time to get treatment. So, we can say that it is expected to have a remarkable social impact.

7.5 Political impact

Our project does not have any direct impact on the political aspect.

7.6 Ethical impact

The aim of our project is to provide the accurate result of brain tumor earlier steps, and to help make people’s life easier, and open up new possibilities for research and development in this field. It intends to help people. Thus, it can be said that the project has a positive ethical impact.

7.7 Health and safety impact

The vision for the project was to positively impact the health and safety of its user. The project is meant to help brain tumor patients and detect their accurate illness report earlier and get them a chance to get proper treatment earlier and live a better life, help doctors as a third hand during medical procedures, and aid people in unsafe situations. Another point worth mentioning is that our project does not use any harmful chemicals or substances, or emit any harmful radiation that could negatively affect the manufacturers, users or the environment. Thus we can say that it has an overall positive health and safety impact.

7.8 Manufacturability

The project requires components which are widely available. The mechanical design and the systems are simple and have been precisely documented. Thus it would not be a complex system to manufacture.

7.9 Summary

This chapter describes some of the positive and negative effect of our project on society, environment, economy, health, politics etc.

Chapter 8
Results and Analysis

8.1 Introduction

The results and findings of our project are discussed in this chapter. We have used accuracy, precision, recall, and f1-score for evaluating our research.

8.2 Result and findings

If we look at result table, we see that, in 499A, we got accuracy of 96.67%, we got precision 96.65%, Recall 97.09% and F-1 score 96.22%. In 499B we trained several CNN models like AlexNet, VGG and ResNet. In these 3 models, we got best result in ResNet50. We got up to 97.22% accuracy, 97.54% precision, 97.19% recall, and f1-score of 97.22%.

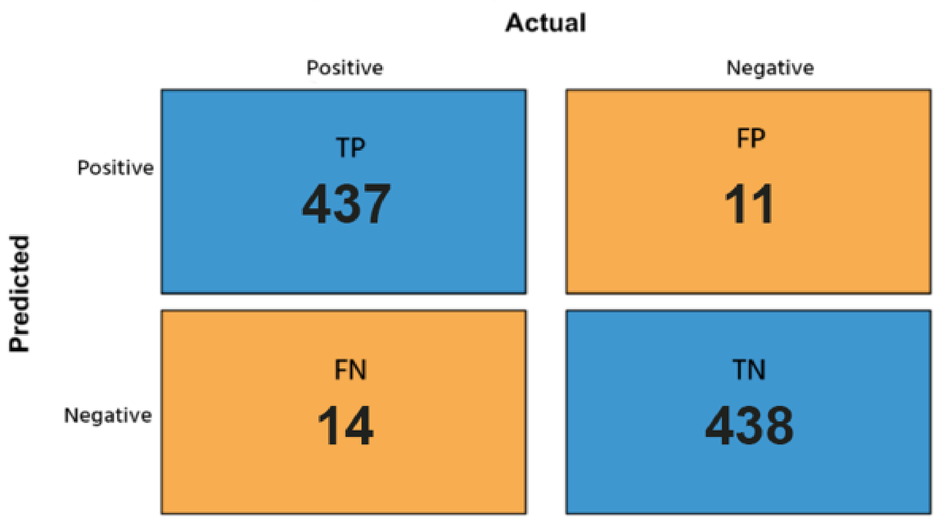
8.2.1 Results Table

TABLE- 8.1. Results Table

Metrics	Scratched Model	AlexNet	VGG	ResNet
Accuracy	96.67%	89.83%	95.03%	**97.22%**
Precision	96.65%	91.74%	94.73%	**97.54%**
Recall	97.09%	86.37%	95.21%	**97.19%**
F1-score	97.22%	88.97%	94.97%	**97.22%**

8.2.2 Confusion Matrix

TABLE- 8.2 Confusion Matrix



In the confusion matrix, there’s have some changes that we see. In 499A True Positive (TP) cases was 433 but in 499B it became 433. False Positive (FP) case was 13 now it became 11. False Negative (FN) case was 17 now it became 14. True Negative (TN) was 437 now it became 438.

8.3 Limitations

GPU or Graphics Processing Unit is now the most important thing in every CNN’s model. AlexNet was the first convolutional network which used GPU to boost performance. AlexNet allows for multi-GPU training by putting half of the model's neurons on one GPU and the other half on another GPU. Not only does this mean that a bigger model can be trained, but it also cuts down on the training time.

Though GPU is very important thing but it was our limitations. GPUs are essential part of a modern artificial intelligence infrastructure, and new GPUs have been developed and optimized specifically for deep learning. We were weak in this important thing. If we could manage this then our esearch would have been faster and our result were more accurate.

8.4 Summary

In this chapter, we have discussed about our findings of the research and analyze the results with several metrics. As we can see, the pre-trained model can detect Brain tumors 97.22% of the times correctly. But with GPU, we think, we could able to achieve much higher results than this.

Chapter 9
Conclusion

In this project, we present two Deep Learning based automatic detection system for Brain tumors. First one is the CNN architecture model, we built from scratch using PyTorch. Here, we were able to achieve an accuracy of 96.67%. To train this model, we have transformed an MRI image, then after resizing and normalizing the image, we fed the image into the training function with a batch size of 128 and learning rate of 0.001. The results were very good with the simplicity of the model architecture. Then In the second one, using transfer learning, we achieved an accuracy of 97.22%. To train this model, after transforming the MRI image, we resized, center cropped, horizontally flipped with a probability of 0.5 and normalized the image. Then we fed the image into the training function with a batch size of 32. For this phase, we have used both 0.001 and 0.0015 as the learning rate. As you can see, the results are very good and it can surely outperform some of the models that are running on industrial level today. Therefore, we can say, our model needs less computational specifications as it takes less execution time, but it is very powerful and can automatically detect the Brain tumor 97.22% of the time.

In summary, our proposed system can play a significance role in the detection of brain tumors in patient. To further boost the model efficiency and accuracy, comprehensive hyper-parameter tuning, high GPU, and a better preprocessing technique can be conceived. Our proposed system is for binary classification problems, however, in future work, the proposed method can be extended for multiclass classification problems such as identification of brain tumor types such as Glioma, Meningioma, and Pituitary or may be used to detect other brain abnormalities. Also, our proposed system can play an effective role in the early diagnosis of dangerous disease in other clinical domains related to medical imaging, particularly other cancer whose mortality rate is very high globally.

Bibliography

[1] Dhillo, Vanshika. (2020). Identification of Brain Tumor using Image Processing and Neural Networks. International Journal of Engineering Research and. V9. 10.17577/IJERTV9IS090038

[2] Nader, Dena & Jehlol, Hashem & Subhi, Anwer & Oleiwi, Abdulhussein. (2015). Brain Tumor Detection Using Shape features and Machine Learning Algorithms

[3] Díaz-Pernas FJ, Martínez-Zarzuela M, Antón-Rodríguez M, González-Ortega D. A Deep Learning Approach for Brain Tumor Classification and Segmentation Using a Multiscale Convolutional Neural Network. Healthcare. 2021; 9(2):153. https://doi.org/10.3390/healthcare9020153

[4] Kang J, Ullah Z, Gwak J. MRI-Based Brain Tumor Classification Using Ensemble of Deep Features and Machine Learning Classifiers. Sensors. 2021; 21(6):2222. https://doi.org/10.3390/s21062222

[5] J. Kang and J. Gwak, "Deep Learning-Based Brain Tumor Classification in MRI images using Ensemble of Deep Features," Journal of the Korean Society for Computer Information and Information Science, vol. 26, no. 7, pp. 37–44, Jul. 2021.

[6] Khan, H. A., Jue, A., Mushtaq, M., & Mushtaq, M. U. (2020, September 15). Brain tumor classification in MRI image using convolutional neural network. Mathematical & Bioscience Engineering. doi:10.3934/mbe.2020328

[7] He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

[8] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.

[9] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger (2018). Densely Connected Convolutional Networks. arXiv:1608.06993

[10] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. (2012). ImageNet Classification with Deep Convolutional Neural Networks.

Appendices

Appendix A
Codes for Brain Tumor Detection

A1: Code for Model Build from Scratch

```
#!/usr/bin/env python
# coding: utf-8

# Importing libraries for dataset preprocessing, which includes digitalization of data, resizing etc...

# In this first part of the notebook, the main aim is to create a model able to distinguish between
MRI images of patients affected by a brain tumor and the ones of healthy subjects. To achieve this
aim, instead of using a pre-trained model, as done in the other published notebooks, I decided to
build a simple Convolutional Neural Network from scratch. Considering that one of the greatest
limitations of Deep learning is the interpretability of the models, I wanted to see if a simpler and
more interpretable model could achieve or even overcome the results obtained in the other
notebooks with deeper models.

#Import Custom Dataset
import numpy as np
import pandas as pd
import random
import os
from torchvision.io import read_image
from torch.utils.data import Dataset

random.seed(123)

os.getcwd()

files = []
labels = []

for dt in ["yes", "no"]:
    dtfolder = os.path.join("Data", dt)
    ld = os.listdir(dtfolder)
    lab = [dt] * len(ld)
    labels.extend(lab)
    files.extend(ld)

def toInt(x):
    if x=="no":
        return 0
    else:
        return 1

intLabels = list(map(toInt, labels))

df_dict = {
    "name": files,
    "label": intLabels
}

annotations_file = pd.DataFrame(df_dict)

annotations_file.head(3)

annotations_file["label"].value_counts()

annotations_file["label"].value_counts()/3000

filestest = []
labelstest = []
for dt in ["pred"]:
    dtfolder = os.path.join("Data", dt)
    ld = os.listdir(dtfolder)
    lab = [dt] * len(ld)
    labelstest.extend(lab)
    filestest.extend(ld)

df_dict_test = {
    "name": filestest,
    "label": labelstest
}

test_file = pd.DataFrame(df_dict_test)

test_file.head(3)

import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split
import tqdm

train_set, val_set = train_test_split(annotations_file, test_size=0.30, random_state=42,
shuffle=True, stratify=annotations_file["label"])

print("Train:", len(train_set))
print("Validation:", len(val_set))

print("Ratio (ALL vs Train vs Validation)")
pd.DataFrame(data={
    "All": annotations_file["label"].value_counts()/len(annotations_file),
    "Train": train_set["label"].value_counts()/len(train_set),
    "Validation": val_set["label"].value_counts()/len(val_set)
})

# annotations_file.to_csv("annotations_file.csv", index=False, header=False)
# train_set.to_csv("train_file.csv", index=False, header=False)
# val_set.to_csv("val_file.csv", index=False, header=False)
# test_file.to_csv("test_file.csv", index=False, header=False)

os.listdir()

# device = ("cuda" if torch.cuda.is_available() else "cpu")

class BrainTumorDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file, header=None)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path, mode=torchvision.io.image.ImageReadMode.RGB)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        sample = {"image": image, "label": label}
        return sample

transformGen = transforms.Compose([
                                transforms.ToPILImage(),
                                transforms.Resize((256, 256)),
                                transforms.ToTensor()
])

train_datasetGen = BrainTumorDataset("train_file.csv", "ImageData", transform=transformGen)
val_datasetGen = BrainTumorDataset("val_file.csv", "ImageData", transform=transformGen)

sample = train_datasetGen[1]

sample

img = sample["image"]

plt.figure()
plt.imshow(img.permute(1, 2, 0))
plt.title(sample["label"])
plt.show()

from torch.utils.data import DataLoader

# Hyper params
num_class = 1
batch_size = 32
learning_rate = 0.001

train_dataloaderGen = DataLoader(train_datasetGen, batch_size=batch_size, shuffle=True)
val_dataloaderGen = DataLoader(val_datasetGen, batch_size=batch_size, shuffle=False)

class GeneralConvModel(nn.Module):
    def __init__(self, num_classes=1):
        super(GeneralConvModel, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.layer3 = nn.Sequential(
            nn.Conv2d(32, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.fc = nn.Linear(32*32*16, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        #out = torch.sigmoid(out)

        return out
```

```
modelGen = GeneralConvModel(num_classes=1)

modelGen

# define criterion, optimizer
criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.Adam(modelGen.parameters(), lr=learning_rate)


len(train_dataloader)


# write the trianing loop
def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader)
    for batch, data in enumerate(dataloader):
        #print(data)
        X = data['image']
        y = data['label']
        # Compute prediction and loss
        pred = model(X)
        y = y.unsqueeze(1)
        y = y.float()
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        loss, current = loss.item(), batch
        print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")


epochs = 10
with tqdm.notebook.tqdm(total=epochs) as pbar:
    for t in range(epochs):
        print(f"Epoch {t+1}\n-------------------------------")
        train_loop(train_dataloader, modelGen, criterion, optimizer)
        pbar.update(1)
print("Done!")

ind = 9
with torch.no_grad():
  p = model(val_dataset[ind]["image"].unsqueeze(0))
  print(p.sigmoid())
  print(val_dataset[ind]["label"])
```

```
def test_loop(model, dataloader):
    #size = len(dataloader.dataset)
    total, total_correct = 0, 0
    preds = torch.tensor([])
    gt = torch.tensor([])

    with torch.no_grad():
        for i, data in enumerate(dataloader):
            X = data["image"]
            y = data["label"]
            pred = model(X)
            pred = pred.sigmoid()
            pred = pred > 0.5
            pred = pred.squeeze()
            total += len(pred)
            correct = (pred == y).sum()
            total_correct += correct

            preds = torch.cat((preds, pred), dim=0)
            gt = torch.cat((gt, y), dim=0)

    print(f"Accuracy: {(100*total_correct)/total}%")
    return {"pred": preds, "gt": gt}


print("***Validation Accuracy***")
modelGen_score = test_loop(modelGen, val_dataloader)

from sklearn.metrics import f1_score, confusion_matrix, precision_score, recall_score,
accuracy_score

def scores(y):
    print("Accuracy : " + str(accuracy_score(y["gt"], y["pred"]).round(4)*100) + "%")
    print("F1-score : " + str(f1_score(y["gt"], y["pred"]).round(4)*100) + "%")
    print("Precision: " + str(precision_score(y["gt"], y["pred"]).round(4)*100) + "%")
    print("Recall   : " + str(recall_score(y["gt"], y["pred"]).round(4)*100) + "%")

scores(modelGen_score)

confusion_matrix(modelGen_score["gt"], modelGen_score["pred"])

tn, fp, fn, tp = confusion_matrix(modelGen_score["gt"], modelGen_score["pred"]).ravel()

(tn, fp, fn, tp)

def plotcm(cm, annot=True, cp="YlGn"):
    fig = plt.figure(dpi=100)
    ax = fig.add_subplot(111)
    cax = ax.matshow(cm, cmap=cp)
    fig.colorbar(cax)
    labels = ["Negative", "Positive"]

    ax.set_xticklabels([''] + labels)
    ax.set_yticklabels([''] + labels)

    if annot:
        for i in range(len(cm)):
            for j in range(len(cm)):
                text = ax.text(j, i, cm[i, j],
                               ha="center", va="center", color="k")

    ax.set_title("***Confusion Matrix***")
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')
    plt.tight_layout()
    plt.show()

plotcm(confusion_matrix(modelGen_score["gt"], modelGen_score["pred"]))

# torch.save(modelGen.state_dict(), 'modelGen_weights.pth')
# torch.save(modelGen, 'modelGen.pth')

AlexNet = torch.hub.load('pytorch/vision:v0.9.0', 'alexnet', pretrained=False)

transformAlex = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

train_dataset_Alex = BrainTumorDataset("train_file.csv", "ImageData", transform=transformAlex)
val_dataset_Alex = BrainTumorDataset("val_file.csv", "ImageData", transform=transformAlex)

# define criterion, optimizer
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(AlexNet.parameters(), lr=learning_rate)

train_dataloaderAlex = DataLoader(train_dataset_Alex, batch_size=batch_size, shuffle=True)
val_dataloaderAlex = DataLoader(val_dataset_Alex, batch_size=batch_size, shuffle=False)
```

A2: Code for Pre-trained Models

```
# -*- coding: utf-8 -*-
```

```
"""Brain Tumor Detection.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1i5EQ1OSpeoiSRKN5rPa5SqsNFjPT-GzF
"""

#Import Custom Dataset
import numpy as np
import pandas as pd
import random
import os
import shutil
from torchvision.io import read_image
from torch.utils.data import Dataset
from torch.utils.data import DataLoader

random.seed(123)

from google.colab import drive
drive.mount('/content/drive')

os.chdir("/content/drive/MyDrive/BrainTumorData")

files = []
labels = []

for dt in ["yes", "no"]:
    dtfolder = os.path.join("Data", dt)
    ld = os.listdir(dtfolder)
    lab = [dt] * len(ld)
    labels.extend(lab)
    files.extend(ld)

def toInt(x):
    if x=="no":
        return 0
    else:
        return 1

intLabels = list(map(toInt, labels))

df_dict = {
    "name": files,
    "label": intLabels
}
```

```
annotations_file = pd.DataFrame(df_dict)

annotations_file.head()

annotations_file["label"].value_counts()

annotations_file["label"].value_counts()/306

import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split

train_val_set, test_set = train_test_split(annotations_file, test_size=0.33, random_state=42,
shuffle=True, stratify=annotations_file["label"])

train_set, val_set = train_test_split(train_val_set, test_size=0.20, random_state=42, shuffle=True,
stratify=train_val_set["label"])

print("Train:", len(train_set))
print("Validation:", len(val_set))
print("Test:", len(test_set))

print("Printing Ratio (ALL vs Train vs Validation vs Test)")
pd.DataFrame(data={
    "All": annotations_file["label"].value_counts()/len(annotations_file),
    "Train": train_set["label"].value_counts()/len(train_set),
    "Validation": val_set["label"].value_counts()/len(val_set),
    "Test": test_set["label"].value_counts()/len(test_set)
})

# annotations_file.to_csv("annotations_file.csv", index=False, header=False)
# train_set.to_csv("train_file.csv", index=False, header=False)
# val_set.to_csv("val_file.csv", index=False, header=False)
# test_set.to_csv("test_file.csv", index=False, header=False)

folders = os.listdir("Data")

folders

# os.mkdir("ImageData")
# target_dir = 'ImageData'

# for i in folders:
#   source_dir = os.path.join("Data", i)
#   print("...On", source_dir, "Directory...")
#   file_names = os.listdir(source_dir)

#   for file_name in file_names:
#     shutil.copy(os.path.join(source_dir, file_name), target_dir)

#   print("Copied From ->", source_dir, "To ->", target_dir)

len(os.listdir("ImageData"))

class BrainTumorDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=False):
        self.img_labels = pd.read_csv(annotations_file, header=None)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path, mode=torchvision.io.image.ImageReadMode.RGB)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = torch.nn.functional.one_hot(torch.tensor(label), num_classes=4)
        sample = {"image": image, "label": label}
        return sample

torch.nn.functional.one_hot(torch.tensor([0]), num_classes=4)

transform = transforms.Compose([
                                transforms.ToPILImage(),
                                transforms.Resize((256, 256)),
                                transforms.ToTensor()
])

os.listdir()

train_dataset = BrainTumorDataset("train_file.csv", "ImageData", transform=transform)
val_dataset = BrainTumorDataset("val_file.csv", "ImageData", transform=transform)
test_dataset = BrainTumorDataset("test_file.csv", "ImageData", transform=transform)

sample = train_dataset[0]

len(train_dataset)
```

```
sample

img = sample["image"]

plt.figure()
plt.imshow(img.permute(1, 2, 0))
plt.show()

# mean calculation
# torch.sum(img, dim=[1, 2])/(256*256)

# Hyper params
num_class = 4
batch_size = 64
learning_rate = 0.001

train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

import tqdm

def get_mean_std(loader):
    # var[X] = E[X**2] - E[X]**2
    channels_sum, channels_sqrd_sum, num_batches = 0, 0, 0
    with tqdm.notebook.tqdm(total=164) as pbar:
      for batch, data in enumerate(loader):
          data = data["image"]
          channels_sum += torch.mean(data, dim=[0, 2, 3])
          channels_sqrd_sum += torch.mean(data ** 2, dim=[0, 2, 3])
          num_batches += 1
          pbar.update(len(data))

    mean = channels_sum / num_batches
    std = (channels_sqrd_sum / num_batches - mean ** 2) ** 0.5

    return mean, std

mean, std = get_mean_std(train_dataloader)

print("Mean of Training Dataset:-----------------", mean)
print("Standard Deviation of Training Dataset:---", std)

device = ("cuda" if torch.cuda.is_available() else "cpu")

class GeneralConvModel(nn.Module):
    def __init__(self, num_classes=1):
        super(GeneralConvModel, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.layer3 = nn.Sequential(
            nn.Conv2d(32, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.fc = nn.Linear(32*32*16, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        #out = torch.sigmoid(out)

        return out

model = GeneralConvModel(num_classes=num_class).to(device)

model

# define criterion, optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# write the trianing loop
def train_loop(dataloader, model, loss_fn, optimizer, Normalize=False, mean=0, std=1):
    size = len(dataloader)
    for batch, data in enumerate(dataloader):
        #print(data)
        X = data['image']
        y = data['label']

        # Normalize image with Dataset mean and std
        if Normalize:
          transforms.Normalize(mean, std)(X)

        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        loss, current = loss.item(), batch
        print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

# for i, j in enumerate(train_dataloader):
#   data = j["image"]
#   break

# len(transforms.Normalize(mean, std)(data))

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, model, criterion, optimizer, True, mean, std)
print("Done!")

def test_loop(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    test_loss, correct = 0, 0

    with torch.no_grad():
        for i, data in enumerate(dataloader):
            X = data["image"]
            y = data["label"]
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= size
    correct /= size
    print(f"Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")

test_loop(test_dataloader, model, criterion)

transformAlex = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std),
])

class AlexNet(nn.Module):

    def __init__(self, num_classes: int = 4) -> None:
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

AlexNet_model = AlexNet()

AlexNet_model

train_dataset_Alex = BrainTumorDataset("train_file.csv", "ImageData", transform=transformAlex)
val_dataset_Alex = BrainTumorDataset("val_file.csv", "ImageData", transform=transformAlex)
test_dataset_Alex = BrainTumorDataset("test_file.csv", "ImageData", transform=transformAlex)
```

```
train_dataloader_Alex = DataLoader(train_dataset_Alex, batch_size=batch_size, shuffle=True)
val_dataloader_Alex = DataLoader(val_dataset_Alex, batch_size=batch_size, shuffle=False)
test_dataloader_Alex = DataLoader(test_dataset_Alex, batch_size=batch_size, shuffle=False)

# define criterion, optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(AlexNet_model.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader_Alex, AlexNet_model, criterion, optimizer, True, mean, std)
print("Done!")

test_loop(train_dataloader_Alex, AlexNet_model, criterion)
test_loop(val_dataloader_Alex, AlexNet_model, criterion)
test_loop(test_dataloader_Alex, AlexNet_model, criterion)

transformAlexDefault = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

AlexNet_Default = AlexNet()

train_dataset_AlexD = BrainTumorDataset("train_file.csv", "ImageData",
transform=transformAlexDefault)
val_dataset_AlexD = BrainTumorDataset("val_file.csv", "ImageData",
transform=transformAlexDefault)
test_dataset_AlexD = BrainTumorDataset("test_file.csv", "ImageData",
transform=transformAlexDefault)

train_dataloader_AlexD = DataLoader(train_dataset_AlexD, batch_size=batch_size, shuffle=True)
val_dataloader_AlexD = DataLoader(val_dataset_AlexD, batch_size=batch_size, shuffle=False)
test_dataloader_AlexD = DataLoader(test_dataset_AlexD, batch_size=batch_size, shuffle=False)

# define criterion, optimizer
optimizer = torch.optim.Adam(AlexNet_Default.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader_AlexD, AlexNet_Default, criterion, optimizer)
print("Done!")
```

```
test_loop(train_dataloader_AlexD, AlexNet_Default, criterion)
test_loop(val_dataloader_AlexD, AlexNet_Default, criterion)
test_loop(test_dataloader_AlexD, AlexNet_Default, criterion)

preprocess = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

densenet121 = torch.hub.load('pytorch/vision:v0.9.0', 'densenet121', pretrained=True)

densenet121

densenet121.classifier.out_features = num_class

densenet121

train_dataset_Dense121 = BrainTumorDataset("train_file.csv", "ImageData",
transform=preprocess)
val_dataset_Dense121 = BrainTumorDataset("val_file.csv", "ImageData", transform=preprocess)
test_dataset_Dense121 = BrainTumorDataset("test_file.csv", "ImageData", transform=preprocess)

train_dataloader_Dense121 = DataLoader(train_dataset_Dense121, batch_size=batch_size,
shuffle=True)
val_dataloader_Dense121 = DataLoader(val_dataset_Dense121, batch_size=batch_size,
shuffle=False)
test_dataloader_Dense121 = DataLoader(test_dataset_Dense121, batch_size=batch_size,
shuffle=False)

# define criterion, optimizer
optimizer = torch.optim.Adam(densenet121.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader_Dense121, densenet121, criterion, optimizer)
print("Done!")

print("***Train***")
test_loop(train_dataloader_Dense121, densenet121, criterion)
print("***Validation***")
test_loop(val_dataloader_Dense121, densenet121, criterion)
print("***Test***")
test_loop(test_dataloader_Dense121, densenet121, criterion)
```

```
densenet169 = torch.hub.load('pytorch/vision:v0.9.0', 'densenet169', pretrained=True)

densenet169.classifier.out_features = num_class

train_dataset_densenet169 = BrainTumorDataset("train_file.csv", "ImageData",
transform=preprocess)
val_dataset_densenet169 = BrainTumorDataset("val_file.csv", "ImageData", transform=preprocess)
test_dataset_densenet169 = BrainTumorDataset("test_file.csv", "ImageData",
transform=preprocess)

train_dataloader_densenet169 = DataLoader(train_dataset_densenet169, batch_size=batch_size,
shuffle=True)
val_dataloader_densenet169 = DataLoader(val_dataset_densenet169, batch_size=batch_size,
shuffle=False)
test_dataloader_densenet169 = DataLoader(test_dataset_densenet169, batch_size=batch_size,
shuffle=False)

# define criterion, optimizer
optimizer = torch.optim.Adam(densenet169.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader_densenet169, densenet169, criterion, optimizer)
print("Done!")

print("***Train***")
test_loop(train_dataloader_densenet169, densenet169, criterion)
print("***Validation***")
test_loop(val_dataloader_densenet169, densenet169, criterion)
print("***Test***")
test_loop(test_dataloader_densenet169, densenet169, criterion)

def my_loss(output, target):
  correct = 0
  total = 0
  _, predicted = torch.max(output, 1)
  total += len(output)
  correct += (predicted == target).sum().item()
  print("Accuracy:", 100*correct/total)
  return torch.tensor((1.0-(correct / total)), requires_grad=True)

train_dataset_densenet169l = BrainTumorDataset("train_file.csv", "ImageData",
transform=preprocess)
val_dataset_densenet169l = BrainTumorDataset("val_file.csv", "ImageData",
transform=preprocess)
```

```
test_dataset_densenet169l = BrainTumorDataset("test_file.csv", "ImageData",
transform=preprocess)

train_dataloader_densenet169l = DataLoader(train_dataset_densenet169l, batch_size=batch_size,
shuffle=True)
val_dataloader_densenet169l = DataLoader(val_dataset_densenet169l, batch_size=batch_size,
shuffle=False)
test_dataloader_densenet169l = DataLoader(test_dataset_densenet169l, batch_size=batch_size,
shuffle=False)

densenet169l = torch.hub.load('pytorch/vision:v0.9.0', 'densenet169', pretrained=True)

# define criterion, optimizer
optimizer = torch.optim.Adam(densenet169l.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader_densenet169l, densenet169l, my_loss, optimizer)
print("Done!")

print("***Train***")
test_loop(train_dataloader_densenet169, densenet169, my_loss)
print("***Validation***")
test_loop(val_dataloader_densenet169, densenet169, my_loss)
print("***Test***")
test_loop(test_dataloader_densenet169, densenet169, my_loss)



ResNet50 = torchvision.models.resnet50(pretrained=True, progress=True)

ResNet50.fc = nn.Linear(2048, num_class)

ResNet50

transformResNet = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

train_dataset_ResNet = BrainTumorDataset("train_file.csv", "ImageData",
transform=transformResNet)
val_dataset_ResNet = BrainTumorDataset("val_file.csv", "ImageData",
transform=transformResNet)
```

```
test_dataset_ResNet = BrainTumorDataset("test_file.csv", "ImageData",
transform=transformResNet)

train_dataloader_ResNet = DataLoader(train_dataset_ResNet, batch_size=batch_size,
shuffle=True)
val_dataloader_ResNet = DataLoader(val_dataset_ResNet, batch_size=batch_size, shuffle=False)
test_dataloader_ResNet = DataLoader(test_dataset_ResNet, batch_size=batch_size, shuffle=False)



# write the trianing loop GPU version
def train_loop(dataloader, model, loss_fn, optimizer, Normalize=False, mean=0, std=1):
    size = len(dataloader)
    for batch, data in enumerate(dataloader):
        #print(data)
        X = data['image'].to(device)
        y = data['label'].to(device)

        # Normalize image with Dataset mean and std
        if Normalize:
          transforms.Normalize(mean, std)(X)

        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        loss, current = loss.item(), batch
        print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

# write the testing loop GPU version
def test_loop(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    test_loss, correct = 0, 0

    with torch.no_grad():
        for i, data in enumerate(dataloader):
            X = data["image"].to(device)
            y = data["label"].to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= size
    correct /= size
    print(f"Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")

if torch.cuda.is_available():
  ResNet50 = ResNet50.to(device)

# define criterion, optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(ResNet50.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader_ResNet, ResNet50, criterion, optimizer)
print("Done!")

print("***Train***")
test_loop(train_dataloader_ResNet, ResNet50, criterion)
print("***Validation***")
test_loop(val_dataloader_ResNet, ResNet50, criterion)
print("***Test***")
test_loop(test_dataloader_ResNet, ResNet50, criterion)

epochs = 30
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader_ResNet, ResNet50, criterion, optimizer)
print("Done!")

print("***Train***")
test_loop(train_dataloader_ResNet, ResNet50, criterion)
print("***Validation***")
test_loop(val_dataloader_ResNet, ResNet50, criterion)
print("***Test***")
test_loop(test_dataloader_ResNet, ResNet50, criterion)



ResNet152 = torchvision.models.resnet152(pretrained=True, progress=True)

ResNet152.fc = nn.Linear(2048, num_class)

ResNet152

transformResNet = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

train_dataset_ResNet = BrainTumorDataset("train_file.csv", "ImageData",
transform=transformResNet)
val_dataset_ResNet = BrainTumorDataset("val_file.csv", "ImageData",
transform=transformResNet)
test_dataset_ResNet = BrainTumorDataset("test_file.csv", "ImageData",
transform=transformResNet)

train_dataloader_ResNet = DataLoader(train_dataset_ResNet, batch_size=batch_size,
shuffle=True)
val_dataloader_ResNet = DataLoader(val_dataset_ResNet, batch_size=batch_size, shuffle=False)
test_dataloader_ResNet = DataLoader(test_dataset_ResNet, batch_size=batch_size, shuffle=False)

if torch.cuda.is_available():
  ResNet152 = ResNet152.to(device)

# define criterion, optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(ResNet152.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader_ResNet, ResNet152, criterion, optimizer)
print("Done!")

print("***Train***")
test_loop(train_dataloader_ResNet, ResNet50, criterion)
print("***Validation***")
test_loop(val_dataloader_ResNet, ResNet50, criterion)
print("***Test***")
test_loop(test_dataloader_ResNet, ResNet50, criterion)

ResNet152 = torchvision.models.resnet152(pretrained=True, progress=True)

ResNet152

modules=list(ResNet152.children())[:-1]
ResNet152=nn.Sequential(*modules)
for p in ResNet152.parameters():
    p.requires_grad = False

ResNet152

with torch.no_grad():
  X = sample["image"]
  X = X.unsqueeze(0)
  pred = ResNet152(X)

len(pred)

pred.shape

pred

a = torch.reshape(pred, (1, 2048))

a

arr = np.array([])

df = pd.DataFrame(a)

df.head()

pd.concat([df, df, df])

preprocess = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

train_dataset = BrainTumorDataset("train_file.csv", "ImageData", transform=preprocess)
val_dataset = BrainTumorDataset("val_file.csv", "ImageData", transform=preprocess)
test_dataset = BrainTumorDataset("test_file.csv", "ImageData", transform=preprocess)

ResNet152 = torchvision.models.resnet152(pretrained=True, progress=True)
modules=list(ResNet152.children())[:-1]
ResNet152=nn.Sequential(*modules)
for p in ResNet152.parameters():
    p.requires_grad = False

data = []
label = []
with torch.no_grad():
  for i in train_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = ResNet152(X)
    a = torch.reshape(pred, (1, 2048))
    a = a.numpy()[0]
    data.append(a)
    label.append(y)

print(len(data))
print(len(label))

df = pd.DataFrame(data)

df.head()

df["y"] = label

df.head()

df.drop("y", axis = 1).head()

X_train, y_train = data, label

from sklearn import svm

ResNet152svm = svm.SVC(decision_function_shape='ovo')
ResNet152svm.fit(df.drop("y", axis = 1), df["y"])
preds = ResNet152svm.predict(df.drop("y", axis = 1))

preds

ResNet152svm.classes_

from sklearn.multiclass import OneVsOneClassifier
from sklearn.multiclass import OneVsRestClassifier

ovo_clf = OneVsOneClassifier(svm.SVC(random_state=42))
ovr_clf = OneVsRestClassifier(svm.SVC(random_state=42))

ovo_clf.fit(df.drop("y", axis = 1), df["y"])
ovr_clf.fit(df.drop("y", axis = 1), df["y"])
predo = ovo_clf.predict(df.drop("y", axis = 1))
predr = ovr_clf.predict(df.drop("y", axis = 1))

predo

predr

from sklearn import metrics

metrics.accuracy_score(df["y"], predr)
```

```
vdata = []
vlabel = []
with torch.no_grad():
  for i in val_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = ResNet152(X)
    a = torch.reshape(pred, (1, 2048))
    a = a.numpy()[0]
    vdata.append(a)
    vlabel.append(y)

tdata = []
tlabel = []
with torch.no_grad():
  for i in test_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = ResNet152(X)
    a = torch.reshape(pred, (1, 2048))
    a = a.numpy()[0]
    tdata.append(a)
    tlabel.append(y)

vdf = pd.DataFrame(vdata)
tdf = pd.DataFrame(tdata)

vdf["y"] = vlabel
tdf["y"] = tlabel

vpredr = ovr_clf.predict(vdf.drop("y", axis = 1))
tpredr = ovr_clf.predict(tdf.drop("y", axis = 1))

print("Train: " + str(metrics.accuracy_score(df["y"], predr)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredr)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredr)*100))

from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(max_depth=3, random_state=0, max_leaf_nodes = 8)

clf.fit(df.drop("y", axis = 1), df["y"])

predrf = clf.predict(df.drop("y", axis = 1))
vpredrf = clf.predict(vdf.drop("y", axis = 1))
```

```
tpredrf = clf.predict(tdf.drop("y", axis = 1))

print("Train: " + str(metrics.accuracy_score(df["y"], predrf)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredrf)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredrf)*100))



vgg16 = torchvision.models.vgg16(pretrained=True, progress=True)
modules=list(vgg16.children())[:-1]
vgg16=nn.Sequential(*modules)
for p in vgg16.parameters():
    p.requires_grad = False

vgg16

data = []
label = []
with torch.no_grad():
  for i in train_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = vgg16(X)
    a = torch.reshape(pred, (1, 25088))
    a = a.numpy()[0]
    data.append(a)
    label.append(y)



df = pd.DataFrame(data)
df["y"] = label



vdata = []
vlabel = []
with torch.no_grad():
  for i in val_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = vgg16(X)
    a = torch.reshape(pred, (1, 25088))
    a = a.numpy()[0]
    vdata.append(a)
    vlabel.append(y)
```

```
tdata = []
tlabel = []
with torch.no_grad():
  for i in test_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = vgg16(X)
    a = torch.reshape(pred, (1, 25088))
    a = a.numpy()[0]
    tdata.append(a)
    tlabel.append(y)

vdf = pd.DataFrame(vdata)
tdf = pd.DataFrame(tdata)

vdf["y"] = vlabel
tdf["y"] = tlabel

ovr_clf = OneVsRestClassifier(svm.SVC(random_state=42))
ovr_clf.fit(df.drop("y", axis = 1), df["y"])

predr = ovr_clf.predict(df.drop("y", axis = 1))
vpredr = ovr_clf.predict(vdf.drop("y", axis = 1))
tpredr = ovr_clf.predict(tdf.drop("y", axis = 1))

clf = RandomForestClassifier(max_depth=6, random_state=0, max_leaf_nodes = 12)
clf.fit(df.drop("y", axis = 1), df["y"])

predrf = clf.predict(df.drop("y", axis = 1))
vpredrf = clf.predict(vdf.drop("y", axis = 1))
tpredrf = clf.predict(tdf.drop("y", axis = 1))

print("*****SVM*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predr)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredr)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredr)*100))

print("\n*****RandomForest*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predrf)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredrf)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredrf)*100))

metrics.confusion_matrix(tdf["y"], tpredr)

import xgboost as xgb
```

```
clfxgb = xgb.XGBClassifier()
clfxgb.fit(df.drop("y", axis = 1), df["y"])

predxgb = clfxgb.predict(df.drop("y", axis = 1))
vpredxgb = clfxgb.predict(vdf.drop("y", axis = 1))
tpredxgb = clfxgb.predict(tdf.drop("y", axis = 1))

clfxgbrf = xgb.XGBRFClassifier()
clfxgbrf.fit(df.drop("y", axis = 1), df["y"])

predxgbrf = clfxgbrf.predict(df.drop("y", axis = 1))
vpredxgbrf = clfxgbrf.predict(vdf.drop("y", axis = 1))
tpredxgbrf = clfxgbrf.predict(tdf.drop("y", axis = 1))

print("*****SVM*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predxgb)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredxgb)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredxgb)*100))

print("\n*****RandomForest*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predxgbrf)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredxgbrf)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredxgbrf)*100))

#sklearn warm start



VGG16 = torchvision.models.vgg16(pretrained=True, progress=True)

VGG16

VGG16.classifier[6] = nn.Linear(4096, num_class)

VGG16

preprocessing = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

train_dataset = BrainTumorDataset("train_file.csv", "ImageData", transform=preprocessing)
val_dataset = BrainTumorDataset("val_file.csv", "ImageData", transform=preprocessing)
test_dataset = BrainTumorDataset("test_file.csv", "ImageData", transform=preprocessing)
```

```
train_dataloader = DataLoader(train_dataset_ResNet, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset_ResNet, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset_ResNet, batch_size=batch_size, shuffle=False)

if torch.cuda.is_available():
  VGG16 = VGG16.to(device)

learning_rate

# define criterion, optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(VGG16.parameters(), lr=0.0015)

epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, VGG16, criterion, optimizer)
print("Done!")

list(VGG16.children())

VGG16.classifier = nn.Linear(25088, 4096)

VGG16

# modules=list(VGG16.children())[:-1]
# VGG16=nn.Sequential(*modules)

for p in VGG16.parameters():
    p.requires_grad = False

data = []
label = []
with torch.no_grad():
  for i in train_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = VGG16(X)
    a = torch.reshape(pred, (1, 4096))
    a = a.numpy()[0]
    data.append(a)
    label.append(y)

df = pd.DataFrame(data)

df["y"] = label
```

```
df.head()



vdata = []
vlabel = []
with torch.no_grad():
  for i in val_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = VGG16(X)
    a = torch.reshape(pred, (1, 4096))
    a = a.numpy()[0]
    vdata.append(a)
    vlabel.append(y)

tdata = []
tlabel = []
with torch.no_grad():
  for i in test_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = VGG16(X)
    a = torch.reshape(pred, (1, 4096))
    a = a.numpy()[0]
    tdata.append(a)
    tlabel.append(y)

vdf = pd.DataFrame(vdata)
tdf = pd.DataFrame(tdata)

vdf["y"] = vlabel
tdf["y"] = tlabel

ovr_clf = OneVsRestClassifier(svm.SVC(random_state=42))
ovr_clf.fit(df.drop("y", axis = 1), df["y"])

predr = ovr_clf.predict(df.drop("y", axis = 1))
vpredr = ovr_clf.predict(vdf.drop("y", axis = 1))
tpredr = ovr_clf.predict(tdf.drop("y", axis = 1))

clf = RandomForestClassifier(max_depth=3, random_state=0)
clf.fit(df.drop("y", axis = 1), df["y"])

predrf = clf.predict(df.drop("y", axis = 1))
```

```
vpredrf = clf.predict(vdf.drop("y", axis = 1))
tpredrf = clf.predict(tdf.drop("y", axis = 1))

clfxgb = xgb.XGBClassifier()
clfxgb.fit(df.drop("y", axis = 1), df["y"])

predxgb = clfxgb.predict(df.drop("y", axis = 1))
vpredxgb = clfxgb.predict(vdf.drop("y", axis = 1))
tpredxgb = clfxgb.predict(tdf.drop("y", axis = 1))

print("*****SVM*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predr)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredr)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredr)*100))

print("\n*****RandomForest*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predrf)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredrf)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredrf)*100))

print("\n*****XGBoost*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predxgb)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredxgb)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredxgb)*100))



DN169 = torchvision.models.densenet169(pretrained=True, progress=True)

DN169

# VGG16.classifier = nn.Linear(4096, num_class)

DN169

preprocessing = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

train_dataset = BrainTumorDataset("train_file.csv", "ImageData", transform=preprocessing)
val_dataset = BrainTumorDataset("val_file.csv", "ImageData", transform=preprocessing)
test_dataset = BrainTumorDataset("test_file.csv", "ImageData", transform=preprocessing)
```

```
train_dataloader = DataLoader(train_dataset_ResNet, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset_ResNet, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset_ResNet, batch_size=batch_size, shuffle=False)

if torch.cuda.is_available():
  DN169 = DN169.to(device)

# define criterion, optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(DN169.parameters(), lr=0.001)

epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, VGG16, criterion, optimizer)
print("Done!")

# modules=list(DN169.children())[:-1]
# DN169=nn.Sequential(*modules)

for p in DN169.parameters():
    p.requires_grad = False

DN169

with torch.no_grad():
  print(DN169(train_dataset[1]["image"].unsqueeze(0)).shape)

data = []
label = []
with torch.no_grad():
  for i in train_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = DN169(X)
    a = pred
    # a = torch.reshape(pred, (1, 81536))
    a = a.numpy()[0]
    data.append(a)
    label.append(y)

df = pd.DataFrame(data)

df["y"] = label

df.head()
```

```
vdata = []
vlabel = []
with torch.no_grad():
  for i in val_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = DN169(X)
    a = pred
    # a = torch.reshape(pred, (1, 81536))
    a = a.numpy()[0]
    vdata.append(a)
    vlabel.append(y)

tdata = []
tlabel = []
with torch.no_grad():
  for i in test_dataset:
    X = i["image"]
    y = i["label"]
    X = X.unsqueeze(0)
    pred = DN169(X)
    a = pred
    # a = torch.reshape(pred, (1, 81536))
    a = a.numpy()[0]
    tdata.append(a)
    tlabel.append(y)

vdf = pd.DataFrame(vdata)
tdf = pd.DataFrame(tdata)

vdf["y"] = vlabel
tdf["y"] = tlabel

ovr_clf = OneVsRestClassifier(svm.SVC(random_state=42))
ovr_clf.fit(df.drop("y", axis = 1), df["y"])

predr = ovr_clf.predict(df.drop("y", axis = 1))
vpredr = ovr_clf.predict(vdf.drop("y", axis = 1))
tpredr = ovr_clf.predict(tdf.drop("y", axis = 1))

clf = RandomForestClassifier(max_depth=3, random_state=42)
clf.fit(df.drop("y", axis = 1), df["y"])

predrf = clf.predict(df.drop("y", axis = 1))
vpredrf = clf.predict(vdf.drop("y", axis = 1))
tpredrf = clf.predict(tdf.drop("y", axis = 1))
```

```
clfxgb = xgb.XGBClassifier()
clfxgb.fit(df.drop("y", axis = 1), df["y"])

predxgb = clfxgb.predict(df.drop("y", axis = 1))
vpredxgb = clfxgb.predict(vdf.drop("y", axis = 1))
tpredxgb = clfxgb.predict(tdf.drop("y", axis = 1))

print("*****SVM*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predr)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredr)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredr)*100))

print("\n*****RandomForest*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predrf)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredrf)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredrf)*100))

print("\n*****XGBoost*****")
print("Train: " + str(metrics.accuracy_score(df["y"], predxgb)*100))
print("Validation: " + str(metrics.accuracy_score(vdf["y"], vpredxgb)*100))
print("Test: " + str(metrics.accuracy_score(tdf["y"], tpredxgb)*100))
```