Department of Electrical & Computer Engineering

ECE-9063 DATA ANALYTICS & FOUNDATION

Instructor: Dr. Katarina Grolinger

Submitted by:

Name	Email	Student Id
Shivam Rana	srana59@uwo.ca	251303122
Nishant Bhimra	nbhimra@uwo.ca	251324730
Uponika Burman Roy	ubarmanr@uwo.ca	251299020
Harshit Kohli	hkohli4@uwo.ca	251310106
Jaspreet Singh Bharaj	jbharaj2@uwo.ca	251323547

Table of Contents

1. Abstract 3
2. Introduction 3
3. Background 4
3.1 Convolutional Neural Network 4
3.2 Transfer Learning 7
4. Methodology 8
4.1 Importing the necessary libraries 8
4.2 Reading the data 9
4.3 Exploring the dataset 9
4.4 Data Pre-processing: Normalization 9
4.5 Training-Validation data split 9
4.6 Training with CNN 10
4.6.1 Base Model 10
4.6.2 Tuned model 1 (Effect of tweaking learning rates and epochs) 11
4.6.3 Tuned model 2 (Effect of changing model topology- filters, strides, drop out layers, batch size, pool_size) 11
4.6.4 Tuned model 3: Training with Transfer Learning model (VGG-16) 12
5. Evaluations/ Results 12
6. References 16

1. Abstract

One of the most difficult tasks in medical image analysis is tumor segmentation, the most prominent research in the field of medical image analysis focuses on brain tumors. Detection of a Brain Tumor at an early stage can help in increasing the life span of the patient. The manual analysis of large number of MRI scanned images become tedious and involves human errors thereby it is important to introduce a cognitive analysis of the images for faster and robust results. In this project, we propose a brain tumor classification method for Magnetic Resonance Images scans. We are detecting tumor by using preprocessing, segmentation, feature extraction, optimization and lastly classification after that preprocessed image use to classify the tumors. The Convolution Neural Network learns on the MRI scanned images through their features and successfully classifies it into four distinct categories. The concept of Transfer Learning with a Visual Geometry Group (VGG 16) model is also implemented on the images for further analysis.

Keywords: Machine Learning, Deep Learning, CNN, VGG-16

2. Introduction

A Brain tumour is considered as one of the aggressive diseases, among children and adults. Brain tumours account for 85 to 90 percent of all primary Central Nervous System (CNS) tumours. Every year, around 11,700 people are diagnosed with a brain tumour. The 5-year survival rate for people with a cancerous brain or CNS tumour is approximately 34 percent for men and 36 percent for women. Brain Tumours are classified as: Glioma Tumour, Meningioma Tumour, Pituitary Tumour, Benign Tumour, etc.

1. **Glioma** – Glioma is a type of tumor that occurs in the brain and spinal cord. Gliomas begin in the gluey supportive cells (glial cells) that surround nerve cells and help them function.[4]

2. **Meningioma** – A Meningioma is a tumor that arises from the meninges (The membrane), that surround the brain and spinal cord. Although it is not technically a brain tumor, but it is included in this category because it may compress or squeeze the adjacent brain, nerves, and vessels. Meningioma is the most common type of tumor that forms in the head.[5]

3. **No tumor** – This class denotes the patients have no Tumor.

4. **Pituitary** – Pituitary tumors are abnormal growths that develop in your pituitary gland. Some pituitary tumors result in too much of the hormones that regulate important functions of your body. Some pituitary tumors can cause your pituitary gland to produce lower levels of hormones.[6]

Proper treatment, planning, and accurate diagnostics should be implemented to improve the life expectancy of the patients. The best technique to detect brain tumours is Magnetic Resonance Imaging (MRI). A huge amount of image data is generated through the scans. These images are examined by the radiologist. A manual examination can be error-prone due to the level of complexities involved in brain tumours and their properties.

In this problem statement, we are trying to propose a system for detecting brain tumours by using Deep Learning Algorithms like Convolutional Neural Network (CNN) and Transfer Learning. Application of automated classification techniques using Machine Learning (ML) and Artificial Intelligence (AI) has consistently shown higher accuracy with faster results than manual classification.

Brain Tumours are complex. There are a lot of abnormalities in the sizes and location of the brain tumour(s). This makes it difficult for complete understanding of the nature of the tumour. Also, a professional Neurosurgeon is required for MRI analysis. More often in developing countries the lack of skilful doctors and lack of knowledge about tumours makes it challenging and time-consuming to

generate reports from MRI. Thus, an automated system can solve this problem and would be of great help to the doctors all around the world.

To solve this problem statement, the initial step was to collect the dataset from Kaggle [1] and then to understand the nature of the data statistically. In the following steps, the data is explored and checked for the consistency of data in each category. The data is found balanced and therefore it goes to the next step of refinement where the size of the images is set at a certain value as required by the model to classify. After data exploration, the data is split into training and validation set while preparing it for model training. The normalization of data is performed to scale the images at a standard uniform image size so that it is easier for the CNN model to learn on the image features. A sequential model is designed with combinations of convoluted layers and max pooling layers to represent the CNN topology. To compile the model, the 'Adam' optimizer is used where the loss is specified as 'categorical_crossentropy' and the 'accuracy' as the performance metric of the model. The model is now fitted for training followed by prediction on test data and generation of the classification report to evaluate the model performance. To improvise the model performance, it goes through two stages of tuning where the strategies followed were hyperparameter tuning like epoch, learning rate, batch size and number of filters, strides, kernel size, padding, number of Dense layers, model topology modification and data augmentation. Based on the observations, it is inferred that the tuned model performed better than the base model. To acquire higher accuracy, we performed transfer learning on the same data with VGG-16 model. This deep convolution network performed best compared to all the model's performance and has advantage of faster training as this model have pre-trained parameters.

In the next section, a detailed study on the background of the models used like Convolution Neural Network (CNN) and VGG-16. The fourth section consists of methodology which describes the approach used to perform the brain tumour classification. In the fifth section, the performance of the experiments is evaluated and the results are displayed and analysed. At the last, the project is concluded with the key observations from the result analysis aligned to the problem statement. All the relevant references are attached at the reference section at the end of the project report.

3. Background

Convolutional Neural Network (CNN) and Transfer Learning models deals best with image dataset. As our data consists of large number of images about different types of brain tumours thus, we chose CNN and VGG-16 over any other neural network architecture as it takes advantage of local spatial coherence of images. This means that CNN can reduce dramatically the number of operations needed to process an image by using convolution on group of adjacent pixels, because adjacent pixels together are meaningful. Moreover, there are pooling layers, which downscale the image and hence, the number of parameters to be learned decreases which reduces network complexity leading to less chances of overfitting.

The concept of Transfer Learning originates from the idea of training the model on pre-trained parameters to optimize the model performance. This is an alternative approach to deal with image classification problem where the model weights are previously defined as benchmark and a deeper network is introduced which can learn on larger dataset in less time as the model is already trained with original parameters. This kind of pre training of hypermeters for optimization is often termed as state-of-the-art performance which is prevalent in the transfer learning models like VGG (VGG-16/ VGG-18), Inception and ResNet with respect to the CNN model.

3.1 Convolutional Neural Network

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics. In CNN, each input

image will pass through a sequence of convolution layers along with pooling, fully connected layers, filters (Also known as kernels). After that, we will apply the Soft-max function to classify an object. [2]

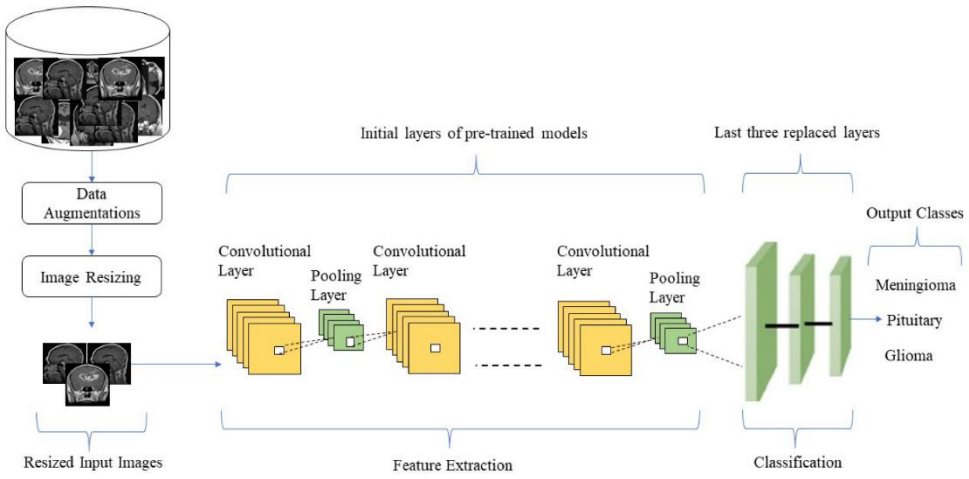

***Figure 1**: Architecture of Convolutional Neural Network*

Convolution Layer

Convolution layer is the first layer to extract features from an input image. Convolutional layer is core building block of CNN, it helps with feature detection. By learning image features using a small square of input data, the convolutional layer preserves the relationship between pixels. It is a mathematical operation which takes two inputs such as image matrix and a kernel or filter.[3]

- The dimension of the image matrix is **h×w×d** (where h = height, w= width and d=dimension of the image)
- The dimension of the filter is $\mathbf{f_h \times f_w \times d}$.
- The dimension of the output is $\mathbf{(h-f_h+1)\times(w-f_w+1)\times 1}$.

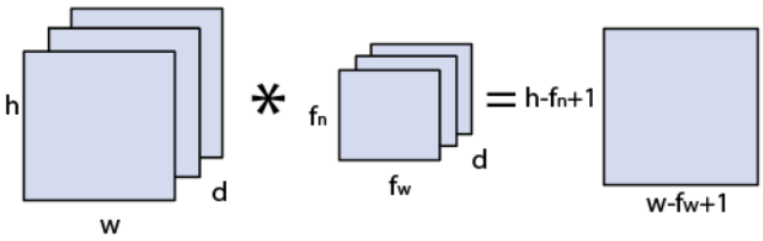

***Figure 2**: Image Matrix multiplies kernel or filter matrix*

Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations. It basically summarises the features generated by a convolution layer. Pooling layer plays an important role in pre-processing of an image. Pooling layer reduces the number of parameters when the images are too large. Pooling is "**downscaling**" of the image obtained from the previous layers. [3]

1. Max Pooling - - Max pooling is typically used, often with a 2x2 dimension. This implies that the input is drastically down-sampled, reducing processing cost.
2. Average Pooling - Down-scaling will perform through average pooling by dividing the input into rectangular pooling regions and computing the average values of each region.
3. Sum Pooling - The sub-region for **sum pooling** or **mean pooling** are set the same as for **max-pooling** but instead of using the max function we use sum or mean.

Fully Connected Layer: The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture. In this, the input image from the previous layers is flattened and fed to the FC layer.

Activation Function: Activation function decides which information of the model should trigger in the forward direction and which ones should not at the end of the network. It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. The sigmoid activation function is used for binary classification and for multi class classification the softmax is used.

Dropout Layer: Overfitting occurs when a particular model works good on the training data but fails while predicting on test data. Out of various ways of eliminating overfitting issue, one approach is using a dropout layer wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

Output Layer: The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer. At the end, the output of the Dense layer is fed to the output layer which consists of the softmax activation used for multiclass classification.

Hyperparameters: The hyperparameters are the design units of any network architecture. The number of hidden neurons, activation function, epochs, batch size, optimizers are the different hyperparameters which are used for this experiment.

- ***Epochs***: Epochs are basically the number of times the model takes to learn the input data. During training, the model needs to learn the pattern of the input data and it cannot learn it at one shot. The model certainly requires multiple cycles of learning to complete understand the pattern. Before the model starts training, the number of epochs is predefined so that the model knows the number of complete passes on the training data.

- ***Batch Size***: Batch size is the number of samples trained in a specific batch. There is no fixed value for this hyperparameter. If there is only one batch that contains the number of samples equal to the total number of training samples then it is called as batch gradient descent. In this experiment, it takes the mini batch gradient descent where in each batch the number of samples are less than number of training samples but more than one sample.

- ***Optimizers***: Gradient Descent is one of the primary concepts of neural network. This signifies the minimization of loss. Before understanding the optimizer function in the experiment, it is noteworthy to discuss the gradient descent at prior. Imagining a hike down from a hill, the trend is always towards the ground through its slope. In the same manner, gradient descent is a technique to calculate the loss while a model trains and then rectifying it to minimize the error. This goes on a loop until it reaches a minimum error point. The optimizers are the hyperparameters which takes care of minimization of error value in the network through the gradient descent approach. There are various available optimizers among which the one which is used for this assignment purpose is Adam. The Adam optimizer rectifies the error rate at an adaptive estimation approach.

- ***Learning Rate***: Learning rate determines the step size of gradient, and it is a hyperparameter which can be tuned to adjust the step size at which the model optimizes. The default value of learning rate in adam optimizer is 0.001.

- ***Filters***: Since feature map size decreases with depth, layers near the input layer tend to have fewer filters while higher layers can have more. To equalize computation at each layer, the product of feature values with pixel position is kept roughly constant across layers.

- ***Kernel:*** The kernel is the number of pixels processed together. It is typically expressed as the kernel's dimensions, e.g., 2x2, or 3x3.

- ***Strides:*** The stride is the number of pixels that the analysis window moves in each iteration. A stride of 2 means that each kernel is offset by 2 pixels from its predecessor. The following figure *Fig.3* shows that the convolution would work with a stride of 2.

- ***Padding****:* Padding is the addition of (typically) 0-valued pixels on the borders of an image. This is done so that the border pixels are not undervalued (lost) from the output because they would ordinarily participate in only a single receptive field instance.[2,3]

3.2 Transfer Learning

Transfer Learning is the reuse of a pre-trained model on a new problem. In transfer learning, the knowledge of an already trained machine learning model is applied to a different but related problem. For example, training a simple classifier to predict whether an image contains a backpack, the model gains a knowledge during its training to recognize other objects like sunglasses. With transfer learning, we basically try to exploit what has been learned in one task to improve generalization in another. The weights are transferred that a network has learned at "task A" to a new "task B." The general idea is to use the knowledge a model has learned from a task with a lot of available labelled training data in a new task that does not have much data. Instead of starting the learning process from scratch, we start with patterns learned from solving a related task.

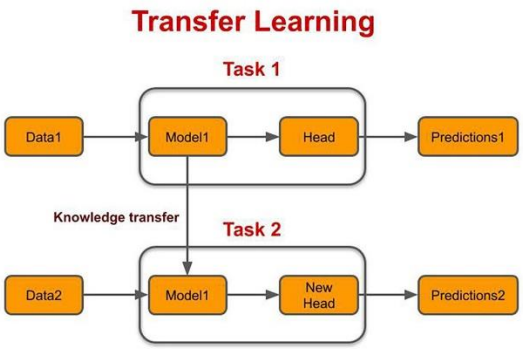

Figure 3*: Transfer Learning*

VGG16 Model

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.[7]

In this model we have 16 layers in the model- 13 convolution layers and 3 dense layers. We also have the Max-Pooling Layers which helps in reducing its dimensionality and allowing for assumptions to be made about features with more accuracy.

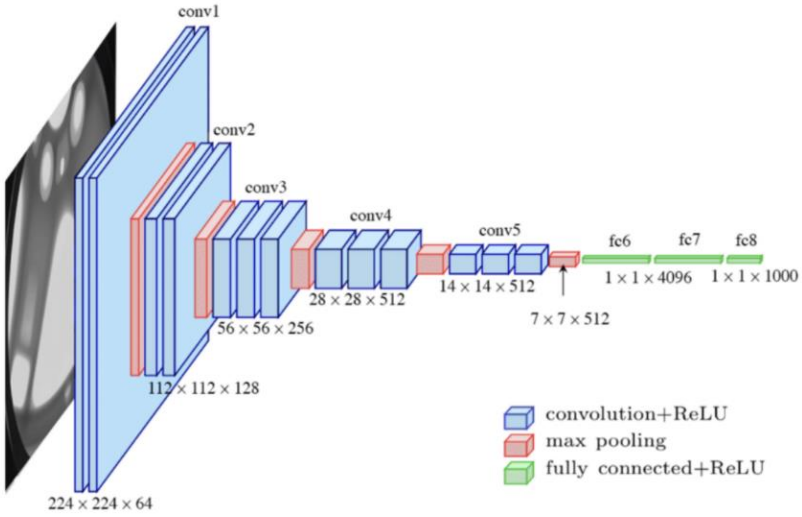

***Figure 4**: VGG16 Model*

The input to conv1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, centre). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels. The convolution stride is fixed to 1 pixel. The padding of conv. layer input is such that the resolution is preserved after convolution, i.e., the padding is 1-pixel for 3×3 conv. layers. Pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2-pixel window, with stride 2).

Three Fully Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels 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. The configuration of the fully connected layers is the same in all networks.

Working of VGG Model

Step 1: Prepare the dataset.
Step 2: Use Transfer Learning.
Step 3: Freeze the layer of the model.
Step 4: Load our Image Dataset.
Step 5: Train our layers.
Step 6: Load the model that we have trained.

4. Methodology

In this section, we are going to focus mainly on the implementation of the CNN model and the VGG model including all the steps performed to get an efficient accuracy on brain tumour classification.

4.1 Importing the necessary libraries:

All the required libraries from the relevant packages are imported. The major utilized packages are pandas for data framing operations, seaborn and matplotlib for data visualization, numpy for numerical estimations, sklearn and tensorflow for all kind of data pre-processing, scaling, machine learning algorithms, convolution neural network designing and performance evaluation.

Tensorflow is an open-source software library for machine learning and deep learning algorithms. Keras is a deep learning API written on top of the Tensorflow library [8]. For designing the CNN and VGG-16 in this experiment, the Keras API is implemented to form the sequential model layers.

The complete experiment is performed in the Jupyter Notebook environment.

4.2 Reading the data:

Once the libraries are imported, both the training and testing data is loaded into the notebook from the respective directories. In each directory, there are four distinct folders containing the corresponding image samples of the brain tumours. Using the 'os' library, the path is defined for every directory to load the images in a variable as an array of images for both training and testing. The below figure shows the sample MRI scans taken as the input data.

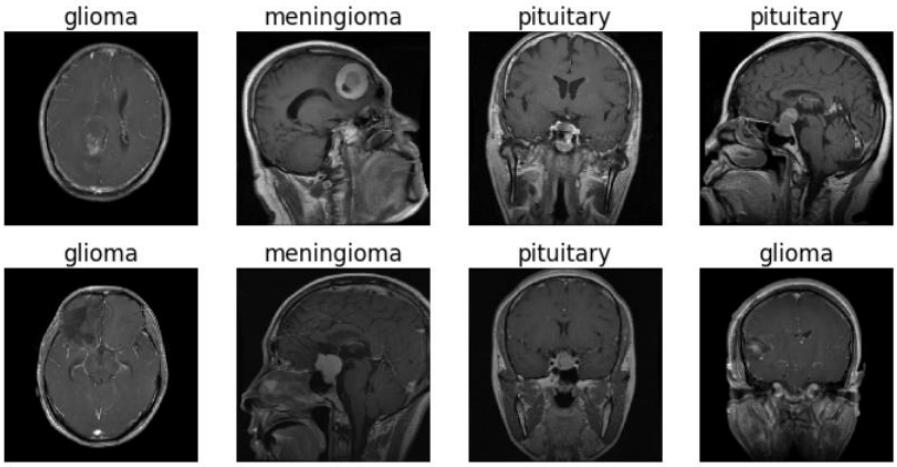

Figure 5*: MRI scans of three types of brain tumors*

4.3 Exploring the dataset:

We are using Brain Tumor MRI Dataset for this project which consists of 7023 images. The dataset consists of two folders Training and Testing, in which both the folders consist of 4 brain tumor categories like – Glioma, Meningioma, No tumor and Pituitary. The training data contains 5712 images and testing data has 1311 images. After creating the variables to store the images, distribution of each category of brain tumour type is visualized with the help of a pie chart. It is observed that the number of image samples are uniformly distributed in both training and testing datasets.

4.4 Data Pre-processing: Normalization

Data normalization is a process of scaling the images in a particular dimension so that the model can efficiently learn from it. Here the image size used is 150 pixels and divided the image by 255 so that it gets scaled while feeding to the model.

In the VGG-16 model that we used for improving the base model performance, we performed data augmentation to reduce overfitting which includes increasing the brightness (80% - 120%) and contrast (80% - 120%) of the input image.

4.5 Training-Validation data split

The training dataset is split into two sections- training and validation set with the ratio of 9:1 which means 90% of the sample images are preserved for training purpose whereas the rest 10% is used for validation purpose. The data splitting is done with random state defined as 42, which mentions that while splitting, the train set and validation set remain same throughout the training of the model. The splitting of data is highly significant to model performance. If the train and validation set varies at every execution, it would impact the training of the model.

The output column stored in y goes through one-hot encoding for mapping the target values into the numerical labels so that it is easier for the model to capture the output classes.

4.6 Training with CNN

4.6.1 Base Model

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 146, 146, 32)      832

 activation_18 (Activation)  (None, 146, 146, 32)      0

 conv2d_13 (Conv2D)          (None, 143, 143, 32)      16416

 activation_19 (Activation)  (None, 143, 143, 32)      0

 max_pooling2d_6 (MaxPooling  (None, 47, 47, 32)       0
 2D)

 conv2d_14 (Conv2D)          (None, 45, 45, 25)        7225

 activation_20 (Activation)  (None, 45, 45, 25)        0

 conv2d_15 (Conv2D)          (None, 44, 44, 25)        2525

 activation_21 (Activation)  (None, 44, 44, 25)        0

 max_pooling2d_7 (MaxPooling  (None, 14, 14, 25)       0
 2D)

 flatten_3 (Flatten)         (None, 4900)              0

 dense_6 (Dense)             (None, 250)               1225250

 activation_22 (Activation)  (None, 250)               0

 dense_7 (Dense)             (None, 4)                 1004

 activation_23 (Activation)  (None, 4)                 0

=================================================================
Total params: 1,253,252
Trainable params: 1,253,252
Non-trainable params: 0
_________________________________________________________________
```

***Figure 6**: Base Model Summary*

Firstly, by using keras API a sequential model is designed. We are using 4 convolution layers and 2 MaxPool layers in a sequence of two conv2D layer followed by one maxPool2d layer.

For first conv2D layer of 32 filters of kernel size 5 and input size of data is 150 which is equal to the image size. ReLu is used as the activation function. Similarly, in the 2nd conv2D layer all parameters are same except for the kernel size which is reduced to 4.

The second important layer in CNN is the pooling (MaxPool2D) layer. This layer simply acts as a down sampling filter. It looks at the 3 neighbouring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce over fitting. The pooling size (the area size pooled each time) is taken as 3x3, more the pooling dimension is high, more the down sampling will be obtained on the data.

A set of convolution layer is added again, with same activation function but reduced input shape, kernel size and the filters. This time the input shape taken is 100 and 25 for the both 3rd and 4th conv2D layers and kernel size is reduced to 2 till fourth conv2D layer. Later, one more pooling layer is added with a size of 3. After that we have added one flatten layer, one hidden layer and one output layer

The Flatten layer is used to convert the final feature and maps it into a one single 1D vector. This flattening step is needed so that we can make use of fully connected layers after performing convolution and max-pooling. It combines all the found local features of the previous Convolutional layers.

In the end, we used the flattened output in two fully-connected (Dense) layers which is just an artificial neural networks (ANN) classifier one with ‘ReLu’ activation function and the other layer uses "softmax" that takes the outputs distribution of probability of each class and correctly associate it with the label having highest probability. Now for compiling the model, we use ‘Adam’ optimizer which works on the principle of ‘accuracy’ metrics and ‘categorical_crossentropy’ loss taken into consideration of the model. The model then uses fit() function to start training on the data. In this function the epoch is set to 5 and the learning rate is set to default value of 0.001.

After training the model, the predict() function is used to check the testing performance on the new set of data (X_test). To evaluate the performance of the test values, the metrics considered are accuracy and precision. The sklearn function accuracy_score() and precision_score() calculates the respective metrics by comparing the true value of y and predicted y.

4.6.2 Tuned model 1 (Effect of tweaking learning rates and epochs)

In this step, we tune the model by changing some of the hyperparameters and addition of some more function has been done to get an optimized model with better accuracy. Like the previous model, we started with the sequential design. All the hyper parameters are kept same however the epoch size is increased to 10 this time and the learning rate is set to 0.0001.

Same as previous model, we have added one flatten layer, one hidden layer and one output layer. Concept of early stopping has been introduced in this model in order to stop the training of the model when the accuracy is not improving. In this case, epoch was set to 10 but due to early stopping call-back function, the model stopped training on the 8^{th} epoch. The model is then tested against the X_test samples of images.

4.6.3 Tuned model 2 (Effect of changing model topology- filters, strides, drop out layers, batch size, pool_size)

In this model we modified the CNN topology with change in number of filters, kernel size and batch size. The dropout layers are added in the model and the hyper parameters are extended including padding and strides. The set of sequential layers contains one convolution 2D layer with 64 filters, kernel size of (5,5), same padding, and activation function is ReLu. The shape of the input is 150. Then we have a maxpool layer for downscaling where pool size is (2,2). One dropout layer is added with size 0.25 after every combination of conv2D and MaxPool2D.

In the next conv2D layer, the number of filters is changed to 128 and kernel size to (3,3). In addition to this, two maxpooling layers are added of size (2,2) with stride of (2,2). The output of this layer goes to the following conv2D layer of number of filters as 128 and kernel size (2,2) followed by maxpool layer with same configuration. One dropout layer of size 0.3 is added.

In the last layer, the conv2D filters are changed to 256 and rest of the parameters are kept same as the previous layer. Same as previous model, we have added one flatten layer, for hidden layer the number of neurons is changed to 1024 and with 50% drop out. Before training the model, the epoch is set to 15 and the batch size is taken as 40. The model is then tested against the X_test samples of images.

Hyperparameters	Tuning 1	Tuning 2
Learning rate	0.0001	0.001
Epoch	10	15
Call back Function	Early Stopping	Early Stopping
Number of Filters	32, 25	64, 128, 256
Strides	1	2
Drop out layers	0	0.25, 0.3, 0.5
Batch size	32	40
Pool size	3	2
Kernel	5,4,3,2	5, 3, 2
Number of neurons (Dense)	250	1024

***Figure 7**: Hyperparameters Tuning in Tuned model 1 & 2*

4.6.4 Tuned model 3: Training with Transfer Learning model (VGG-16)

In the VGG model, the pre-trained base learner model is loaded keeping the 'include_top' parameter as False which represents that the input layer and the output layer are change according to the requirements of the brain tumour dataset. The image size is taken as 200 for training purpose. In this experiment, total three pre-trained layers of weight of 'imagenet' is taken from the Keras API.

Now, the output of the pre-trained base model is flattened using the Flatten layer and it is followed by a drop out of 30%. Another dense layer of 128 neurons is added which is activated by ReLu and has a drop out functionality of 20%. Finally, the output layer is defined with softmax as the activation function. For compilation of the model, the Adam optimizer is used with learning rate 0.0001, loss as 'sparse_categorical_crossentropy' and 'sparse_categorical_accuracy' as the metric.

With the above design of the VGG model, few other hyperparameters like epoch, batch size and step size are predefined. The epoch is 4, batch size as 20 and steps per epoch is the ratio of total number of sample images to the batch size. The model is trained on the specified parameters followed by testing against the test samples.

5. Evaluations/ Results

Different experiments were conducted to classify the MRI scans of brain tumour into four categories. To achieve the solution of the given problem statement, Python programming is used extensively with its various libraries and packages.

As a first step, data pre-processing is performed to scale the MRI scan images using normalization so that the images can be fed into the model for training purposes. The proposed dense- CNN base model achieves training accuracy of 94%, validation accuracy of 91% and testing accuracy of 92% as shown in below plotted graphs.

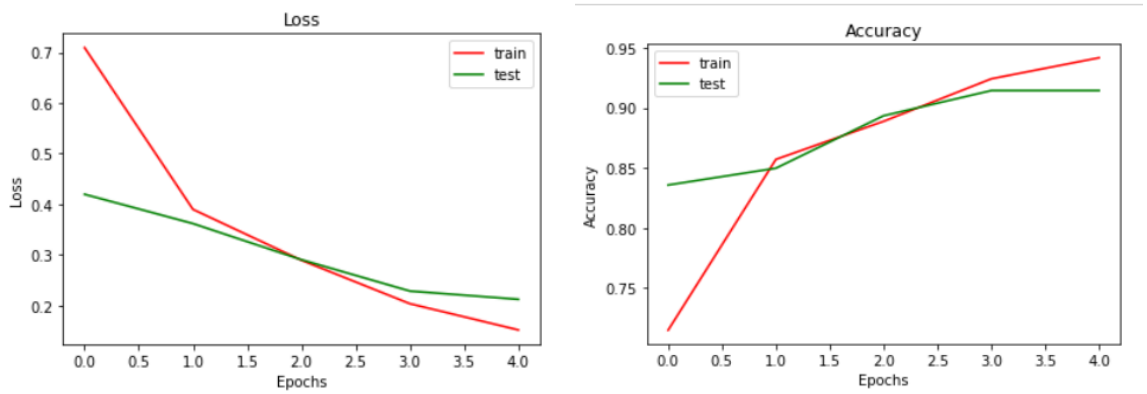

Figure 8: *Graph representing model accuracy and model loss for training and validation set using the dense-CNN base model.*

The experiment has been performed in 5 epochs and image size 150. After first epoch, the training accuracy is observed as 71%. But once the model starts learning, the training accuracy drastically increased to 85% after second epoch. The model keeps improving after every epoch till 4th where it freezes to output the accuracy as 94%. Initially the validation accuracy was 83% but after 4th epoch it reached till 91%.

The base model is tuned with hyperparameters where the learning rate is set to 0.0001 and epoch as 10 with early stopping. In this tuned model, the training accuracy initially observed as 71% which shoots up after second epoch to 85%. After 8th epoch the model saturates where the training accuracy reaches 99% because the training accuracy was not improving so the further training of model was stopped by call back.

A similar trend of validation accuracy is also visible starting from 86% to 92% after 8th epoch. The model after testing on the test dataset also gives a better accuracy of 93%. Therefore, the first tuned model was successful in improving the model performance as compared to the base model.

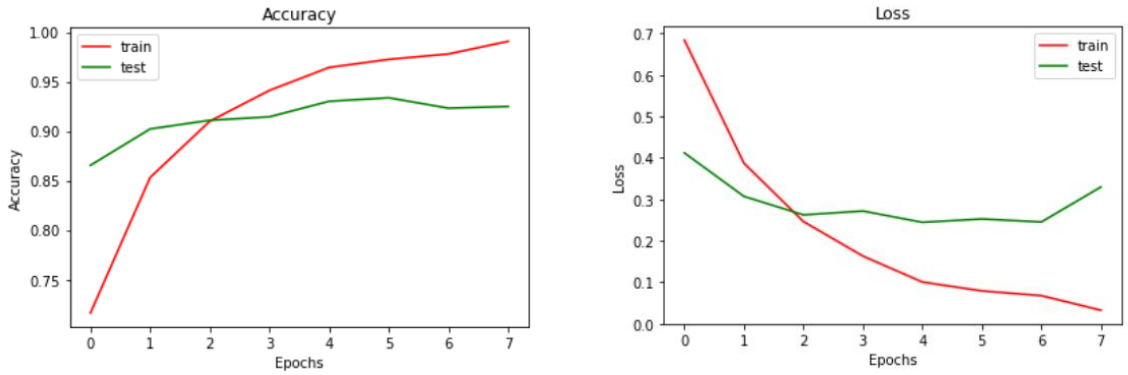

Figure 9: *Graph representing model accuracy and model loss for training and validation set using the dense-CNN first tuned model.*

The second stage of tuning involves the several changes like batch size as 40 and epoch as 15 along with the modifications in network design as discussed in the methodology which includes hyperparameter tuning. After first epoch, the training accuracy is observed as 58%. But once the model starts learning, the training accuracy drastically increased to 77% after second epoch. By the end of 15th epoch, the model gives the training accuracy as 95%. Initially the validation accuracy was 65% but after 15th epoch it reached till 91%. The model is then tested against the test data where the test accuracy is observed to be 90%. This result displays that the hypermeter tuning did not improve the result as compared to the base model and as well as the first tuned model.

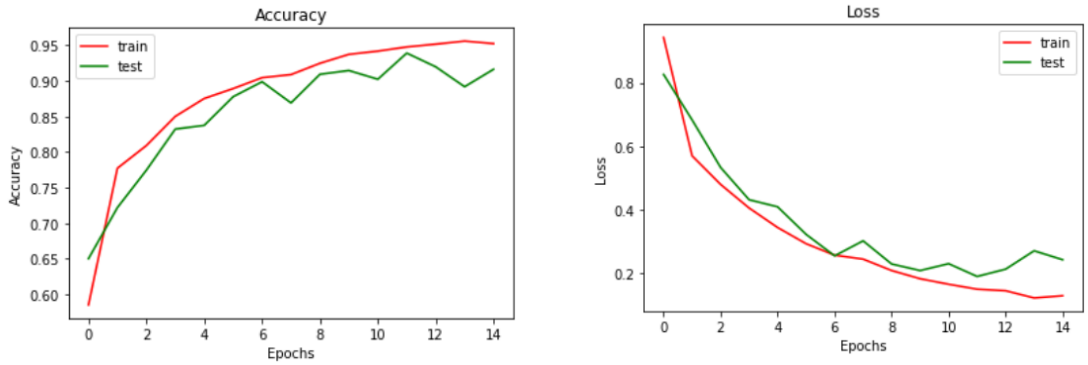

Figure 10*: Graph representing model accuracy and model loss for training and validation set using the dense-CNN second tuned model.*

For better accuracy of the brain tumour classification model, we decided to experiment with VGG-16 as a transfer learning model. In this experiment, after performing the data augmentation step the pre-trained hyperparameters are loaded for the VGG-16 model and then further trained through dense neural network. The epoch is set to 4, batch size as 20, image size as 200 and the steps per epoch as the ratio of total number of sample images to the batch size. With this configuration, while training the model it is observed that the sparse categorical accuracy received initially after epoch 1 is 84%. After training through four epochs the accuracy rises to 97%. The test accuracy reaches 96% after testing the model against X_test. This gives the highest accuracy so far among all the experimental models.

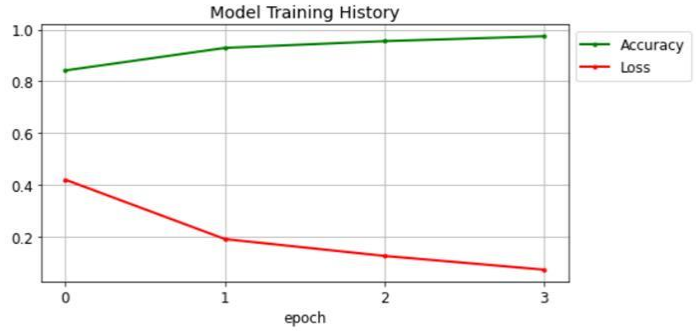

Figure 11: *Graph representing model accuracy and model loss for training set using the VGG-16 model.*

In the below figure, the consolidated classification report is shown for each of the discussed models.

Base Model - CNN

	precision	recall	f1-score	support
glioma	0.93	0.86	0.89	300
meningioma	0.85	0.84	0.85	306
notumor	0.97	0.98	0.97	405
pituitary	0.93	0.99	0.96	300
accuracy			0.92	1311
macro avg	0.92	0.92	0.92	1311
weighted avg	0.92	0.92	0.92	1311

First Tuned Model - CNN

	precision	recall	f1-score	support
glioma	1.00	0.78	0.87	300
meningioma	0.81	0.96	0.88	306
notumor	0.99	0.98	0.98	405
pituitary	0.96	0.98	0.97	300
accuracy			0.93	1311
macro avg	0.94	0.93	0.93	1311
weighted avg	0.94	0.93	0.93	1311

Second Tuned Model - CNN

	precision	recall	f1-score	support
glioma	0.80	0.96	0.87	300
meningioma	0.95	0.63	0.76	306
notumor	0.94	0.99	0.97	405
pituitary	0.92	0.99	0.95	300
accuracy			0.90	1311
macro avg	0.90	0.89	0.89	1311
weighted avg	0.91	0.90	0.89	1311

Third Tuned Model – VGG-16

	precision	recall	f1-score	support
glioma	0.93	0.96	0.94	300
meningioma	0.95	0.90	0.92	306
notumor	0.99	1.00	0.99	405
pituitary	0.97	0.98	0.97	300
accuracy			0.96	1311
macro avg	0.96	0.96	0.96	1311
weighted avg	0.96	0.96	0.96	1311

Figure 12: *Classification reports for all the models*

Different performance measures, such as accuracy, precision, recall, and F1-score, were utilized to compare the suggested model's performance. In all the models, the used performance metric for comparison is accuracy. In the other metrics also, the rate of performance is quite good. In the VGG-16 model, it is observed that the rate of precision, recall and F1 score is above 90 in all the different categories of brain tumors. The accuracy is considered as the primary metric for comparison since it is quite commonly used for analysis and better understanding of the predicted model evaluation. In this specific problem statement, it is important to efficiently categorize patients having the tumor and not having the tumor, without giving the false results. Thus, accuracy which measures the true positives

and true negatives is significant to evaluate the classification model. These parameters are evaluated using the confusion matrix. The details were also examined using the confusion matrix which is shown in Figure 13. In confusion matrix, a table is generated among the true values and predicted values of the four classes.

Typically, from this 4 X 4 matrix, it is easier to calculate any performance metrics of classification [9].

True Positives: The sample class is Positive and even predicted as Positive.

True Negatives: The sample class is Negative and even predicted as Negative.

False Positives: The sample class is Negative but predicted as Positive.

False Negative: The sample class is Positive but predicted as Negative.

Accuracy is the ratio of total correctly predicted values to the total observations.

$$Accuracy = TP+TN/TP+FP+FN+TN$$

Precision is the ratio of the total predicted positive values to the total actual positive observations.

$$Precision = TP/TP+FP$$

Recall is the ratio of actual positive labels correctly identified by the model.

$$Recall = TP/TP+FN$$

F1 Score is the harmonic mean of precision and recall.

$$F1 = 2 * (Precision * Recall)/ (Precision + Recall)$$

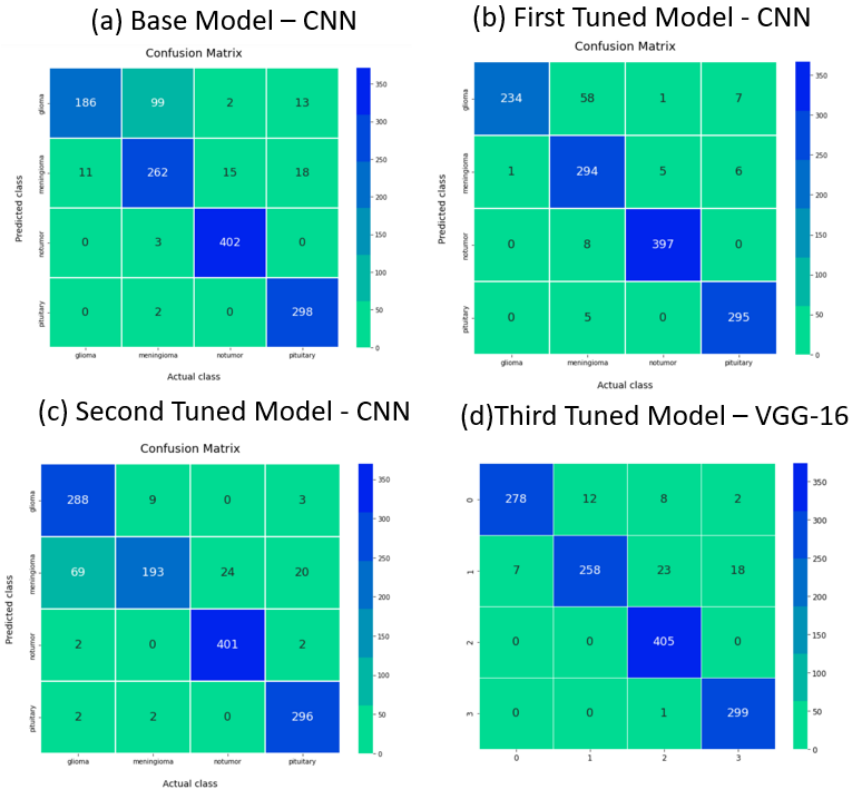

Figure 13. *Confusion matrix of: (**a**) proposed dense CNN base model; (**b**) Tuned Model 1; (**c**)Tuned Model 2; (**d**) Tuned 3- VGG-16 model*

6. References

1. Brain Tumor Classification (MRI) | Kaggle
2. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science
3. Convolutional Neural Network - javatpoint
4. Glioma - Symptoms and causes - Mayo Clinic
5. Meningioma - Symptoms and causes - Mayo Clinic
6. Pituitary tumors - Symptoms and causes - Mayo Clinic
7. VGG refer: https://neurohive.io/en/popular-networks/vgg16/
8. https://keras.io/guides/
9. https://www.analyticsvidhya.com/blog/2021/06/confusion-matrix-for-multi-class-classification/