MTBTDet_CNN: Manually Tunned Hyperparameters Convolutional Neural Network for Detection of Brain Tumor

Kalpana Devi, Aman Kumar Sharma

[a] Department of Computer Science, Himachal Pradesh University, Summer Hill, Shimla 171005, India

ABSTRACT

Accurate detection and classification of brain tumors from MRI scans are vital for timely diagnosis and treatment planning. Convolutional Neural Networks (CNNs) have demonstrated significant promise in this domain; however, their performance largely depends on the careful selection of hyperparameters such as learning rate, optimizer, activation function, pooling type, batch size, number of convolutional layers, and epochs. Most existing studies rely on automated optimization techniques like genetic algorithms, grid search, or Bayesian optimization, which operate as black-box approaches with limited interpretability, while others use arbitrary or partially tuned hyperparameters without systematic experimentation. To address these gaps, this research conducts an extensive manual hyperparameter optimization process across seven key parameters through six controlled experiments using a publicly available Kaggle MRI brain tumor dataset comprising 3,000 MRI images of both tumorous and non-tumorous brain classes. The results reveal that the Adam optimizer with a learning rate of 0.003 provides the best balance between convergence stability and classification accuracy. Using this configuration, a customized CNN model named MTBTDet_CNN was developed for MRI-based brain tumor detection and classification. Experimental findings demonstrate that the proposed model achieves superior performance across multiple evaluation metrics - including accuracy (0.997), precision (0.993), recall (1.000), F1-score (0.997), and specificity (0.993), outperforming existing optimization and manual tuning approaches, thereby validating the effectiveness of the proposed manual tuning strategy.

Keywords: Hyperparameters; Optimization techniques; Manually tunned; Brain tumors; Detection; CNN

1. Introduction

An abnormal mass or development of cells in or close to the brain is called a brain tumor. The human body's organs and tissues consist of small blocks called cells. These cells are divided to generate new cells during healing, growing, and repairing in a controlled manner. The human body signals the cells about when to divide, grow, and stop growing [1]. But when the normal mechanism of the cells goes wrong i.e., they divide the cells in an uncontrolled manner called a tumor. Both benign (noncancerous) and malignant (cancerous) brain tumors have existed. Primary brain tumors and secondary brain tumors are the two primary forms of brain tumors. Of these tumors, about 70% are primary, meaning they start in the brain itself. The remaining 30% are secondary, or metastatic, meaning they start in other body parts, such as the liver, kidney, or lung, and then move to the brain [2]. Benign (noncancerous) primary brain tumors may grow slowly and press on brain tissue while malignant (cancerous) primary brain tumors can grow rapidly and destroy brain tissue, causing damage. Glioblastoma and medulloblastoma are examples of primary brain tumors. Metastatic brain tumors are considered malignant. The symptoms of a brain tumor might vary depending on its, size, location, and speed of growth. Headaches, weakness, poor coordination disturbed perceptions, and other symptoms are possible [3, 4].

Radiologists work on different medical imaging modalities to identify the tumor in brain images [5]. Information about human body components is often obtained by using Magnetic Resonance Imaging (MRI) and Computerized Tomography (CT) scans. An advanced technique of MRI generates high-quality 3D slices from multiple directions of the human body parts. Therefore, the tumor of the most sensitive organ brain is analyzed by the MRI modality and hence has an essential factor in the diagnosis of the tumor's stage for deciding the correct treatment or therapy for the infected person [6, 7]. But the identification of brain tumors before any treatment or therapy is quite difficult because of the different tumor sizes and shapes. MRI images can be used in a variety of ways to identify brain cancers. Traditionally, healthcare professionals identify tumors by visually analyzing medical images and accurately identifying tumor locations. Due to the presence of adjacent healthy tissues, tumor borders can be challenging to discern. Manual identification takes a long time and may lead to misinterpretation of tumors. The reasons for misinterpretation are that the human eye struggles to differentiate the various shades of gray visible in MRI and the noisy MRIs due to the variations in imaging equipment and exhausted radiologists [8, 9]. Due to these challenges, conventional tumor identification methods are gradually giving way to automated systems for classifying brain tumors. Methods based on deep learning consistently exhibit strong efficacy in various image-processing applications in the medical field [10].

Automated brain tumor detection techniques can be built on machine learning (ML) and deep learning (DL). Manually creating the extraction of features is the foundation of traditional machine learning techniques, which require that certain features be

taken out of training images before the learning process. This approach often requires expert intervention to identify crucial features. However, when dealing with large databases, ML-based methods exhibit limited detection accuracy and are prone to errors [11]. Algorithms based on deep learning have proven very efficient in addressing these issues across extensive applications, including imaging in medicine. The Convolutional Neural Network (CNN) is a well-known deep learning model because of its architecture of sharing the weights, which facilitates the automated extraction of both high- and low-level features from training data. Thus, DL-based methods for brain tumor identification have drawn attention from scientists [6]. Creating a deep learning algorithm i.e., CNN is a multifaceted endeavor that encompasses several key components. These include model selection, dataset collection and preparation, and hyperparameter optimization. Each of these elements significantly influences the algorithm's ultimate performance. Specifically, hyperparameter optimization is essential for fine-tuning a CNN model by identifying optimal values for its hyperparameters during the learning phase.

Adapting the complexity of a CNN model to the specific task is crucial. A very simple model might not be able to extract all the important information from the data, which would lead to underfitting and poor generalization. On the other hand, a model that is too complicated could make the training set overfit and still have poor generalization. The recommended approach for manual model tuning is to begin with a basic architecture, optimization of hyperparameters, and analyze its results using the training set. If the model's training metrics are unsatisfactory, consider adding more layers (increasing complexity) and repeating the process. Alternatively, Neural Architecture Search (NAS) techniques can streamline this process, albeit at the expense of additional computing time [12].

1.1. Hyperparameters of CNN

ImageNet Large Scale Visual Recognition Competition (ILSVRC) [13] is the popular CNN, which was the subject of intense study in many different domains. The architecture of CNN affects performance when we apply it to our given problems, hence a suitable design is required. Because of this, numerous research on CNN architecture design has been published [14 - 16] comprises model parameters and hyperparameters. Weights and biases are examples of model parameters that let the model adjust to the data. On the other hand, because they are unlearned from the training process, hyperparameters, which control the entire training process, must be preset. The model's learning ability, complexity, and rate of convergence for model parameters are all determined by the hyperparameters. Consequently, finding the optimal values for hyperparameters improves efficiency and yields better results for the model [17]. A useful method for locating the ideal parameters in the recommended CNN model is hyperparameter tuning. Additionally, as not all hyperparameters have the same effect on the training and testing of the model's performance, the selection of hyperparameters for tuning has a substantial influence on the outcomes. Hyperparameter tuning identifies the optimal parameter combination for the model, resulting in maximum performance [18]. The following hyperparameters must be configured before the CNN model is trained:

1.1.1. The Number of Convolutional Layers

Numerous parameters affect a convolutional layer; the most crucial ones are stride, zero padding, the height and width of each convolutional filter, and the number of filters deployed to every layer. If there is no padding at all, the convolved image gets smaller. The stride determines how much the kernel moves after each calculation. A neural network's depth is established by the number of convolutional layers it contains. While deeper architectures have generally shown improved results, designs with fewer layers have also been suggested. These shallower architectures show that a decently deep network can nevertheless function when competing with more sophisticated versions, which is especially helpful for embedded systems with limited computing capacity [19]. Generally, it is advisable to continue adding layers to a neural network until the test error plateaus. However, this comes with the computational cost of training. Having very less layers can cause underfitting, while a larger number of layers is usually safe when regularized properly [20]. The impulse response, also known as a mask or filter, is multiplied by the input image in the convolution layer's operation. In this procedure, a two-dimensional (2D) convolution is carried out. It does this by convolving in both the horizontal and vertical directions inside the 2D spatial domain, as stated as:

$$O_{(m,n)} = (I)_{(m,n)} F_{(m,n)} = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I_{i,j} \times F_{(m-i,n-j)} \quad (1)$$

where the 2-D input picture is represented by $(I)_{(m,n)}$. A 2-D picture is vertical for I and horizontal for m, n. The filter or kernel that produces weights is represented by $F_{(m,n)}$, where k is the range and i, j spans the filter's dimensions.

1.1.2. Pooling Layers (PLs)

In CNN, pooling has a few key functions. Firstly, pooling makes the feature map smaller, which allows for less computation during training. Secondly, pooling makes each neuron's response field bigger, which improves recognition ability by enabling every neuron to detect a greater area of the input image. Thirdly, pooling mitigates the adverse impacts caused by tiny

distortions and noise. As only the delegates' values will remain after the poling, therefore the light noise can be eliminated. Hence, pooling seeks to reduce calculations and enhance recognition performance at the same time [21].

- **Max Pooling:** Max pooling is a widely used pooling strategy that chooses the feature map's maximum element sheltered by the filter, as illustrated in left side of Figure 1. Based on the stride and filter's values or size, the output obtained is a feature map, which contains the dominant features from the input [22]. Consequently, when it comes to photos, Max Pooling supports keeping the input's lighter regions.
- **Average Pooling**: Average pooling determines the mean value of the pooled area [23], as illustrated in right side of Figure 1. Unlike max pooling, which looks for the best features, average pooling finds a patch of features, uses that patch to inform some calculations, and produces a smoother output. This may lead to the result by decreasing accuracy. Usually, it depends on the use of the outcome and the consistency of the features (pixels).

Fig. 1. The image illustrates the max pooling (left) operation using a 2×2 window on a 4×6 input matrix and get the 2×3 output matrix and average pooling (right) operation using a 2×2 window on a 4×6 input matrix and get the 2×3 output matrix.

- **Global Max and Global Average Pooling:** Rather than selecting specific patches from the input feature map, global pooling downsizes the whole feature map to only one value by calculating the maximum or average.
- **Mixed Pooling:** Max pooling and average pooling are the two conventional pooling operations that are mostly used and fixed in an image or channel. Allowing the pooling operations to be coupled (max pooling and average pooling) to learn about themselves is another logical extension of pooling operations [24].

1.1.3. Non-linear Layers/ Activation functions

Activation functions (AFs) typically come in after the convolutional layer, which results in non-linearity in each neuron's output. Consequently, the network will be able to acquire a range of difficult jobs. Non-linear activation layers, also called the learnable layers like convolutional and fully-connected layers, are applied in CNN design following all weighted layers. Because of the non-linear performance of the activation layers, the aligning of input to output will be non-linear, allowing the CNN to learn incredibly difficult jobs. The ability to distinguish is another key criterion for the activation function as it makes it feasible to train the network through error back-propagation. This nonlinearity is essential since a model that solely employs linear operations will be unable to adequately represent the complicated, non-linear behavior of many events in the actual world. As a result, activation functions enable CNNs to mimic a greater variety of functions and become more expressive. In general, the model's convergence and interpretation can be greatly impacted by the selection of activation functions in CNNs. As, different functions have varying advantages and disadvantages, a key component of creating a successful CNN is selecting the appropriate function for the given task [25-27]. The most frequently utilized activation function types in CNN are as follows:

- **Sigmoid:** Real numbers are the AF's input, and the output can only be between zero and one and widely used for the task of binary classification [28]. The S-shaped sigmoid function curve is mathematically represented as:

$$f(x)_{sigm} = \frac{1}{1+e^{-x}} \tag{2}$$

- **Tanh:** Since real numbers constitute its input, it is a hyperbolic tangent function like the sigmoid function, but it can only produce a smooth output that is between -1 and 1[29]. It is represented mathematically as:

$$f(x)_{tanh} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3}$$

- **ReLU (Rectified Liner Unit):** The function that is most frequently utilized when using CNN is the ReLU AF. It converts every value in the input to a positive integer. Its primary advantage over the other AF is the less computational load. It is mostly used in the hidden layers as it solves the problem of vanishing gradient [30]. Mathematical representation this function is:

$$f(x)_{ReLU} = \max(0, x) \tag{4}$$

In the context of using the Rectified Linear Unit (ReLU), several notable challenges can arise. For example, when the negative input passes through a back-propagation method with a substantial gradient, which flows over the ReLU function, it can lead to weight updates that deactivate the neuron. This phenomenon is commonly known as the 'Dying ReLU' issue.

- **Leaky ReLU:** The Leaky ReLU AF makes sure that negative inputs are never entirely ignored, in contrast to ReLU, which down-scales undesirable inputs. It is specifically designed to handle the problem of 'Dying ReLU' [28]. The mathematical representation of Leaky ReLU can be shown as:

$$f(x)_{LeakyReLU} = \begin{cases} x \; if \; x > 0 \\ mx \; if \; x \leq 0 \end{cases} \tag{5}$$

Where 'm' is a leak factor that has been set to be a very modest value for instance 0.01.

- **PReLU (Parametric ReLU):** The Parametric ReLU (PReLU) shares similarities with Leaky ReLU. However, its key distinction lies in the dynamic leak factor, which adapts during the process of training the model. Mathematically expression for the Parametric ReLU is:

$$f(x)_{PReLU} = \begin{cases} x \; if \; x > 0 \\ ax \; if \; x \leq 0 \end{cases} \tag{6}$$

Where 'a' is the learnable weight.

1.1.4. Optimizers

One of the two main problems in the learning process is choosing the learning algorithm, often known as the optimizer. The second is using various upgrades, such as momentum, AdaDelta, and Adagrad, in confluence with the learning technique to enhance the output. The gradient descent algorithm continuously alters the network parameters during every training session to reduce the training error. In particular, to calculate the gradient, or slope, of the cost function and update the parameters accordingly, it must apply a first-order derivative to the network parameters. To reduce the error, the parameter is then modified in the exact reverse direction as the gradient. The procedure of extending the gradient at each neuron to every other neuron in the above layer is known as network back-propagation., which is how the parameter update process is carried out [31, 32]. The mathematical representation of this operation is:

$$W_{ij^t} = W_{ij^{t-1}} - \Delta W_{ij^t}, \qquad W_{ij^t} = \alpha \times \frac{\partial L}{\partial W_{ij}} \tag{7}$$

The weight in this epoch of training is represented by W_{ij^t}, whereas the weight from the previous (t-1) training epoch is represented by $W_{ij^{t-1}}$. α is the learning rate and L is the prediction error or the loss. The following are many gradient-based learning algorithms:

- **Mini-Batch Gradient Descent (MBGD):** MBGD approach breaks down the training samples into many mini-batches, each of them is an undersized collection of samples that does not overlap [32]. The settings are then adjusted after the gradient of each mini-batch has been calculated. This method's advantage relies upon a combination of the benefits of both BGD (Batch Gradient Descent) and SGD (Stochastic Gradient Descent) approaches. As a result, it has extra memory utility, higher computational efficiency, and consistent convergence as compared to SGD and BGD. However, it produces noise due to which the convergence is a little bit slow [33].

- **SGD with Momentum:** The convolutional neural network's objective function is to utilize this technique. By combining the calculated gradient from the earlier training phase, which is weighted using a quantity called the momentum factor, it increases training speed and accuracy simultaneously. Instead of locating the global minimum, it can occasionally become stuck in a local minimum. A primary drawback of gradient-based learning techniques is this restriction. These problems frequently occur when there is a lack of convex surface (or solution space) for the problem. Momentum is employed in conjunction with the learning method to overcome this, and its mathematical expression is represented as:

$$w_t = w_{t-1} - \alpha v_t$$
$$where \; v_t = \beta v_{t-1} + (1-\beta)\nabla_w L(w_{t-1}) \tag{8}$$

where w_t are the updated weights, with the learning rate α, the Loss_Function's gradient for the weights at time t - 1 is represented by $\nabla_w L(w_{t-1})$, where β denotes the momentum coefficient and v_t represents the momentum at time t.

The momentum factor i.e., β is kept between 0 and 1 to minimize inaccuracy. As a result, to reduce errors, the weight update step size increases toward the minimum. The model can no longer avoid local minima at very low momentum factors. On the other hand, a large momentum component makes convergence go more quickly. However, the model may overshoot the global minimum if the learning rate (LR) is coupled with a high momentum component. Weight updates are smoothed when the gradient direction varies constantly throughout training by using an appropriate momentum factor, or hyperparameter.

- **AdaGrad (Adaptive Gradient Descent):** The AdaGrad method alters the learning rate for every weight depending on the gradient magnitudes detected during training. Convergence is slower along steeper paths and faster along flat ones made possible by this adaptation. The following is an expression of the AdaGrad-specific updating rule:

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{G_t + \epsilon}} \nabla_w L(w_{t-1})$$

Where, $G_t = G_{t-1} + (\nabla_w L(w_{t-1}))^2$ (9)

Where ϵ is a small constant that prevents division by zero and G_t is the diagonal matrix of sums of squares of previous gradients up to time t.

- **RMSprop (Root Mean Squared Propagation):** The RMSprop algorithm dynamically uses the average change of the squared gradients to adjust the rate of learning for every weight. This adjustment keeps the learning rate from increasing unnecessarily. The following is the RMSprop update rule:

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{G_t + \epsilon}} \nabla_w L(w_{t-1})$$

$$where \;\; G_t = \beta G_{t-1} + (1-\beta)(\nabla_w L(w_{t-1}))^2 \quad (10)$$

where β is the decay rate and G_t is the moving average of the squared gradients up to time t.

- **Adam (Adaptive Moment Estimation):** The Adam optimizer, an effective algorithm in machine learning, combines concepts from momentum and AdaGrad. It adapts the learning rate for a variety of neural networks. The following is how Adam's update rule is expressed.

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t \quad (11)$$

Such that, $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$

$$where \quad m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_w L(w_{t-1})$$

$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$

$$where \quad v_t = \beta_2 m_{t-1} + (1-\beta_2)(\nabla_w L(w_{t-1}))^2$$

The gradients' first and second-moment values are denoted by m_t and v_t, the decay rates for the first and second moments are represented by β_1 and β_2, the first and second-moment values are bias-corrected by $\hat{m}_t$ and $\hat{v}_t$, and ϵ is a small constant that prevents division by zero.

- **Nadam (Nesterov Accelerated Adam):** Nesterov-accelerated Adaptive Moment Estimation, or Nadam [34], is an enhanced form of Adam momentum that leverages knowledge from NAG (Nesterov Accelerated Gradient). Nesterov redefines momentum for non-random targets so that the momentum step is independent of the current gradient. The gradient update produced by Nesterov's momentum is superior to the classical momentum.

1.1.5. Learning Rate

To prevent adverse effects on the learning process, attention must be paid to selecting the learning rate and determining the step size for parameter changes. The learning rate (LR) in an optimization algorithm governs the frequency of weight updates. It controls how quickly the network adjusts its parameters during training. Figure 2 demonstrates the impact of various LRs on gradient descent. A high learning rate promotes rapid convergence but can lead to unstable, oscillating training. Conversely, a low LR ensures stable, smooth training but may slow convergence. To strike the right balance, it is essential to experiment with various LRs and find the optimal trade-off between training speed and stability [35]. We have possibilities like a set

learning rate, that decreases gradually, momentum-based approaches, or adjustable learning rates, based on the optimizer.

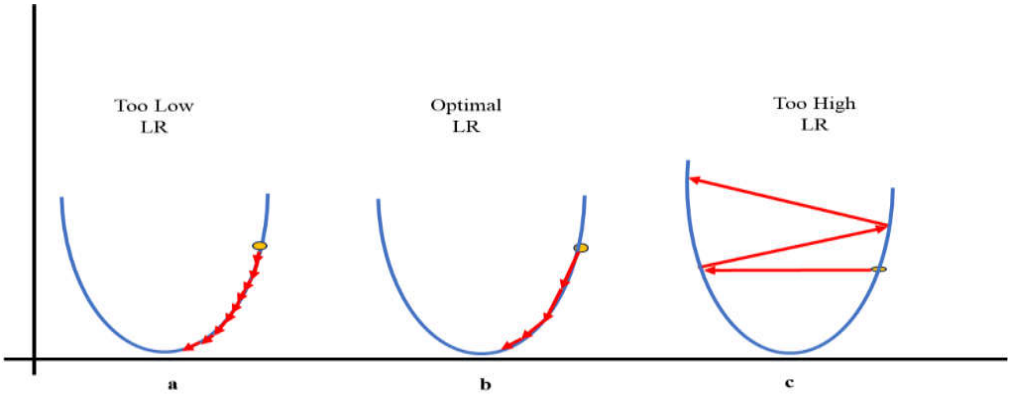

Fig. 2. The impact of different learning rates on gradient descent (a) a small LR needs numerous upgrades to achieve the minimum point. (b) optimal LR smoothly reaches the minimum point (c) very large of a LR results in abrupt updates that induce diverse behaviors.

1.1.6. Batch Size

Another hyperparameter is the Batch Size (BS), which affects how many samples the network processes in a training iteration. Larger BSs can enhance gradient estimate stability and training stability, but they come with increased memory demands and potentially slower convergence. Conversely, smaller BSs reduce memory requirements and speed up convergence, but they may yield noisier gradient estimates. To strike the right balance, it is crucial to test out different batch sizes and select the one that optimally balances stability and speed. A common recommendation is using BS powers of two for more efficient code execution [36].

1.1.7. Epochs

In machine learning, an epoch is a single pass of the learning process across the whole training dataset. Stated differently, the neural network learns patterns from all the data samples within an epoch, and the model's weights and biases are changed according to the computed loss or error. The model gains knowledge from the complete dataset like a full training cycle. Remember that because the learning rate is a hyperparameter, choosing the right one is essential to preventing the learning process from being adversely affected. Epochs represent how often the neural network processes the entire training dataset. When a small discrepancy arises between training and test errors, it is advisable to raise the number of epochs.

1.2. Contribution or Novelty of Work

While numerous papers in the literature explore various approaches for setting convolutional neural network (CNN) hyperparameters, none of them have proposed a universally applicable and robust systematic solution to this problem. Consequently, hyperparameter optimization remains an ongoing challenge. In this work, we examine experimentally how different hyperparameters based on CNN accuracy affect the results. This research tries to throw light on the practical implications of these hyperparameters and their effects on model performance. Furthermore, an incorrect choice of a single hyperparameter (such as using a sigmoid activation function) can prevent the neural network from converging, regardless of other hyperparameter settings. The heuristics can help guide hyperparameter selection but do not guarantee optimal results. The behavior of a CNN is highly reliant on the specific dataset, which complicates the establishment of a universally applicable theory for defining appropriate hyperparameters across all problems. The main contributions of this article are:

- *Find Research Gaps*: Analyzing the research gaps in each study and comparing the methods of hyperparameter optimization used in existing custom CNN models for the detection and categorization of MRI brain cancers.

- *Selection of Hyperparameters*: For the manual tuning optimization technique, seven hyperparameters with the range of values such as gradient optimizers (SGDM, Adagrad, RMSprop, Adam, and Nadam), LR (0.03, 0.02, 0.01, 0.003, 0.002, 0.001, 0.0003, 0.0002, 0.0001), BS (8, 16, 32, 64, and 128), activation functions (ReLU, Leaky_ReLU, ELU, and PReLU), pooling layers (Max Pooling, Average Pooling, Global_Max Pooling, Global_Average Pooling, and Mixed_scale Pooling), number of convolutional layers (1, 2, 3, 4, 5, and 6), and epochs (20, 25, 30, 35, and 40) are selected.

- *Experiments on Selected Hyperparameters*: Six experiments are carried out to fine-tune selected hyperparameters through

CNN model training and evaluation of the optimal hyperparameters according to the accuracy measure.

- *Proposed a custom CNN model (MTBTDet_CNN)*: Using the MRI Dataset of brain tumors and the optimal combination of manually tuned hyperparameters, which yields remarkable results depending on various kinds of performance measures like recall, accuracy, f1-score, precision, and specificity.

- *Comparative Analysis*: We conduct a thorough evaluation of the proposed model against the studies of optimization techniques and manually tuned methods for the detection and classification of MRI brain tumors. The comparison shows that our proposed model performs better than the existing techniques, thereby strengthening the efficacy of our strategy of manually tunned hyperparameters.

The additional sections of the manuscript are arranged as follows: Comparative Analysis of Related Work (Section 2), which outlines the advancements made to date and their gaps; Research Methodology (section 3), including the details of the proposed approach, an explanation of the dataset, and the values of the hyperparameter that needs to be adjusted; Results and Discussion (section 4), outlining the results of experiments carried out to adjust the hyperparameters, the result of proposed model based on various metrics, and comparison analysis with existing methods; and Conclusion (section 5), offering a summary of the work and suggestions for additional work.

2. Comparative Analysis of Related Work

Table 1 presents a comparative analysis and highlights research gaps in various hyperparameter optimization techniques employed by custom CNN models for the detection or classification of MRI brain tumors. The table systematically outlines key elements such as the reference and year of each study, the dataset used, the number and size of classes, the optimization methods applied, the hyperparameter search space considered, and the optimum hyperparameter values obtained. Additionally, it summarizes the test accuracy achieved by each model and identifies critical research gaps. This structured comparison provides insights into the effectiveness and limitations of different optimization strategies, offering direction for future research in enhancing CNN-based brain tumor detection.

In [37], the researchers employ a Genetic Algorithm for the optimization of the hyperparameters and propose the custom CNN with an accuracy of 94.2%. They select the appropriate parameters, such as the number of convolutional and max-pooling layers, the number of kernels and their sizes, the number of fully connected layers, the activation function, the dropout probability, the optimization technique, and the learning rate with their corresponding values, to evolve the optimal structure of CNN. However, the final output values of the selected parameters are not mentioned and the recall value of the classification results are also not evaluated. In [38], for the task of classification, the researchers used the Grid Search Optimization Algorithm for the MRI brain tumor by using three datasets and proposed three CNN models with accuracies of 99.33%, 92.66%, and 98.14%. The number of convolution and max pooling layers, number of fully connected layers, number of filters, filter size, activation function, mini-batch size, momentum, learning rate, and L2 Regularization are the included hyperparameters in the Grid Search technique. The two main hyperparameters, gradient-optimizers and number of epochs are not included by the researchers. In [39], researchers implied a unique CNN architecture for the categorization of MRI brain tumors and used the Bayesian Optimization approach for hyperparameter tuning. The only architectural hyperparameters that the authors address are dropout percentages, max pooling size, Conv2D kernel size, Conv2D filters, and dense filters. During the optimization process, the fine-tuning of other hyperparameters such as the optimizer, LR, BS, etc., are ignored. In [40], the researchers also used the Bayesian Optimization technique for tuning the hyperparameters and proposed the architecture of CNN for the classification task by achieving an accuracy of 98.70%. The activation function, dropout rate, BS, number of dense nodes, and gradient descent optimizer are the hyperparameters used in optimization approaches. There is a missing number of epochs and LR.

Researchers in [41] manually tuned the hyperparameters and suggested using a customized CNN model to categorize MRI brain tumors into three classes. They explained how the feature learning sensitivity or recall of the built model is affected by how many epochs there are. The value of all other hyperparameters is chosen randomly. The authors in [2] proposed the novel CNN model to classify brain tumors utilizing two databases by the process of manual tuning of hyperparameters and achieved 96.13% and 98.7% accuracies. They used many hyperparameters for tuning but the optimum values of only AF, BS, and optimizer are shown. In [42, 43] studies, the researchers proposed the custom CNN models for the classification and detection the brain tumors and achieved 100% and 97.28% accuracies respectively. In both studies, the authors took the random values of the hyperparameters. In [44], researchers proposed the scratch CNN model for binary classification of tumors in the brain and achieved 96.49% accuracy. They tested the model for a few hyperparameters such as PL, AF, optimizer, and initializer during tunning and other hyperparameter values chosen randomly. In [45-47] studies, researchers proposed their custom CNN models for the classification and detection of tumors in the brain. They all not focused on the hyperparameters tunning and choose the random values for the hyperparameters. In [48], the authors employed the CNN model by using a hard swish-based ReLU AF for the classification task. The model is tested or compared with only sigmoid and tanh AF and no experimental result was conducted with other hyperparameters. The en-CNN model was presented by authors [49] for the manual hyperparameter tuning-based binary classification of brain tumors. The hyperparameters chosen by them for tunning are the convolutional

+ReLU layers, dropout layers, epochs, fully connected layers, optimizers, BS, and dropout Rate. The experiments are conducted for only optimizers and the number of epochs, and all other values for the hyperparameters are chosen randomly. In the [50] study, the authors propose an innovative and robust model for automatically classifying brain tumors, which effectively extracts crucial features from MRI datasets. For the tuning of hyperparameters, the model is tested for only three hyperparameters such as optimizers, LRs, and epoch numbers. The values of the rest of the hyperparameters were chosen randomly. A differential deep CNN model was built in the [51] study to classify various kinds of tumors present in the brain. They extract extra differential feature maps derived from CNN's original feature maps, by integrating differential operators into the differential deep-CNN architecture. However, the authors gave no detailed information about the hyperparameters and their respective values.

In [52] study, the researchers proposed a custom CNN model combined with the manual-tunning of hyperparameters and achieve the optimal results in classifying the tumor present in the brain. The testing of this custom architecture was conducted in three scenarios by using three hyperparameters such as dropout rate, dense layer, and optimizer. A multi-level attentional approach to the identification of brain tumors was presented by the authors [53]. Spatial and cross-channel attention are merged in the suggested multi-level attention network (MANet). It maintains cross-channel temporal connections included in the semantic characteristic sequence derived from the Xception backbone in addition to giving priority to the tumor region. The values of the hyperparameters are chosen randomly, no experimental tests were conducted during the setup of the suggested model. The lightweight CNN model with learnable parameters and fewer layers was presented by researchers [54] to classify and detect brain cancers. They took several hyperparameters for testing the proposed model and obtained the optimum values of these hyperparameters. However, the results of experimental testing of these hyperparameters are not shown. Researchers presented a parallel deep convolutional neural network (PDCNN) architecture in a novel study strategy that was described in [55]. With the combined use of batch normalization and dropout regularization, this topology addresses overfitting while extracting both local and global characteristics from two concurrent phases. They focus only on architectural design and augmentation techniques not on the hyperparameters tuning and their optimal values. The authors created a unique technique [56] that uses 2D magnetic resonance imaging to detect brain tumors. This system leverages a hybrid deep learning technique with the combination of SVM (Support Vector Machine) and CNN model. While manually tuning the hyperparameters, authors test the no. of convolutional layers, LR, splitting ratio, BS, and epochs. In the [57] study, the authors proposed a scratch CNN model by using augmentation and image preprocessing techniques and achieved 100% accuracy. The values of only two hyperparameters are mentioned and ignored the others. In [58, 59] studies, the authors presented the new CNN model for the four-class classification of MRI brain tumors. Both studies not perform the experimental testing for the hyperparameters tunning, they choose the random values for some hyperparameters. Researchers classify categories of brain tumors such as gliomas, meningiomas, and pituitary in [60] using a unique CNN model. In [61], scientists presented a novel approach that combines image enhancement methods, such as CLAHE-based Adaptive Histogram Equalization and Gaussian-blur-based sharpening, to precisely classify various categories of brain tumors, like pituitary, glioma, and meningioma, and normal images. The research publication [62] presents BTC-fCNN, a quick and effective classification method. Three distinct types of brain tumors will be distinguished by a deep learning-based method: pituitary, glioma, and meningioma. TumorDetNet is an innovative end-to-end deep learning network for detecting and classifying brain tumors that was first presented in this [63] paper. No experimental results were shown during the value assignments of hyperparameters.

2.1. Research Gaps and Motivation

Despite the transparent success of applying CNNs across the identification of brain tumors, in practice, the design of well-functioning CNN models is not insignificant. Conditionally good choice hyperparameters like the convolutional layers, activation functions, optimizers, LR, BS, etc. have a high influence that how well the CNN model performs. Numerous methods for improving the CNN's tuning parameters for brain tumor identification or categorization have been documented in the literature. For example, genetic optimization algorithms [37], Grid Search [38], and Bayesian optimization [39, 40]. But, these automated optimization approaches of selecting optimal hyperparameters to hyperparameter tuning for a CNN model are often formulated as a black-box optimization (BBO) problem. In this scenario, a set of hyperparameters is mapped to a performance score using the CNN as an unknown objective function. Furthermore, due to the unknown nature of this mapping, we cannot guarantee that the optimization problem is convex. As a result, we classify the problem as of the global type.

Most researchers [2, 41-64] suggested creating bespoke CNN models and manually adjusted the hyperparameters for brain tumor detection or classification. The studies [2, 42, 43, 45-50, 53-55, 57-64] do not perform any experimental tests for obtaining the optimum values for the hyperparameters, researchers chose the random values for the selected hyperparameters. Out of these, [2, 54] studies mentioned the range of values for different hyperparameters but the experimental results are not shown for selecting the optimum values of these hyperparameters. Only a few studies have shown the experimental results for obtaining the optimum values of the hyperparameters, but researchers performed the experiments on only a few hyperparameters such as in the [41] study, researchers focused on only the number of epochs, [44] study experiments on only four hyperparameters such as PLs, AFs, optimizers, and initializers. The study [48] focuses on only the two values of activation function, the study [50] performed experiments on only optimizers, LR and epochs. The study [52] showed the experimental results on only three hyperparameters such as dropout layers, dense layers, and optimizers.

Table 1. Comparative analysis and the research gaps in hyperparameter optimization techniques used by custom CNN models for the detection /classification of MRI Brain Tumor

Reference & Year	Dataset	Classes with Size	Optimization Method	Search space for tunning of hyperparameters	Test. Acc.	Research Gaps
[37] 2018	Figshare (Cheng et al., 2017)	MEN (708) vs Glioma (1426) vs PT (930)	Genetic Algorithm	FC + DL (1, 2, 3), Conv + MP (2, 3, 4, 5, 6), Filters (16, 24, 32, 48, 64, 96, 128), Filters (2, 3, 4, 5, 6, 7) AF (ReLU, Leaky ReLU, ELU, SELU), Opt (SGD, Adam, AdaMax, Nadam, AdaGrad, AdaDelta), and FC neurons (128, 192, 256, 384, 512), DR (0.1, 0.2, 0.3, 0.4, 0.5), LR (1e-4, 1e-3, 1e-2)	94.2%	• The result for optimum values of hyperparameters from the genetic algorithm are not mentioned.
[2] 2019	Figshare (Cheng et al., 2017) REMBRANDT	MEN (708) vs Glioma (1426) vs PT (930) Grade II (205) vs Grade III (129) vs Grade IV (182)	Manual Tuning	Conv +ReLU (1, 2, 3, 4), BN (1, 2, 3), DL (1, 2, 3), FC layers (1, 2, 3), Filters (8, 16, 32, 64, 128, 256), Filter sizes (2, 3, 5, 7, 9, 10, 11), Epochs (20, 40, 50, 60, 80, 100), FC layers (1, 2, 3). PL (ML and AP), Opt (SGD, Adam, RMSProp), BS (1, 4, 8, 16, 32, 64, and 128) DR (0.1, 0.15, 0.2, 0.25, 0.5), ILR (0.01, 0.001, 0.0001), LR drop factor (0.1, 0.2, 0.3)	96.13% 98.7%	• No experimental results of hyperparameters tunning are shown. • The optimum value of only two hyperparameters is given.
[41] 2020	Radiopedia	Normal (286), MEN (380), EP (311), MB (281)	Manual Tuning	Epochs (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)	96%	• Focus on only the effect of epochs • Other hyperparameter values are chosen randomly.
[42] 2020	Kaggle (Navoneel et al., 2019	Cancerous (155) vs. non-Cancerous (98)	-	-	100%	• Randomly chosen the values of only few hyperparameters, others are missing.
[43] 2020	Figshare (Cheng et al., 2017)	MEN (708) vs Glioma (1426) vs PT (930)	-	-	97.28%	• Randomly chosen the values of only few hyperparameters, others are missing.
[44] 2020	BraTS 2018	HGG (209) vs. LGG (75)	Manual Tuning	PL (MP, AP), AF (ReLu, Selu, and Tanh), Opt (Adam, SGD), Initializer (Glorot normal and the Glorot uniform)	96.49%	• Model is tested for only 4 hyperparameters.
[45] 2020	Figshare (Cheng et al., 2017)	MEN (708) vs Glioma (1426) vs PT (930)	-	-	99%	• Values of hyperparameters are chosen randomly without any experiments
[46] 2020	Kaggle (Navoneel et al., 2019	Cancerous (155) vs. non-Cancerous (98)	-	-	96.8%	• Randomly choose the values of only a few hyperparameters and others such as learning rate, batch size, etc. are ignored.
[47] 2021	Kaggle (Navoneel et al., 2019	Cancerous (155) vs. non-Cancerous (98)	-	-	98%	• Values of hyperparameters are chosen randomly without any experiments
[48] 2021	Figshare (Cheng et al., 2017)	MEN (708) vs Glioma (1426) vs PT (930)	Manual Tuning	AF (sigmoid, tanh)	98.6%	• Model is tested with only two activation functions sigmoid and tanh.
[49] 2021	BraTS 2018	GBM (1000) vs. LGG (1000)	Manual Tuning	Opt (Adam, RMSProp, SGD), Conv +ReLU (1, 2, 3, 4, 5, 6), DL (1, 2, 3), BS (32, 64, 128), Epochs (30, 50, 100, 150, 200, 250, 300), FC layers (1, 2),	97%	• The model is tested for only optimizers and epochs hyperparameters.

				DR (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8)		• The values of some of the hyperparameters are chosen randomly and some are ignored.
[50] 2021	Figshare (Cheng et al., 2017) Radiopedia REMBRANDT	MEN (708) vs Glioma (1426) vs PT (930) MEN-I (36) vs Gliomas-II (32) vs Gliomas-III (25) vs GBM-IV (28) Normal (1041) vs Tumorous (1091) Normal (1041) vs HGG (484) vs. LGG (631) Normal (1041) vs AST (557) vs OLI (219) vs GBM (339) AST-II (356) vs AST-III (201) vs OLI-II (128) vs OLI-III (91) vs GBM-IV (339) Normal (1041) vs AST-II (356) vs AST-III (201) vs OLI-II (128) vs OLI-III (91) vs GBM-IV (339)	Manual Tuning	Opt (Adam, SGD, Adadelta, Adagrad), LR (0.01, 0.001, 0.002, 0.003, 0.004), and Epochs (10, 20, 30)	94.74% 93.71% 100% 97.22% 97.2% 88.86% 95.72%	• The model is tested for optimizer, learning rate, and epochs and rest are chosen randomly.
[51] 2021	Tianjin Universal Center of Medical Imaging and Diagnostic (TUCMD)	Metastasis vs MEN vs Glioma vs Astrocytoma vs Germ cell vs Craniopharyngiomas	-	-	99.25%	• Only the activation function is given. • No detailed information about the hyperparameters is given.
[39] 2021	Figshare (Cheng et al., 2017)	MEN (708) vs Glioma (1426) vs PT (930)	Bayesian optimization	Filters (16, 32, 64, 128, 256, 512), DR (0, 0.5), Filter Size (2, 3, 4, 5), MP Size (2, 3, 4, 5), Filter (16, 32, 64, 128, 256, 512, 1024, 2048, 4096)	97.37%	• Only the architectural hyperparameters are used for tunning by optimization method. • Obtained optimum values of hyperparameters are not mentioned.
[52] 2021	Kaggle (S. Bhuvaji 2020)	MEN (937) vs Glioma (926) vs PT (901) vs Normal (500)	Manual Tuning	DR (0.2, 0.5), Opt (Adam, RMSProp, Adamax, SGD), Dense layer (1024, 512)	96%	• Hyperparameters tuning is based on only three hyperparameters dropout, dense layers, and optimizers and the values of rest are chosen randomly
[38] 2021	RIDER, REMBRANDT, TCGA-LGG, Figshare (Cheng et al., 2017)	Tumor (1640) vs non-tumor (1350) Normal (850) vs Glioma (950) vs MEN (700) vs PT (700) vs Metastatic (750) Grade-II (1676) vs Grade-III (1218) vs Grade-IV (1676)	Grid Search Optimization	AF (ReLU, ELU, SELU, Leaky ReLU), Conv and MP (1, 2, 3, 4), BS (4, 8, 16, 32, 64), FC layers (1, 2, 3, 4), Filters (16, 24, 32, 48, 64, 96, 128), Filter size (3, 4, 5, 6, 7), Momentum (0.80, 0.85, 0.90, 0.95), LR (0.0001, 0.0005, 0.001, 0.005), and L2 Regularization (0.0001, 0.0005, 0.001, 0.005).	99.33% 92.66% 98.14%	• Optimizers and epochs are missing during grid search optimization.
[53] 2022	Figshare (Cheng et al., 2017) BraTS 2018	MEN (708) vs Glioma (1426) vs PT (930) HGG (210) vs. LGG (75)	-	-	96.5% 94.91%	• Randomly choose the values of only a few hyperparameters and some are ignored.

[40] 2022	Figshare (Cheng et al., 2017)	MEN (708) vs Glioma (1426) vs PT (930)	Bayesian optimizatio n	AF (ReLU, ELU, Sigmoid, SELU, Tanh), BS (1 to 128), DR (0.1 to 0.5), Dense nodes (32 to 1024), Opt (Adam, Nadam, AdaMax, RMSProp, SGD)	98.70%	• Learning rate and epochs are not included in the optimization method.
[54] 2022	Figshare (Cheng et al., 2017)	MEN (708) vs Glioma (1426) vs PT (930)	Manual Tuning	Image Size (128×128, 227×227), Conv Layer + ReLU (1, 2, 3), PL (MP, AP), Cross channel normalization layer(0, 1, 2), BN (1, 2, 3), DL (1, 2), FC Layer (1, 2), Grouped Conv Layers (0,1, 2), DR (0.25, 0.5), Opt (Adam, SGDM), LR (0.0001, 0.0002, 0.0003, 0.0005), Epochs (10, 20, 30, 40), BS (4, 8, 10), Filters (32, 64, 128, 256), Filter size (2, 3, 5)	97.2%	• The results of the experimental tests on hyperparameters are not shown. • No experimental results about getting the optimum values of the hyperparameters are shown.
[55] 2022	Kaggle (S. Bhuvaji 2020)	MEN (937) vs Glioma (926) vs PT (901) vs Normal (500)	-	-	99%	• Values of some hyperparameters are chosen randomly without any experiments and some are ignored.
[56] 2023	Kaggle (Navoneel et al., 2019) Figshare (Cheng et al., 2017) Kaggle (S. Bhuvaji 2020)	Tumour (155) vs. Normal (98) MEN (708) vs Glioma (1426) vs PT (930) MEN (937) vs Glioma (926) vs PT (901) vs Normal (500)	-	-	97.33% 97.60% 98.12%	• Only the value of activation function chosen randomly and rest are ignored
[57] 2023	Kaggle (Navoneel et al., 2019	Cancerous (155) vs. non-Cancerous (98)	Manual Tuning	Conv Layers (5, 6, 7), Training -Testing ratio (70:30, 80:20), LR (0.001, 0.005, 0.01), BS (16, 32), epochs (8, 9, 10, 11)	97.86%	• Only few hyperparameters are tuned and other such as activation function, optimizers etc. are ignored
[58] 2023	Kaggle (Navoneel et al., 2019	Cancerous (155) vs. non-Cancerous (98)	-	-	100%	• Randomly chosen the values of only two hyperparameters and others are ignored.
[59] 2023	Kaggle (S. Bhuvaji 2020)	MEN (937) vs Glioma (926) vs PT (901) vs Normal (500)	-	-	93.30%	• Randomly assigned the values of few hyperparameters and others are ignored.
[60] 2023	Figshare (Cheng et al., 2017), Kaggle (S. Bhuvaji 2020), and Kaggle dataset (Br35H)	MEN vs Glioma vs PT vs Normal	-	-	95.44%	• Randomly chosen values, no experimentation.
[61] 2023	Figshare (Cheng et al., 2017)	MEN (708) vs Glioma (1426) vs PT (930)	-	-	98.04%	• Values of hyperparameters are chosen randomly without any experiments
[62] 2023	Figshare (Cheng et al., 2017), Kaggle (S. Bhuvaji 2020), and Kaggle dataset (Br35H)	MEN vs Glioma vs PT vs Normal	-	-	97.84%	• Values of hyperparameters are chosen randomly without any experiments
[63] 2023	Figshare (Cheng et al.,	MEN (708) vs Glioma (1426) vs PT (930)	-	-	98.86%	• Values of hyperparameters are chosen randomly without any experiments

	2017)					
[64] 2023	Kaggle dataset (Br35H) Kaggle (Navoneel et al., 2019) Kaggle Dataset (Alif Rahman) Kaggle Dataset (Prajakta Sabale) Kaggle dataset (SARTAJ) Figshare (Cheng et al., 2017)	Tumorous (1500) and non-Tumorous (1500 Tumour (155) vs. Normal (98) Benign (350) vs Malignant (350) Benign (1200) vs Malignant (200) MEN (937) vs Glioma (898) vs PT (926) MEN (708) vs Glioma (1426) vs PT (930)	-	-	99.83% 96.08% 100% 100% 99.27% 98.47%	• Values of hyperparameters are chosen randomly without any experiments
[65] 2024	Kaggle (Figshare, SARTAJ, and Br35H)	MEN (1645) vs Glioma (1621) vs PT (2000) vs Normal (1757)	-	-	99%	• Values of hyperparameters are chosen randomly
[66] 2024	TCIA (REMBRANDT)	Metastasis, Glioma, and Meningiomas	-		98%	• Values of hyperparameters are chosen randomly
[67] 2024	Figshare, SARTAJ, and Br35H Kaggle Navoneel et al. Kaggle dataset (Br35H)	MEN (1645) vs Glioma (1621) vs PT (2000) vs Normal (1757) Tumour (155) vs. Normal (84) Tumorous (1500) and non-Tumorous (1500)	Manual Tuning	learning rate, batch size, number of epochs, optimizer, shuffle, verbose, dropout rate, filters, filter size, and activation function.	97.18% 93% 96%	• Never describe the range of hyperparameters for tuning
[68] 2024	BraTS 2020	Tumorous and non-Tumorous	Manual Tuning	Activation Function and Optimizers	92.59%	• Only two hyperparameters are tunned
[69] 2025	Kaggle (Figshare, SARTAJ and Br35H)	MEN (1645) vs Glioma (1621) vs PT (2000) vs Normal (1757)	-	-	99%	• Values of hyperparameters are chosen randomly
[70] 2025	Kaggle (Figshare, SARTAJ and Br35H)	MEN (1645) vs Glioma (1621) vs PT (2000) vs Normal (1757)	-	-	99.64%	• Values of hyperparameters are chosen randomly

MEN: Meningioma, PT: Pituitary, HGG: High Grade Glioma, LGG: Low Grade Glioma, AST: Astrocytomo, GBM: Glioblastoma Multiforme, OLI: Oligodendroglioma, AF: Activation Function, LR: Learning Rate, Opt: Optimizer, BS: Batch Size, DL: Dropout Layer, DR: Dropout Rate, PL: Pooling Layer, BN: Batch Normalization, FC: Fully Connected, ILR: Initial Learning Rate, MP: Max Pooling, AP: Average Pooling

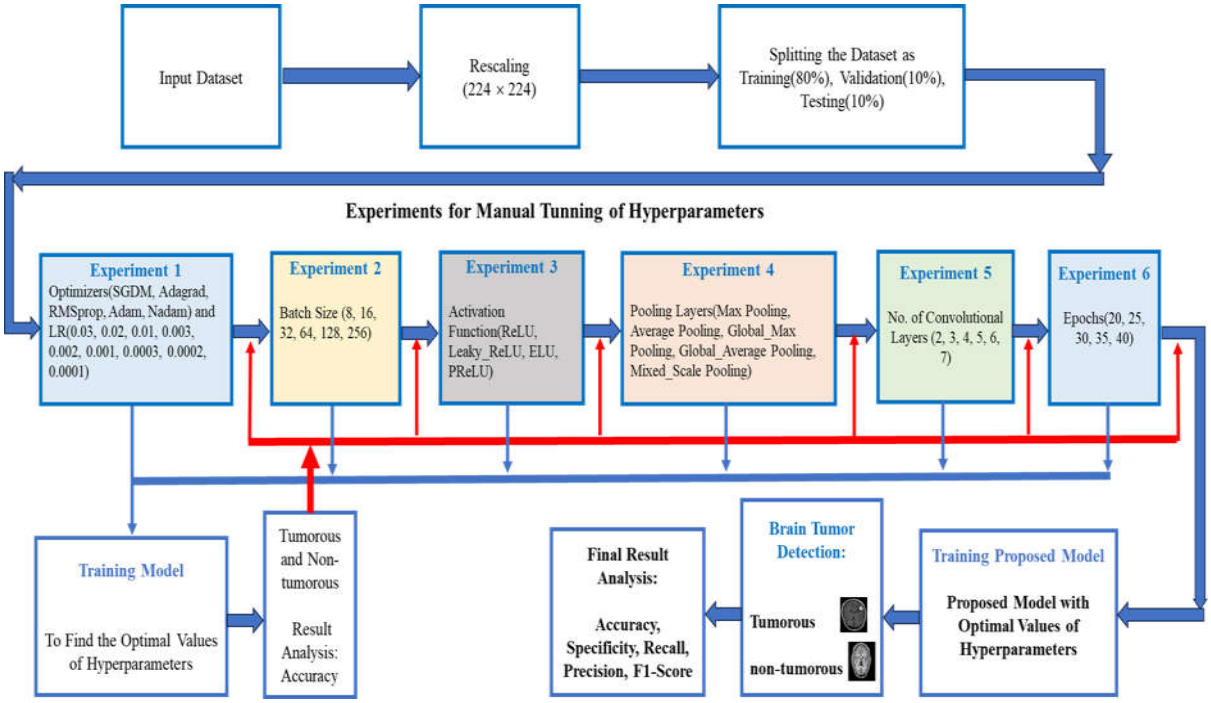

Fig. 3. The complete workflow of the proposed methodology for the experimental setup to manually tune hyperparameters for building a custom MTBTDet_CNN.

3. Research Methodology

Figure 3 shows the proposed methodology for manually tuning the hyperparameters in a custom CNN model, where different experiments are accomplished to obtain the ideal hyperparameter values. The workflow starts with the input dataset. and rescaled to a dimension of 224 × 224 pixels. To be set up for model training and assessment, the dataset is then separated into testing (10%), validation (10%), and training (80%) sections. In the next pipeline, six experiments are designed for the manual tunning of seven hyperparameters (convolutional layers, types of PLs, AFs, gradient optimizers, LRs, BS and epochs) by training the CNN models. The result analysis focuses on the accuracy metric to identify which combination of hyperparameters yields the best outcomes and to get the optimal results of these hyperparameters for the proposed model. Experiment 1 (section 4.1) focuses on determining the best optimizer and LR among 5 optimizers (SGDM, Adagrad, RMSprop, Adam, Nadam) and 9 values of LRs (0.03, 0.02, 0.01, 0.003, 0.002, 0.001, 0.0003, 0.0002, 0.0001) by train the model 45 times. This experiment's goal is to determine the pair of optimizers and the LR that produces the best accuracy. Experiment 2 (section 4.2) uses the best optimizer and LR obtained from experiment 1 and examines the effect of different BSs (8, 16, 32, 64, 128, and 256) depending on how well the model performs. The motive of this experiment is to obtain the optimal BS which leads to the best accuracy and most efficient training. Optimal LR, optimizer and BS are used in experiment 3 (section 4.3) to test various AFs (ReLU, Leaky_ReLU, ELU, and PReLU), and determine the favorable AF which improves convergence and model performance. Experiment 4 (section 4.4) utilizes the optimal parameters of the optimizer, LR, BS, and AF to test the various PLs, which include Global_Max Pooling, Global_Average Pooling, Mixed_scale Pooling, and Max Pooling. The objective of this experiment is to determine which pooling strategy enhances model performance in reducing dimensionality while retaining important features. In experiment 5 (section 4.5), the convolutional layers which are modified to determine the most efficient depth of the network are determined using the ideal hyperparameters that were obtained from the above experiments. Finding the ideal number of layers that results in the maximum accuracy is the aim. The number of epochs that yield the highest model performance without overfitting was determined in experiment 6 (section 4.6) by utilizing all the optimal hyperparameters from experiments 1 through 5.

After completing all experiments, the proposed model is developed using the optimal hyperparameters obtained from the experiments. Based on performance evaluation measures along with accuracy, specificity, recall, precision, and f1-score, this model is predicted to produce the best results.

3.1. Dataset Description

The models with different hyperparameters are trained and tested by using brain tumor MRI dataset that is freely accessible on Kaggle. The dataset, organized and maintained by Naveenprakasha [71] consists of 3,000 JPG-formatted human brain MRI scans, evenly split into two classes i.e., 1,500 positive cases (brain tumor) and 1,500 negative cases (no tumor). Figure 4 illustrates the

MRI images for binary classes i.e., 'no' and 'yes' of the dataset. Normalizing the degree of intensity data enhances the performance of later models and algorithms, and rescaling the images to a defined resolution (224×224×3) ensures uniformity throughout the collection. The ideal dataset splitting ratio for the custom CNN model's training and assessment for training, validation, and testing is 80:10:10 respectively [72]. Table 2 provides a thorough explanation of the dataset.

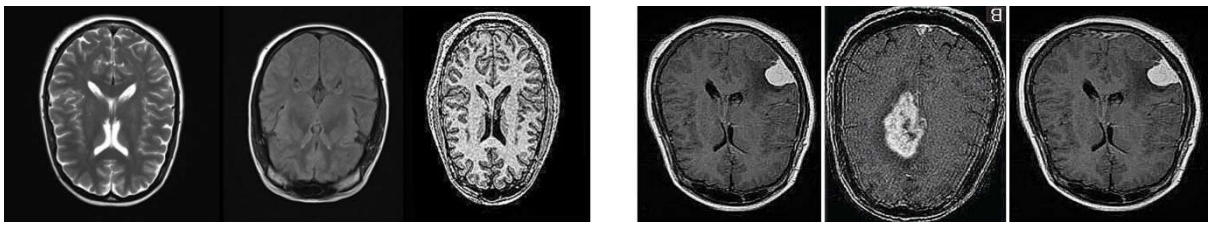

Fig. 4. Non-tumorous left) and tumorous (right) MRI images of the human brain from the dataset.

Table 2. Details and distribution of the dataset of MRI scans of brain tumors for training, validation, and testing.

Class Name	No. of Images	Training	Testing	Validation
Yes (Brain Tumor)	1500	1200	150	150
No (No Tumor)	1500	1200	150	150
Total	3000	2400	300	300

3.2. Hyperparameters Choices

Although CNNs are powerful, their effectiveness and accuracy depend on parameter selection. When choosing CNN parameters, it is common to use an optimal combination of several parameters. CNN models are intricate architectures that involve numerous hyperparameters. In general, these hyperparameters fall into two categories: fine-tuning hyperparameters and architectural hyperparameters. The range of convolutional PLs, fully connected layers, filters, filter sizes, and AFs are examples of architectural hyperparameters. However, optimizers, BS, LR, L2 regularization, and other factors are included in the fine-tuning of hyperparameters. Seven hyperparameters, as indicated in Table 3, are manually adjusted in this article. These include the number of convolutional layers, AFs, PLs, gradient optimizers, LR, BS and epochs.

Table 3. Ranges of the hyperparameters used to train the proposed CNN model.

Hyperparameters	Value
Gradient-Optimizers	SGDM, Adagrad, RMSprop, Adam, and NAdam
Learning Rate	0.03, 0.02, 0.01, 0.003, 0.002, 0.001, 0.0003, 0.0002, 0.0001
Batch Size	8, 16, 32, 64, and 128
Activation Function	ReLU, Leaky_ReLU, ELU, and PReLU
Number of Convolutional layers	1, 2, 3, 4, 5, and 6 (excluding input conv_layer)
Pooling Layers	Max Pooling, Global_Max Pooling, Average Pooling, Global_Average Pooling, and Mixed_scale Pooling
Epochs	20, 25, 30, 35, and 40

4. Results and Discussions

Six experiments are conducted to construct the proposed MTBTDet_CNN model on seven hyperparameters (Optimizers, LRs, BS, AFs, No. of Conv. Layers, PLs, and Epochs). The result of each experiment has been examined independently based on the accuracy metric to determine the ideal values of each hyperparameter. The ideal value of the hyperparameter from experiment 1 transfers to experiment 2 and so on. Hence, after six experiments, we get the optimal values of seven hyperparameters. The model is repeatedly trained and assessed in each experiment. (depending on the values of the hyperparameters) by using the dataset.

4.1. Experiment 1: For Optimal Optimizer and Learning Rate

The LR and optimizer are two of the most influential hyperparameters in training deep neural networks, as they directly affect the model's convergence speed, stability, and final accuracy. The optimizer determines how the model's weights are updated during backpropagation, while the LR controls the size of these updates. Their performance is highly interdependent - an inappropriate LR can hinder even the best optimizer, leading to slow convergence or unstable training. Therefore, this experiment was designed to systematically evaluate the combined effects of different optimizers (SGDM, Adagrad, RMSprop, Adam, and Nadam) with varying LR (0.03, 0.02, 0.01, 0.003, 0.002, 0.001, 0.0003, 0.0002, 0.0001) to identify the most effective configuration for achieving optimal accuracy.

When training a CNN model, the coordination of optimizers and LRs is essential since it influences how the model learns and converges on a solution. The LR determines the magnitude of these updates, whereas the optimizer handled the model's weight updates during training. For the model to function at its best, the two must collaborate well. The optimizers with LRs combined and hence the CNN model is trained and tested 45 times to obtain the best combination of LR and optimizer based on accuracy. Other hyperparameter values are preset, including BS (32), no. of convolutional layers (4), PL (max pooling), AF (ReLU), and epochs (20). The Figure 5 shows the accuracy in the bar graph of trained models and the figure shows the accuracy in the line graph over the epochs. The bar graph shows that the SGDM, Adagrad, Adam, and Nadam models perform almost similarly across LRs, while RMSprop shows more variation in performance across LRs and the LRs with the best performance vary across optimizers, indicating that fine-tuning the learning rate is critical for each optimizer. For most optimizers and LRs, the test accuracy is almost the same and it is challenging to determine the ideal value for the optimizer and LR, therefore Table 4 is constructed to find the optimal result and shows that the Adagrad, RMSprop, and Adam optimizers and 0.002, 0.0003, and 0.0001 LRs having accuracy greater than 0.985 in more models as compared to others. Adam optimizer showed two models (Adam_0.002 and Adam_0.0003) with the highest accuracy (0.993). But as shown in Figure 6, the loss function of the Adam_0.0003 model is 0.027 and the Adam_0.002 model has 0.037.

Table 4. Comparison of accuracies of CNN models with different values of LRs and optimizers to find the perfect combination of LR and optimizer.

Accuracies with combinations of LRs and Optimizers						
LR	**SGDM**	**Adagrad**	**RMSprop**	**Adam**	**Nadam**	**LR with Optimizers (Accu. >= 0.985)**
0.03	**0.987**	**0.993**	0.953	0.979	0.967	02
0.02	0.983	**0.993**	0.980	0.980	0.950	01
0.01	**0.987**	**0.993**	0.970	0.980	0.927	02
0.003	0.983	0.977	0.980	0.960	0.953	00
0.002	**0.993**	**0.990**	0.973	**0.993**	0.947	**03**
0.003	0.983	0.980	**0.987**	0.983	**0.990**	02
0.0003	0.983	0.983	**0.990**	**0.993**	**0.987**	**03**
0.0002	0.983	0.947	**0.990**	**0.987**	0.983	02
0.0001	0.980	0.930	**0.993**	**0.990**	**0.990**	**03**
Optimizer with LRs (Accu. >= 0.985)	03	**04**	**04**	**04**	03	

Table 4 compares the performance of CNN models using different optimizers across a range of LRs to determine the optimal configuration. The results indicate that the model's accuracy is highly dependent on both the choice of optimizer and the LR. Adagrad achieved consistently high accuracies at relatively higher LRs (0.01–0.03), demonstrating its ability to adaptively adjust parameter updates. Adam and RMSprop, on the other hand, showed superior performance at lower LRs (≤ 0.003), with Adam achieving the highest accuracy (0.993) at 0.002, 0.0003, and 0.0001, highlighting its robustness and stability during optimization. SGDM exhibited stable performance across all LRs, peaking at 0.002 (0.993), while Nadam performed comparably to Adam but showed slightly higher sensitivity to LR changes. Overall, Adam emerged as the most reliable optimizer for lower LRs, whereas Adagrad demonstrated optimal convergence at higher LRs, suggesting that the interaction between LR and optimizer plays a critical role in maximizing CNN classification performance.

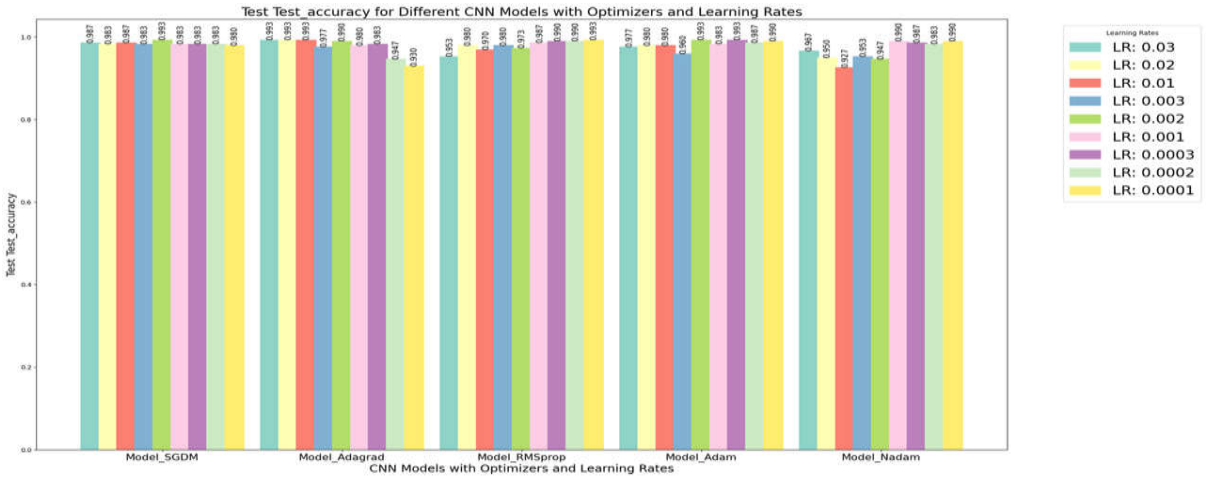

Fig 5. Bar graph of test accuracy for different CNN models with optimizers and LRs in which each group of bars represents different LR for each optimizer and color-coded bars show the test accuracy corresponding to each LR (0.03, 0.02, etc.).

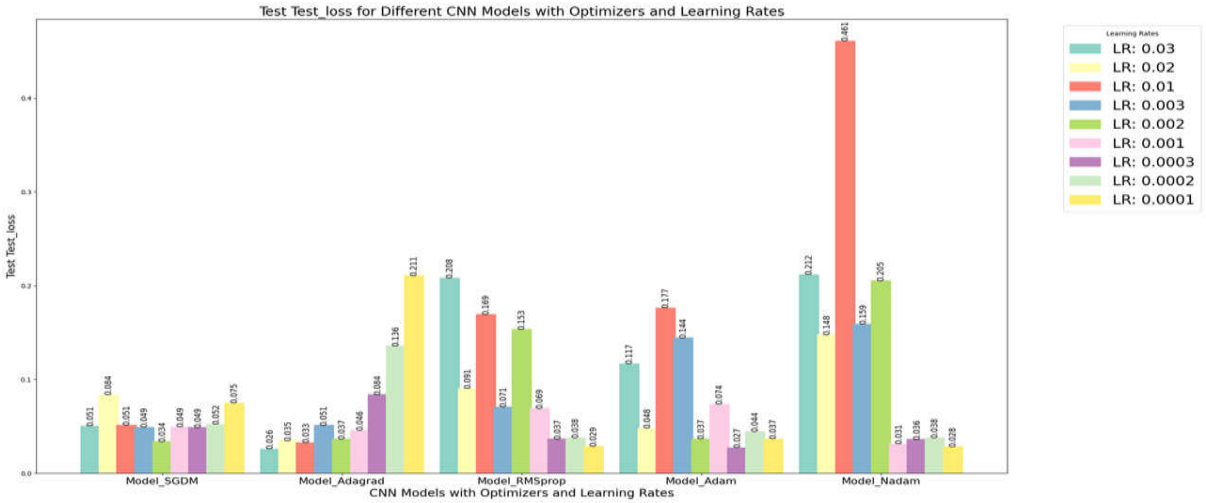

Fig 6. Bar graph of test loss for different CNN models with optimizers and LRs in which each group of bars represents different LRs for each optimizer and color-coded bars show the test loss corresponding to each LR (0.03, 0.02, etc.).

Therefore, the combination of the Adam optimizer with a LR of 0.0003 is identified as the most effective configuration for further experiments, providing an optimal balance between convergence speed, training stability, and classification accuracy.

4.2. Experiment 2: For Optimal Batch Size

BS is an important hyperparameter in training a CNN model, as it affects the learning process, model interpretation, and computational efficiency. This experiment demonstrates how various batch sizes (8, 16, 32, 64, 128) influence the test accuracy of a CNN model as shown in Figure 7 with Adam optimizer and learning rate 0.0003 as obtained from the result of experiment 1. Smaller BS (8 and 16) perform well, showing competitive accuracy and quick convergence. However, they exhibit more fluctuations early on. Larger BSs (64 and 128), especially BS_128, perform poorly, with BS_128 severely impacting the model's ability to learn effectively. This is possibly due to the model's slower updates when using larger batches, resulting in lower performance and less flexibility in learning. BS 32 performs best, achieving the highest test accuracy and quickly stabilizing with minimal fluctuation. This suggests it is the optimal BS for the given CNN architecture and learning rate. The batch size significantly influences both the model's stability and final performance, with medium batch sizes (like 32) typically offering a balance between stability, speed, and accuracy.

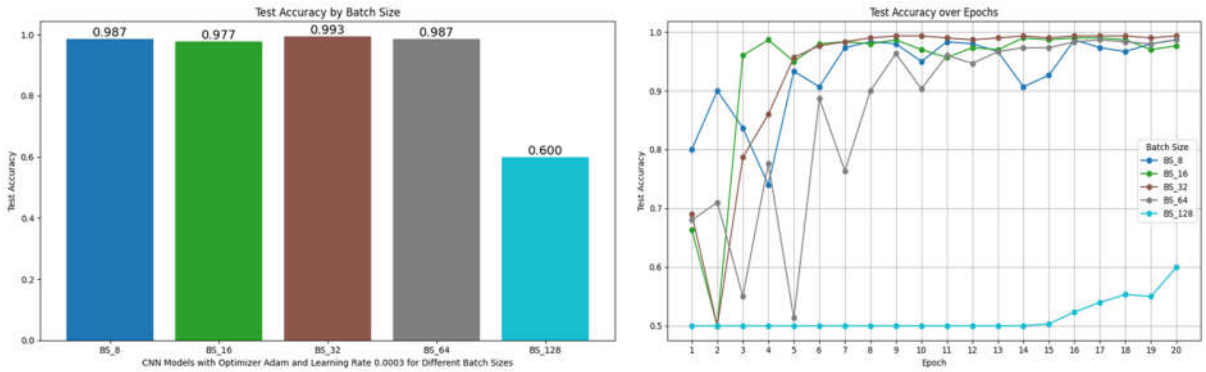

Fig. 7. The Bar graph compares the test accuracy for CNN models trained with different BSs (Left) and Line Graph tracks the test accuracy over 20 epochs for each BS (Right).

4.3. Experiment 3: For Optimal Activation Function

AFs are critical components in CNNs as they assess the network's capacity to learn complex patterns and make decisions. The AF adding non-linearity into the model, makes it possible to handle more challenging tasks beyond linear classification or regression. This experiment helps to determine the most appropriate activation function based on our model's requirements and elaborates on how the various activation functions can influence the test accuracy of the CNN models. Figure 8 depicts the results of final test accuracies for different AFs (ReLU, Leaky_ReLU, ELU, PReLU) in the format of a bar graph and a line graph. The bar graph demonstrates that ReLU is better than other functions in terms of final accuracy, making it the outperforming AF for this specific CNN model based on test accuracy. However, ELU performs closely to ReLU, and Leaky ReLU still performs well, while PReLU

has the lowest accuracy among the four. The line plot reveals that ELU and Leaky ReLU demonstrate faster convergence, meaning they achieve high accuracy earlier in training. ReLU and PReLU take a bit longer to stabilize but ReLU ultimately achieves the highest accuracy. Hence, ReLU is the most optimal choice in terms of achieving the highest test accuracy.

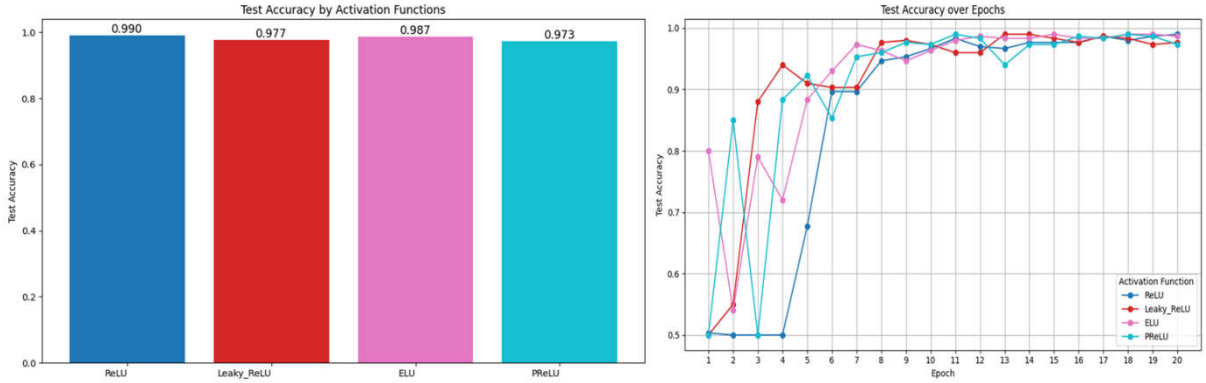

Fig. 8. The bar chart compares the final test accuracies achieved using different AFs (Left) and the line graph shows how test accuracy evolves over 20 epochs for each AF.

4.4. Experiment 4: For Optimal Pooling Layer

CNNs rely heavily on PLs to carry out several critical tasks, including feature extraction, dimensionality reduction, translation invariance, and overfitting prevention. In general, pooling layers are necessary to improve CNN efficiency and performance. This experiment highlights the importance of choosing the right pooling layer depending on the specific problem by performing the comparisons between the test accuracies of different PLs (Max Pooling, Global_Max Pooling, Average Pooling, Global_Average Pooling, Mixed_scale Pooling) as shown in Figure 9 Average Pooling, Max Pooling, and Mixed Scale Pooling consistently perform best, with nearly identical final accuracies (0.993). These methods effectively downsample feature maps while retaining important information. These PLs also show rapid convergence and reach near-perfect accuracy after just a few epochs. Global Max Pooling performs reasonably well but does not reach the same accuracy as other pooling methods and shows slower convergence. This might be happening due to the possibility of losing important data when simplifying each feature map to a single value. Global Average Pooling underperforms and shows slower convergence, likely because averaging all values across the entire feature map can result in excessive smoothing, losing important details, and reducing model performance. The value of the loss function for the various pooling layers such as Max Pooling, Average Pooling, and Mixed Scale Pooling, is displayed in Figure 10 Max pooling gives the least value i.e., 0.040 as compared to the Average pooling (0.044), and Mixed Scale pooling (0.048). Hence the ideal value of the pooling layer is the 'Max Pooling.'

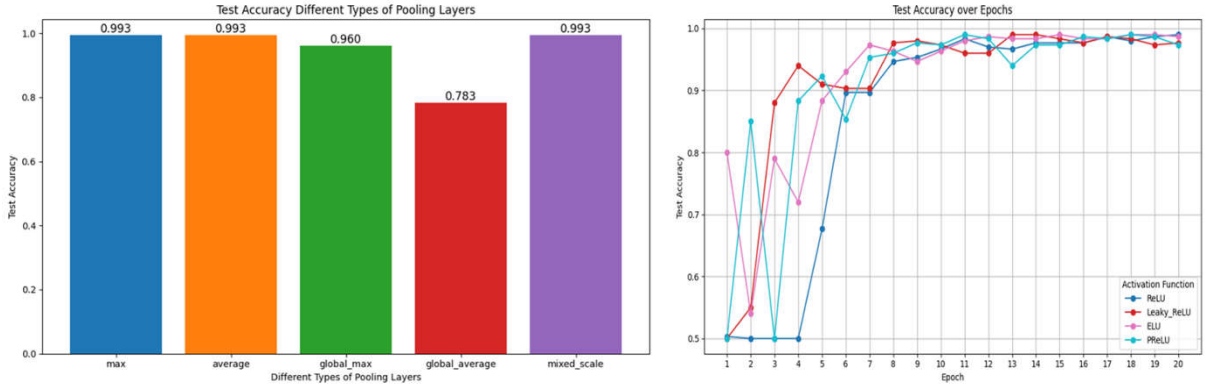

Fig 9. The bar graph compares the final test accuracy for CNN models with different types of PLs (Left) and the line graph shows how the model's accuracy changes throughout 20 epochs for each PL.

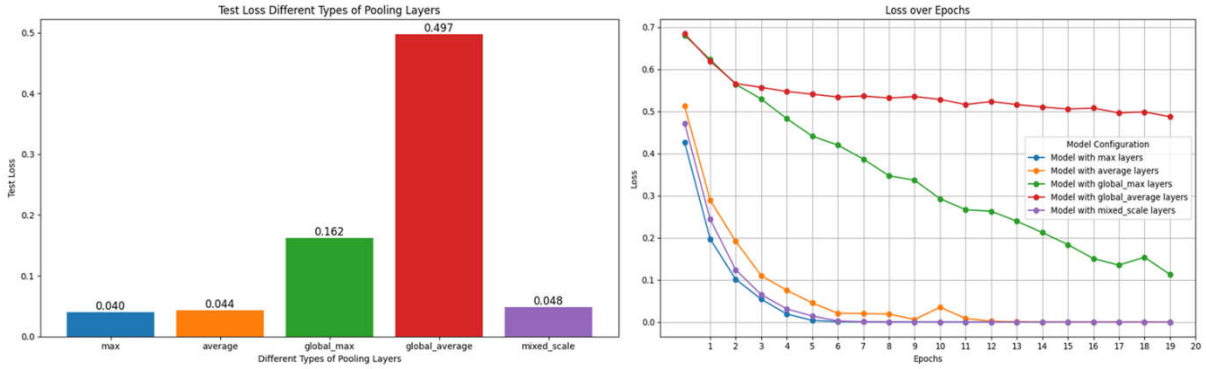

Fig. 10. The bar graph compares the final test loss for CNN models with different types of pooling layers (Left) and the line graph shows how the model's loss function changes throughout 20 epochs for each pooling layer.

4.5. Experiment 5: For Optimal Number of Convolutional Layers

The number of convolutional layers in a CNN is a crucial architectural hyperparameter that impacts the model's performance, capacity, and generalization. Generally, very less layers may cause underfitting (unable to capture complex patterns), and too many layers may lead to overfitting (memorizing the training data) therefore, finding the ideal number of layers is important to ensure that the model can extract meaningful features without over-complicating the architecture. This experiment is often used to determine how many layers are ideal for a certain task and dataset, results are shown in Figure 11 based on test accuracy. The bar graph shows that the model with 3 convolutional layers achieves the highest accuracy (0.993), closely followed by the model with 2 layers. Models with 4 and 5 layers have slightly lower accuracy (0.990) and the model with 6 layers has the lowest test accuracy (0.930), suggesting that adding the number of layers beyond 5 may cause to overfitting or diminished returns. The line graph shows that all other models (1–5 layers) follow similar trends, with the models having 2 and 3 layers converging faster and achieving slightly higher final accuracy. So, the CNN with 2 or 3 layers provides the best accuracy and convergence speed. But the value of test loss with 2 convolutional layers is 0.026 while 0.013 with 3 convolutional layers is depicted in Figure 12. Hence, the optimal CNN with 3 Convolutional layers (excluding the input convolutional layer) provides the best accuracy, least test loss, and convergence speed.

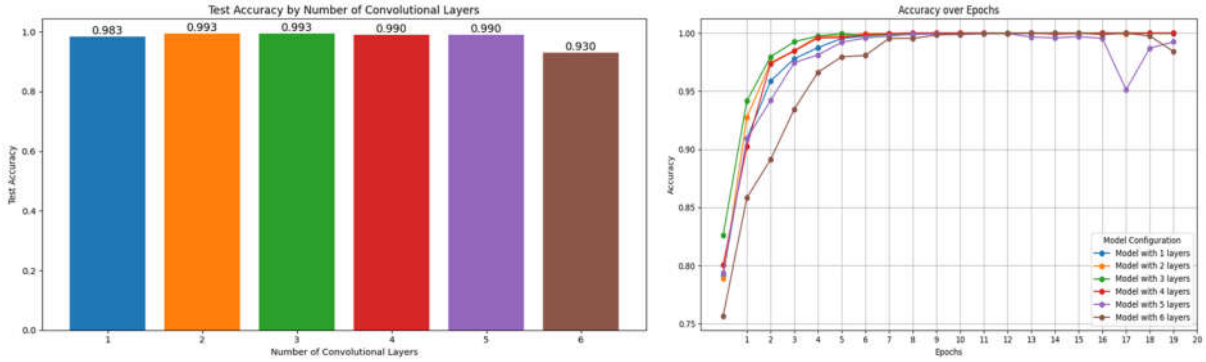

Fig. 11. The bar chart shows the test accuracy achieved by models with varying numbers of convolutional layers (1 to 6, excluding input conv layer) (Left) and the line plot shows how accuracy evolves over 20 epochs for each model configuration (from 1 to 6 convolutional layers).

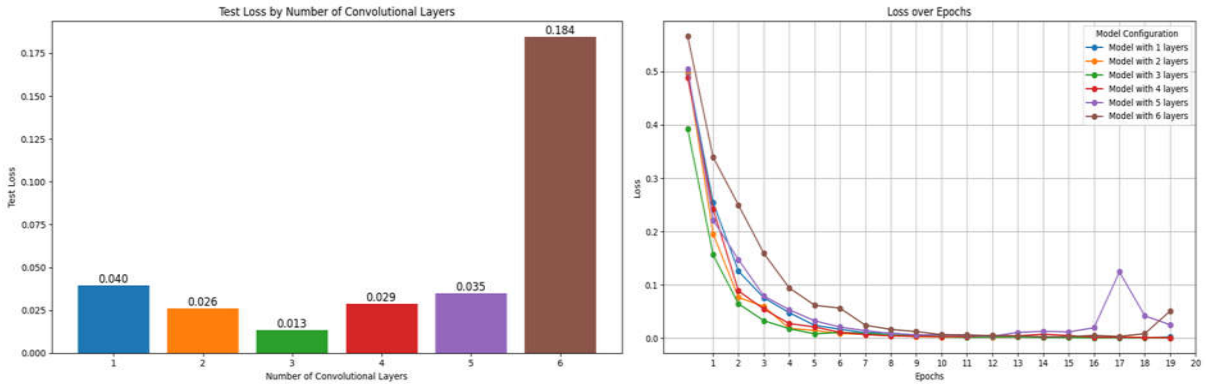

Fig. 12. The bar chart shows the test loss achieved by models with varying numbers of convolutional layers (Left) and the line plot shows how loss evolves over 20 epochs for each model configuration (from 1 to 6 convolutional layers).

4.6. Experiment 6: For Optimal Number of Epochs

One crucial hyperparameter in CNN model training is the number of epochs that influence the model's learning process. One whole run through the training dataset is referred to as an epoch, which enables the model to modify its weights in response to errors observed during each pass. The test accuracy of trained CNN models for different numbers of epochs (20, 25, 30, 35, 40) is shown as a bar graph and line plot in Figure 13. The bar graph depicts that the highest test accuracy is 99.7% for models trained with 20 and 25 epochs. The test accuracy starts to slightly decrease after 30 epochs, indicating potential overfitting or stabilization of performance, as further training does not significantly improve accuracy. Training beyond 25 or 30 epochs does not result in any meaningful improvement and may indicate that 20-25 epochs are sufficient for this task. Thus, these graphs suggest that extending training beyond 25 epochs offers diminishing returns, and using a lower number of epochs may be more efficient for this CNN model. Hence the 20 epochs is the optimal number of epochs.

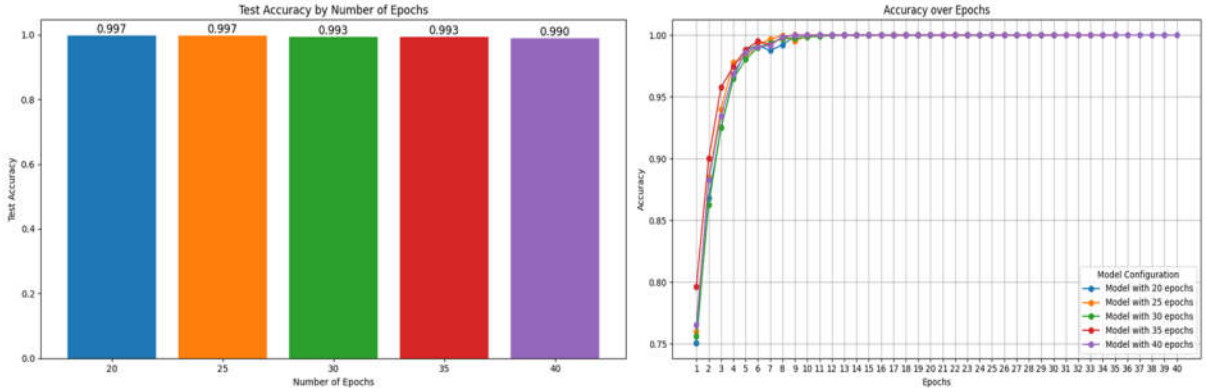

Fig. 13. The Bar chart presents the final test accuracy achieved by CNN models trained for different numbers of epochs (20, 25, 30, 35, 40) (left) and the line plot shows how the accuracy changes throughout Training (from 1 to 40 epochs).

4.7. Proposed MTBTDet_CNN using Optimal Hyperparameters

The optimal values of the hyperparameters which are tunned experimentally are depicted in Table 5. Adam is the optimal optimizer for the proposed model. Because of its variable LR and momentum characteristics, which make it perfect for complicated neural networks like CNN, it is commonly employed in deep learning. The LR of 0.0003 was found to be the most effective for training. A lower LR ensures the model converges smoothly without overshooting the optimal weights. For the training procedure, a BS of 32 provided an ideal compromise between model performance and computational economy. ReLU was chosen as the AF because of its effectiveness and simplicity of use in managing non-linearities, as well as its ability to lessen the probability of the vanishing gradient issue. The model is made up of 3 convolutional layers (excluding the input layer), that were found to be the ideal depth for capturing pertinent characteristics of the input data. Max pooling is used to downsample, which decreases the feature maps' spatial dimensions while keeping the most crucial information. Twenty training epochs are enough to train the model to a great level of accuracy without overfitting. The architectural design of the proposed MTBTDet_CNN model by using these optimal hyperparameters is shown in Figure 14.

Table 5. Optimal outcomes of the hyperparameters obtained from the experiments for training the proposed CNN model.

Hyperparameters	Optimal Value
Gradient-Optimizers	Adam
Learning Rate	0.0003
Batch Size	32
Activation Function	ReLU
Number of Convolutional layers	4 (including input conv_layer)
Pooling Layers	Max Pooling
Epochs	20

The 16-layered architecture shown in Figure 14 represents the design of the proposed MTBTDet_CNN model for the identification of brain tumors using MRI. The model follows a deep learning architecture using convolutional layers, batch normalization, activation functions, and pooling layers to detect MRI brain images as "Tumorous" or "Non-Tumorous." The model is made up of 4 convolutional layers encompassing the input layer (the result of experiment 5) having 32, 64, 64, and 128 filters of size 3 3, which are responsible for selecting features out of the input images. The ReLU activation function (the outcome of experiment 3) introduces non-linearity by following each convolutional layer, enabling the model to recognize intricate patterns. After each convolutional layer + ReLU, max pooling (the result of experiment 4) is applied with a 2×2 filter to minimize the feature maps' spatial dimensions while maintaining crucial features and cutting down on computational complexity. After each max pooling,

batch normalization is applied to normalize the activations, helping to enhance model performance and speed up the training procedure. The feature maps are flattened into a 1D vector in the model's thirteenth layer, known as the "flatten layer," which provides the input for the fully connected layers. There are two fully connected dense layers with 128 units following the flattening step, which helps combine the features learned by the convolutional layers. ReLU activation is used in the first dense layer, and the sigmoid AF is utilized in the second dense layer, which generates probabilities for the binary classification job: "Tumorous" or "Non-Tumorous." Dropout is applied between the two dense layers with a rate of 0.5 to avoid overfitting during training by randomly dropping half of the neurons.

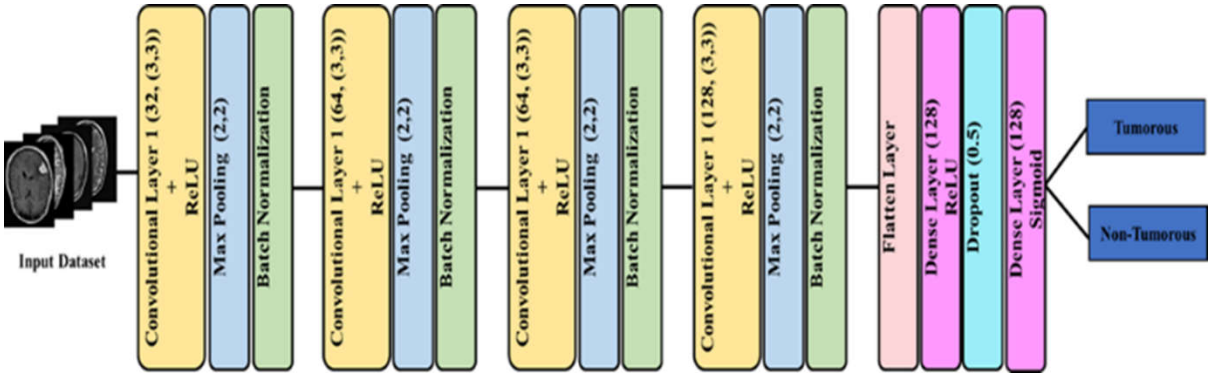

Fig. 14. The architecture of the proposed MTBTDet_CNN model having 4 convolutional layers with ReLU activation function and max pooling layer for the detection of MRI brain tumor images.

Training the proposed model by using the ideal values of the hyperparameters such as Adam optimizer and 0.0003 LR (the result of experiment 1), 32 BS (the result of experiment 2), and 20 epochs (the result of experiment 6). The results of the trained model according to the performance metrics (accuracy, recall, F1 score, precision, and specificity) are demonstrated in Figure 15. The model's accuracy, which indicates its overall correctness, is 0.997. Recall, sometimes referred to as true positive rate or sensitivity, is a statistic that indicates how well the model can find all relevant cases. The model correctly detects all positive occurrences when the recall is 1.000. The precision metric calculates the percentage of positive cases that were accurately predicted. With a precision of 0.993, the model was accurate in 99.3% of the positive predictions it made. The F1 score is the harmonic mean of precision and recall. The result of 0.997 suggests that the model is correctly balanced between recall and precision. Specificity gauges how well the model can recognize negative examples. With a 0.993 score, 99.3% of negative cases were properly detected by the model.

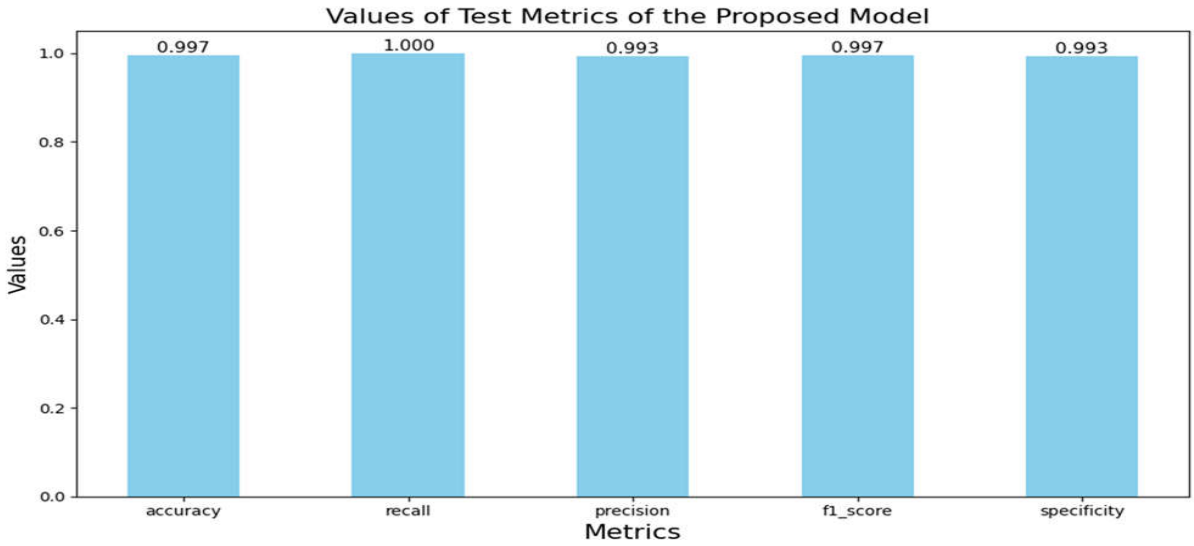

Fig. 15. The bar graph shows the evaluation performance of the proposed model MTBTDet_CNN for the detection of MRI brain tumors based on the metrics accuracy, recall, precision, f1_score, and specificity.

As a result, the graph formats an excellent interpretation of the suggested custom CNN (MTBTDet_CNN) model, which was optimized by several tests. All interpretation metrics (accuracy, recall, precision, F1 score, and specificity) showed excellent results for the model when the optimizer, LR, BS, AF, number of convolutional layers, PL, and epochs were set to their ideal settings. The graph confirms that the model does remarkably well in classifying the data, with near-perfect scores across all metrics.

4.8. Comparison Analysis of Proposed Model with State-of-art Method

The comparison of the suggested model with other innovative methods for optimizing or tuning hyperparameters in CNN models and their corresponding interpretation metrics such as accuracy, recall, precision, and F1 score as depicted in Table 6. Different techniques, including Genetic Algorithm, Bayesian Optimization, Grid Search Optimization, and Manual Tuning, were used to optimize hyperparameters like the number of convolutional layers, filters, AFs, LRs, PLs, and dense layers. When comparing these methods to the proposed model, it becomes evident that our model, which was manually tuned through various experiments, performs exceptionally well. The proposed model attains an accuracy of 99.7%, recall of 1.000, precision of 0.993, and F1 score of 0.997. This places it at the upper end of the spectrum, surpassing or matching most of the methods in terms of overall performance. For instance, methods using Bayesian Optimization by [40] reached an accuracy of 97.37% and 98.7% in different cases, while models optimized using Grid Search Optimization achieved accuracy as high as 99.3%. However, when it comes to recall and F1_score, with a 100% recall score, the suggested model performs better than these techniques and has a nearly perfect F1 score, indicating a highly balanced and effective model for classification tasks.

Additionally, many methods using Manual Tuning showed high performance, with accuracies ranging from 96% to 99.7%, but often lacked the perfect recall and precision that our model has. These comparisons indicate that the proposed CNN model, despite relying on manual tuning of hyperparameters, competes favorably and outperforms several models that use advanced optimization techniques like Genetic Algorithm and Bayesian Optimization in terms of critical evaluation metrics like recall and F1 score.

Table 6. Comparison analysis of proposed model MTBTDet_CNN with the state-of-art methods/custom CNNs-based hyperparameters to be tunned, hyperparameters optimization techniques, and performance evaluation metrics.

References and Year	Hyperparameters to be Optimized or Tunned	Optimization Techniques	Accuracy (%)	Recall (%)	Precision (%)	F1_ Score (%)	Specificity (%)
[37] 2019	Optimizers, No. of Conv + max-PL, AF, No. of FC + dropout layers, Dropout rate, No. of filters, Kernel sizes, No. of FC neurons, LR.	Genetic Algorithm	94.2	-	-	-	-
[39] 2021	Conv2D Kernel Size, Dropout Percentages, Conv2D Filters, Dense Filter, Max Pooling Size.	Bayesian Optimization	97.37	97.38	97.4	97.3	98.02
[38]2021	AF, No.of Conv and max pooling layers, Momentum, No. of FC layers, No. of filters, LR, Filter size, BS, and L2 Regularization	Grid Search Optimization	99.3	99.4	99.25	-	99.4
[40] 2022	Gradient descent optimizer, BS, AF, Dropout rate, Number of dense nodes.	Bayesian Optimization	98.70	98.66	98.33	98.66	-
[2] 2019	No. of Conv +ReLU, Dropout, No. of Norm. layers, dropout layers, Max. epochs, No. of FC layers, No.of Conv Kernels, Initial Learning Rate, Kernel sizes, PL, Optimizers, BS, LR drop factor	Manual Tuning	98.7	98.3	98.8	-	99.3
[41] 2020	Epochs	Manual Tuning	96	96	100	-	-
[42] 2020	-	Manual Tuning	100	100	100	100	100
[43] 2020	-	Manual Tuning	97.28	97.82	97.15	97.47	-
[44] 2020	PL, AF, Optimizer, Initializer	Manual Tuning	96.49	-	-	-	-
[45] 2020	-	Manual Tuning	99	99	99	99	99
[46] 2020	-	Manual Tuning	96.8	96	96	96.4	-
[47] 2021	-	Manual Tuning	98	100	97	98	95
[48] 2021	AF	Manual Tuning	98.6	98.6	99.6	99	-
[49] 2021	No. of Conv +ReLU, Dropout layers, epochs, No. of FC layers, Optimizers, BS, Dropout Rate	Manual Tuning	97	97.03	97	97	96.97
[50] 2021	Optimizer, LR, and epochs	Manual Tuning	94.74	94.39	94.03	94.19	97.35
[51] 2021	-	Manual Tuning	99.25	95.89	97.22	95.23	93.75
[52] 2021	Dropout rate, Dense layer, Optimizer	Manual Tuning	96	96	96	96	-
[53] 2022	-	Manual	96.5	-	-	-	-

		Tuning					
[54] 2022	Conv Layer+ReLU, Batch Normalization Layer, Dropout Layer, Cross channel normalization layer, FC Layer, Grouped Conv Layers, PL, Dropout Rate, LR, Optimizers, Epochs, BS, No. of Conv kernels, Conv kernel size, Image Size	Manual Tuning	97.2	96	97	-	-
[64] 2022	-	Manual Tuning	99	-	-	-	-
[56] 2023	-	Manual Tuning	97.33	97.50	97.50	97.50	-
[57] 2023	No. of Conv Layer, BS, Training - Testing ratio, LR, epochs	Manual Tuning	97.86	-	-	-	-
[58] 2023	-	Manual Tuning	100	100	100	-	-
[59] 2023	-	Manual Tuning	93.30	91.13	-	-	-
[60] 2023	-	Manual Tuning	95.44	-	-	-	-
[61] 2023	-	Manual Tuning	98.04	98	98%	98	-
[62] 2023	-	Manual Tuning	97.84	97.85	97.85	97.90	-
[63] 2023	-	Manual Tuning	98.86	98.83	98.72	98.77	99.41
[64] 2023	-	Manual Tuning	99.83	99.66	100	99.83	100
[67] 2024	learning rate, batch size, number of epochs, optimizer, shuffle, verbose, dropout rate, filters, filter size, and activation function.	Manual Tuning	97.18 93 96	97 91 96	97 95 96	97 93 96	-
[68] 2024	Activation Function and Optimizers	Manual Tuning	92.59	-	-	-	-
Proposed (MTBTDet_CNN)	Optimizer, LR, BS, AF, PL, No. of Convolutional layers, Epochs	**Manual Tuning**	**99.7**	**100**	**99.3**	**99.7**	**99.3**

In conclusion, the proposed model's ideal hyperparameters such as the LR of 0.0003, Adam optimizer, BS of 32, ReLU activation, three convolutional layers, max pooling, and 20 epochs, result in a highly effective model. The model's performance measures demonstrate equality with or superiority over current state-of-the-art techniques, indicating that it is robust in attaining accurate and precise detection results of the brain tumor.

5. Conclusion and Future Scope

This research presents a comprehensive manual hyperparameter optimization strategy for MRI-based brain tumor detection using a customized CNN model, MTBTDet_CNN. By systematically evaluating seven key hyperparameters across six controlled experiments, the study identified an optimal configuration: 4 convolutional layers including the input layer, ReLU activation, max pooling, Adam optimizer with a LR of 0.0003, batch size of 32, and 20 epochs. Using this setup, the proposed model achieved outstanding performance metrics: accuracy of 0.997, precision of 0.993, recall of 1.000, F1-score of 0.997, and specificity of 0.993, demonstrating its robustness in accurately detecting and classifying tumorous and non-tumorous brain MRI scans. These results validate the effectiveness of systematic manual hyperparameter tuning over arbitrary or black-box optimization methods.

Future work can explore the integration of more advanced architectures such as ResNet or DenseNet to further enhance feature extraction. Incorporating data augmentation and multi-modal MRI datasets could improve generalization to diverse clinical scenarios. Additionally, real-time deployment and interpretability techniques, such as Grad-CAM, can be investigated to provide clinicians with transparent and actionable insights for decision-making. Expanding this approach to multi-class tumor classification and longitudinal studies could further establish the clinical applicability of MTBTDet_CNN.

References

1. Folman J (1971) Tumor angiogenesis: therapeutic implications. N Engl J Med, 285(21): 1182–1186. 10.1056/NEJM197111182852108
2. Sultan HH, Salem NM, Al-Atabany W (2019) Multi-classification of brain tumor images using deep neural network. IEEE Access 7: 69215–69225. 10.1109/ACCESS.2019.2919122
3. Nie D, Zhang H, Adeli E, Liu L, Shen D (2016, October) 3D deep learning for multi-modal imaging-guided survival time prediction of brain tumor patients. In: Proc Springer, International conference on medical image computing and computer-assisted intervention, MICCAI Athens, Greece, pp 212–220. https://doi.org/10.1007/978-3-319-46723-8_25
4. Peri C, Michael M, Smith W (10th July) Types of brain cancer. Available: https://www.webmd.com/cancer/brain-cancer/brain-tumor-types. Accessed 11 Jun 2024.
5. Zhang Y, Li A, Peng C, Wang M (2016) Improve glioblastoma multiforme prognosis prediction by using feature selection and multiple kernel learning. IEEE/ACM transactions on computational biology and bioinformatics, 13(5): 825–35. 10.1109/TCBB.2016.2551745
6. Litjens G, Kooi T, Bejnordi BE, Setio AA, Ciompi F, Ghafoorian M, et al (2017) A survey on deep learning in medical image analysis. Medical image analysis, 42: 60–88. https://doi.org/10.1016/j.media.2017.07.005
7. Singh L, Chetty G, Sharma D (2012) A novel machine learning approach for detecting the brain abnormalities from mri structural images. In: IAPR international conference on pattern recognition in bioinformatics, Tokyo, Japan. Springer Berlin, pp 94–105. https://doi.org/10.1007/978-3-642-34123-6_9
8. Ghassemi N, Shoeibi A, Rouhani M (2020) Deep neural network with generative adversarial networks pretraining for brain tumor classification based on MR images. Biomed Signal Process Control, 57: 101678. https://doi.org/10.1016/j.bspc.2019.101678
9. Hashemzehi R, Mahdavi SJS, Kheirabadi M, Kamel SR (2020) Detection of brain tumors from MRI images base on deep learning using hybrid model CNN and NADE. Biocybern Biomed Eng, 40(3): 1225–1232. https://doi.org/10.1016/j.bbe.2020.06.001
10. Shen D, Wu G, Suk HI (2017) Deep learning in medical image analysis. Annu Rev Biomed Eng, 19: 221–248. https://doi.org/10.1146/annurev-bioeng-071516-044442
11. Khalid S, Khalil T, Nasreen S (2014) A survey of feature selection and feature extraction techniques in machine learning. In: 2014 science and information conference. IEEE, pp 372–378. 10.1109/SAI.2014.6918213
12. Passos D, Mishra P (2022) A tutorial on automatic hyperparameter tuning of deep spectral modelling for regression and classification tasks. Chemometrics and Intelligent Laboratory Systems, 223: 104520. https://doi.org/10.1016/j.chemolab.2022.104520
13. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC (2015) Imagenet large scale visual recognition challenge. International journal of computer vision, 115: 211-52. https://doi.org/10.1007/s11263-015-0816-y
14. Fu L, Zhao Y, Sun X, Huang J, Wang D, Ding Y (2021) Video object segmentation based on motion-aware ROI prediction and adaptive reference updating. Expert Systems with Applications, 167: 114153. https://doi.org/10.1016/j.eswa.2020.114153
15. Peng D, Xiong S, Peng W, Lu J (2021) LCP-Net: A local context-perception deep neural network for medical image segmentation. Expert Systems with Applications, 168: 114234. https://doi.org/10.1016/j.eswa.2020.114234
16. Khan ZY, Niu Z (2021) CNN with depthwise separable convolutions and combined kernels for rating prediction. Expert Systems with Applications, 170: 114528. https://doi.org/10.1016/j.eswa.2020.114528
17. Prabhu (2019) Understanding Hyperparameters and its Optimisation techniques. Available: https://towardsdatascience:com/ understanding-hyperparameters-and-its-optimisation-techniques- f0debba07568, [Accessed: 24-July-2024].
18. Lee WY, Park SM, Sim KB (2018) Optimal hyperparameter tuning of convolutional neural networks based on the parameter-setting-free harmony search algorithm. Optik, 172: 359-67. https://doi.org/10.1016/j.ijleo.2018.07.044
19. Kar P (2019) CS231n Convolutional Neural Networks for Visual Recognition." Available: http://cs231n:github:io/convolutional-networks/ [Accessed: 27 - Jun - 2024].
20. Shaver MM, Kohanteb PA, Chiou C, Bardis MD, Chantaduly C, Bota D, Filippi CG, Weinberg B, Grinband J, Chow DS, Chang PD (2019) Optimizing neuro-oncology imaging: a review of deep learning approaches for glioma imaging. Cancers. 11(6): 829. https://doi.org/10.3390/cancers11060829
21. Galanis NI, Vafiadis P, Mirzaev KG, Papakostas GA (2022) Convolutional neural networks: A roundup and benchmark of their pooling layer variants. Algorithms, 15(11): 391. https://doi.org/10.3390/a15110391
22. Murray N, Perronnin F (2014) Generalized max pooling. InProceedings of the IEEE conference on computer vision and pattern recognition, pp 2473-2480. https://doi.org/10.1109/CVPR.2014.317
23. Zubair S, Yan F, Wang W (2013) Dictionary learning based sparse coefficients for audio classification with max and average pooling. Digital Signal Processing, 23(3): 960-70. https://doi.org/10.1016/j.dsp.2013.01.004
24. Lee CY, Gallagher PW, Tu Z (2016) Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. InArtificial intelligence and statistics, PMLR, pp 464-472. https://doi.org/10.48550/arXiv.1509.08985
25. Dewa CK (2018) Suitable CNN weight initialization and activation function for Javanese vowels classification. Procedia computer science 144:124-32. https://doi.org/10.1016/j.procs.2018.10.512
26. Mondal A, Shrivastava VK (2022) A novel Parametric Flatten-p Mish activation function based deep CNN model for brain tumor classification. Computers in Biology and Medicine, 150: 106183. https://doi.org/10.1016/j.compbiomed.2022.106183
27. Jiang Y, Xie J, Zhang D (2022) An adaptive offset activation function for CNN image classification tasks. Electronics, 11(22): 3799. https://doi.org/10.3390/electronics11223799
28. Sharma S, Sharma S, Athaiya A (2017) Activation functions in neural networks. Towards Data Sci, 6(12): 310-316. 10.33564/ijeast.2020.v04i12.054
29. Nwankpa C, Ijomah W, Gachagan A, Marshall S (2018) Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv, 1811: 03378. https://doi.org/10.48550/arXiv.1811.03378
30. Agostinelli F, Hoffman M, Sadowski P, Baldi P (2014) Learning activation functions to improve deep neural networks. arXiv preprint arXiv, 1412: 6830. https://doi.org/10.48550/arXiv.1412.6830
31. Xie N, Li X, Li K, Yang Y, Shen HT (2019) Statistical karyotype analysis using CNN and geometric optimization. IEEE Access, 7: 179445–179453. 10.1109/ACCESS.2019.2951723

32. Ndong PSB, Adoni WYH, Nahhal T, Kimpolo C, Krichen M, Byed AE, Assayad I, Mutombo FK (2021). A face-mask detection system based on deep learning convolutional neural networks. In: Advances on Smart and Soft Computing: Proceedings of ICAC 2021 Springer: Berlin/Heidelberg, Germany pp 273–283. https://doi.org/10.1007/978-981-16-5559-3_23
33. Hinton G, Srivastava N, Swersky K (2012) Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on 14(8): 2.
34. Dozat, Timothy. "Incorporating nesterov momentum into adam." (2016).
35. Larose DT, Larose CD (2015) Data Mining and Predictive Analytics. Wiley Series on Methods and Applications in Data Mining, 25(9): 1682-1690.
36. Sengupta J (2023) How to decide the hyperparameters in CNN. Available: https://medium.com/@sengupta.joy4u/how-to-decide-the-hyperparameters-in-cnn-bfa37b608046. Accessed: 26-Aug-2024.
37. Anaraki AK, Ayati M, Kazemi F (2019) Magnetic resonance imaging-based brain tumor grades classification and grading via convolutional neural networks and genetic algorithms. biocybernetics and biomedical engineering, 39(1): 63-74. https://doi.org/10.1016/j.bbe.2018.10.004
38. Irmak E (2021) Multi-classification of brain tumor MRI images using deep convolutional neural network with fully optimized framework. Iranian Journal of Science and Technology, Transactions of Electrical Engineering, 45(3): 1015-1036. https://doi.org/10.1007/s40998-021-00426-9
39. Alshayeji M, Al-Buloushi J, Ashkanani A, Abed SE (2021) Enhanced brain tumor classification using an optimized multi-layered convolutional neural network architecture. Multimedia Tools and Applications, 80(19): 28897-917. https://doi.org/10.1007/s11042-021-10927-8
40. Ait Amou M, Xia K, Kamhi S, Mouhafid M (2022) A Novel MRI Diagnosis Method for Brain Tumor Classification Based on CNN and Bayesian Optimization. In Healthcare, MDPI, 10(3): 494. https://doi.org/10.3390/healthcare10030494
41. Mohammed BA, Al-Ani MS (2020) An efficient approach to diagnose brain tumors through deep CNN. Math. Biosci. Eng, 18: 851-67. 10.3934/mbe.2021045
42. Khan HA, Jue W, Mushtaq M, Mushtaq MU (2020) Brain tumor classification in MRI image using convolutional neural network. Math. Biosci. Eng, 17(5): 6203-16. 10.3934/mbe.2020328
43. Badza MM, Barjaktarovic MC (2020) Classification of brain tumors from MRI images using a convolutional neural network. Applied Sciences, 10(6): 1999. https://doi.org/10.3390/app10061999
44. Mzoughi H, Njeh I, Wali A, Slima MB, BenHamida A, Mhiri C, Mahfoudhe KB (2020) Deep multi-scale 3D convolutional neural network (CNN) for MRI gliomas brain tumor classification. Journal of Digital Imaging, 33(4): 903-15. https://doi.org/10.1007/s10278-020-00347-9
45. Ismael SA, Mohammed A, Hefny H (2020) An enhanced deep learning approach for brain cancer MRI images classification using residual networks. Artificial intelligence in medicine, 102: 101779. https://doi.org/10.1016/j.artmed.2019.101779
46. Roy SS, Rodrigues N, Taguchi Y (2020) Incremental dilations using CNN for brain tumor classification. Applied Sciences, 14: 4915. https://doi.org/10.3390/app10144915
47. Rai HM, Chatterjee K (2021) 2D MRI image analysis and brain tumor detection using deep learning CNN model LeU-Net. Multimedia Tools and Applications, 80: 36111-41. https://doi.org/10.1007/s11042-021-11504-9
48. Alhassan AM, Zainon WM (2021) Brain tumor classification in magnetic resonance image using hard swish-based RELU activation function-convolutional neural network. Neural Computing and Applications, 33(15): 9075-87. https://doi.org/10.1007/s00521-020-05671-3
49. Hapsari PA, Dewinda JR, Cucun VA, Nurul ZF, Joan S, Anggraini DS, Peter MA, I Ketut EP, Mauridhi HP (2021) Brain tumor classification in MRI images using en-CNN. International Journal of Intelligent Engineering and Systems, 14(4): 437-51. https://inass.org/wp-content/uploads/2021/07/20210...
50. Ayadi W, Elhamzi W, Charfi I, Atri M (2021) Deep CNN for brain tumor classification. Neural Processing Letters, 53(1): 671-700. https://doi.org/10.1007/s11063-020-10398-2
51. Abd El Kader I, Xu G, Shuai Z, Saminu S, Javaid I, Salim Ahmad I (2021) Differential deep convolutional neural network model for brain tumor classification. Brain Sciences, 11(3): 352. https://doi.org/10.3390/brainsci11030352
52. Minarno AE, Mandiri MH, Munarko Y, Hariyady H (2021) Convolutional neural network with hyperparameter tuning for brain tumor classification. Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control, 6(2): 127-132. https://doi.org/10.22219/kinetik.v6i2.1219
53. Shaik NS, Cherukuri TK (2022) Multi-level attention network: application to brain tumor classification. Signal, Image and Video Processing, 16(3): 817-824. https://doi.org/10.1007/s11760-021-02022-0
54. Kibriya H, Masood M, Nawaz M, Nazir T (2022) Multiclass classification of brain tumors using a novel CNN architecture. Multimedia Tools and Applications, 81(21): 29847-29863. https://doi.org/10.1007/s11042-022-12977-y
55. Tiwari P, Pant B, Elarabawy MM, Abd-Elnaby M, Mohd N, Dhiman G, Sharma S (2022) CNN based multiclass brain tumor detection using medical imaging. Computational Intelligence and Neuroscience, 2022(1): 1830010. https://doi.org/10.1155/2022/1830010
56. Rahman T, Islam MS (2023) MRI brain tumor detection and classification using parallel deep convolutional neural networks. Measurement: Sensors, 26: 100694. https://doi.org/10.1016/j.measen.2023.100694
57. Solanki S, Singh UP, Chouhan SS, Jain S (2023) Brain tumour detection and classification by using deep learning classifier. International Journal of Intelligent Systems and Applications in Engineering, 11(2s): 279-92. https://ijisae.org/index.php/IJISAE/article/view/2624
58. Gupta M, Sharma SK, Sampada GC (2023) Classification of Brain Tumor Images Using CNN. Computational Intelligence and Neuroscience, 2023(1): 2002855-2002861. https://doi.org/10.1155/2023/2002855
59. Mahmud MI, Mamun M, Abdelgawad A (2023) A deep analysis of brain tumor detection from mr images using deep learning networks. Algorithms, 16(4): 176. https://doi.org/10.3390/a16040176
60. Mahjoubi MA, Hamida S, Gannour OE, Cherradi B, Abbassi AE, Raihani A (2023) Improved multiclass brain tumor detection using convolutional neural networks and magnetic resonance imaging. International Journal of Advanced Computer Science and Applications, 14(3): 406-14. 10.14569/IJACSA.2023.0140346
61. Rasheed Z, Ma YK, Ullah I, Al Shloul T, Tufail AB, Ghadi YY, Khan MZ, Mohamed HG (2023) Automated Classification of Brain Tumors from Magnetic Resonance Imaging Using Deep Learning. Brain Sciences, 13(4): 602. https://doi.org/10.3390/brainsci13040602
62. Rasheed Z, Ma YK, Ullah I, Ghadi YY, Khan MZ, Khan MA, Abdusalomov A, Alqahtani F, Shehata AM (2023) Brain tumor classification from MRI using image enhancement and convolutional neural network techniques. Brain Sciences, 13(9): 1320. https://doi.org/10.3390/brainsci13091320
63. Abd El-Wahab BS, Nasr ME, Khamis S, Ashour AS (2023) Btc-fcnn: fast convolution neural network for multi-class brain

tumor classification. Health information science and systems, 11(1): 3. https://doi.org/10.6084/m9.figshare.1512427.v5.

64. Ullah N, Javed A, Alhazmi A, Hasnain SM, Tahir A, Ashraf R (2023) TumorDetNet: A unified deep learning model for brain tumor detection and classification. Plos one, 18(9): e0291200. https://doi.org/10.1371/journal.pone.0291200
65. Albalawi E, Thakur A, Dorai DR, Bhatia Khan S, et al. Enhancing brain tumor classification in MRI scans with a multi-layer customized convolutional neural network approach. Frontiers in computational neuroscience. 2024; 18:1418546. https://doi.org/10.3389/fncom.2024.1418546
66. Mohanty N, Sarmadi M. Brain tumor MRI classification and identification using an image classification model via Convolutional Neural Networks. medRxiv. 2024 Sep 25:2024-09. https://doi.org/10.1101/2024.09.13.23299832
67. Aamir M, Namoun A, Munir S, Aljohani N, Alanazi MH, Alsahafi Y, Alotibi F. Brain tumor detection and classification using an optimized convolutional neural network. Diagnostics. 2024; 14(16):1714.
68. Sudhakar B, Sikrant PA, Prasad ML, Latha SB, et al. Brain Tumor Image Prediction from MR Images Using CNN Based Deep Learning Networks. Journal of Information Technology Management. 2024; 16(1):44-60. 10.22059/jitm.2024.96374
69. Rasool N, Wani NA, Bhat JI, Saharan S, et al. CNN-TumorNet: leveraging explainability in deep learning for precise brain tumor diagnosis on MRI images. Frontiers in Oncology. 2025; 15:1554559. 10.3389/fonc.2025.1554559
70. Chandraprabha K, Ganesan L, Baskaran K. A novel approach for the detection of brain tumor and its classification via end-to-end vision transformer-CNN architecture. Frontiers in Oncology. 2025; 15:1508451. 10.3389/fonc.2025.1508451
71. Prakasha N (2023) Brain MRI Images. Kaggle 2023. Available online: https://www.kaggle.com/datasets/naveenprakasha/brain-mri-images
72. Devi K, Sharma AK (2023) Detection of Brain Tumor Using Novel Convolutional Neural Network with Magnetic Resonance Imaging. In2023 Seventh International Conference on Image Information Processing (ICIIP), IEEE pp. 57-62.