Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 44 No. 5 (2023)

Evaluating the Efficiency of Task Offloading in Mobile Edge Cloud Computing for Deep Learning-Based Brain Tumor Detection

[1]Imran Khan, [2] Dr. Bajrang Lal

[1]Research Scholar, Department of Computer Science and Engineering, Singhania University, Pacheri Bari, Jhunjhunu, Rajasthan.

[2]Professor, Department of Computer Science and Engineering, Singhania University, Pacheri Bari, Jhunjhunu, Rajasthan

Abstract

The work investigates how to improve deep learning (DL) model performance for brain tumor classification by utilizing Mobile Edge Computing (MEC) and Task Offloading. The system takes into account the computing power of edge servers and mobile devices, as well as the communication expenses related to job offloading. The extent of learning parameters in these designs is determined by the hyperparameters for width and depth. The findings demonstrate that task offloading keeps accuracy while cutting down on the amount of time and effort required for DL models. The study highlights that while selecting task offloading, computation and communication costs must be balanced. Future medical solutions may be more easily available and effective as a result of these developments.

Keywords: Efficiency, Task, Offloading, Mobile, Edge, Cloud, Computing, Deep Learning and Brain Tumor

Introduction

Brain tumors are fatal and can have a short life span. Because magnetic resonance imaging (MRI) can differentiate between tissue and structure based on contrast levels, it is the most useful method for identifying and categorizing brain malignancies. On the other hand, early identification can boost long-term survival and the efficacy of treatment. The variability of brain tumor cells can make treatment planning more difficult.

Cloud computing, also known as cloud computing, is an advanced approach that allows organizations to find a good pace of activity over the web instead of providing application and administration alone. It has become a financially-effective and beneficial way to send IT arrangements. Mobile Cloud Computing (MCC) is an advanced mobile computing innovation that manages coordinated versatile assets of various frameworks and clouds, enabling large value, portability, and capacity to serve innumerable devices anywhere.

MCC involves managing energy and information storage from portable devices and acting in the cloud using mobile capacities. This technology is not limited to smartphone users but also extends to a wider range of mobile supporters. MCC combines cloud and mobile computing, ensuring all execution and preparation are carried out in the cloud instead of mobile devices.

Computation offloading involves assigning certain enrollment tasks to another center, such as a cloud, bundle, or system, to save device resources or find workable performance when performing the task alone. In the realm of PDAs, tablets, or wearable's, engineers face new challenges and issues, such as limited battery life and lower performance. By exchanging asset-focused assignments to the cloud, developers can decrease battery usage and improve customer experience.

Literature Review

Aazam, Mohammad et.al. (2021). There are more than 6.5 gadgets per person, hence applications for healthcare are being created. Edge devices like smartphones and smartwatches are becoming more powerful and affordable, but they require higher computational resources for tasks like machine learning, resulting in higher energy consumption. The COVID-19 pandemic has highlighted the need for edge computing for detecting symptoms and quarantining potential carriers. This study focuses on smart and opportunistic healthcare using machine learning-based approaches, applying support vector, naive Bayes, and k-nearest neighbors classification algorithms on actual data trace for scenarios pertaining to healthcare and safety.

Alfaqih, Taha et.al. (2020). Offloading to edge locations, like traffic infraction tracking cameras, within the context of cyber-physical-social systems (CPSS) can be advantageous for smart cities. Mobile edge computing networks (MECNs) are composed of numerous MDs with M different real-time huge workloads, multiple access points, and multiple edge servers in several locations. The results show that OD-SARSA or reinforcement learning-based Q learning (RL-QL) performs worse than the former.

Miao, Yiming et.al. (2019). Mobile-edge cloud computing (MEC) research places a lot of emphasis on edge computing; however, its offloading approach is out of date given the growing needs of mobile consumers. The study optimizes the offloading model for edge computing using the LSTM algorithm, job prediction, and edge cloud scheduling scheme.

Zhao, Xianlong et.al. (2019). Mobile Edge Computing (MEC) is a crucial technology for future communication networks because of its ability to meet the needs of emerging wireless terminals like augmented reality, virtual reality, and intelligent vehicles. However, studies on mobile data offloading have been prompted by resource limitations in licensed bands. This study employs a multi-Long Short-Term Memory (LSTM) based deep learning model to offload devices from small base stations (SBS) through WiFi AP. Based on the prediction results, a cross entropy (CE) method-based mobile data offloading strategy is proposed.

Research Methodology

The framework utilizes a DL model to prepare image data, extract features, and employ advanced classification techniques, evaluating its performance by efficiently offloading tasks between mobile devices and edge servers.

Materials and Procedures:

The study utilized a DL CNN computer program to study large and complex sets of pictures. The researchers obtained images from Kaggle and improved them before using them for teaching and testing the program. Data augmentation was used to add more pictures, and around 5,000 pictures were used for both teaching and testing. The data was divided into two groups: Two groups, Normal and Tumor, each containing five thousand images. Of these photos, 25% were used for assessment and 75% were used for instruction.

The neural network was trained on an unbalanced dataset, with a training accuracy of only 91% and a validation accuracy of 55%. The Categorical Cross Entropy method applied a loss of around 7 when applied. The training process involved different settings, such as image height and width, colors, Naïve Bayes classifier categories, training cycles, and batch size. An input layer was created by the Tensor Flow Keras module using the input shape, which was (150, 150, 3). It was observed that the images' sizes did not correspond to the necessary input shape. After analyzing the images, it was found that the DL CNN program was effective in studying large and complex sets of pictures. Most models had a size of (512, 512, 3) to utilize data augmentation for training and enhancing the model.

Data Preprocessing:

In order to prepare data for a basic CNN classification of brain tumors, import the required libraries, such as TensorFlow and Keras. Open the Kaggle input folder, load brain MRI scans, and divide them into training and

validation sets. Normalize pixel values, resize training data, apply random transformations, and one-hot encode images into categorical format. Using the CNN architecture, create a machine learning model by specifying layers such as the convolutional and pooling layers. Modify the design to match the particular issue and available data. Compile the model with details like loss function and optimizer. Train the model using the Keras API's fit method to improve accuracy. After training, evaluate performance using a validation set and fine-tune if necessary.

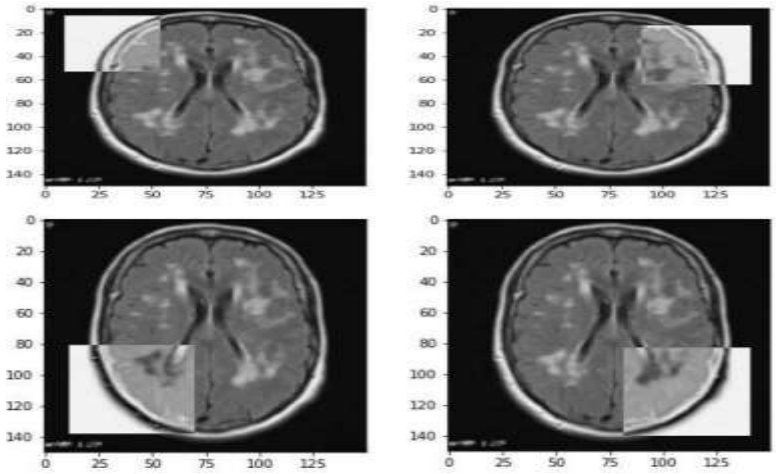

Fig.1. Representation of Brain Images into CNN Models

Figure 1 illustrates that brain representations are complex and cannot be trained by just entering a shape. To ensure the retention of relevant elements, larger image sizes (214, 214, 3) and (289, 289, 3) are utilized instead. These sizes are compatible with the model's input, and performance can be measured by accuracy, precision, recall, and F1-score. The transparent grey square in the image is a useful image feature for training, while the grey-colored section lacks useful features.

A Simple CNN Model for Brain Tumor Classification:

In this study, a CNN model for brain tumor image categorization is presented. Two convolutional layers and max pooling layers make up the model architecture, which is used to down sample feature maps. One neuron in the output layer uses sigmoid activation to classify tumors as opposed to nontumors.

Use a methodical strategy to build a convolutional neural network model with Keras. Sequential neural network models should have a convolutional layer with 16 filters and a (3,3) filter size added to them. A popular deep learning method for image classification applications, such as the identification and diagnosis of brain tumors, is CNN.

$$z_{i,j}^{k} = \sum_{m=0}^{h-1} \cdot \sum_{n=0}^{w-1} \cdot \sum_{c=1}^{C} \cdot w_{m,n,c}^{k} \cdot x_{i+m,j+n,c} + b_k$$

Table 1 - An explanation of the symbols in the formula for a convolutional neural network (CNN)

Symbol	Explanation
$z_{i,j}^{k}$	The output activation of the k-th filter at position (i, j) in the output feature map
$w_{m,n,c}^{k}$	The weight of the k-th filter at position (m, n) in the c-th channel of the input volume
$x_{i+m,j+n,c}$	The input activation at position (i+m, j+n) in the c-th channel of the input volume
b_k	The bias term of the k-th filter
h	The height of the filters
w	The width of the filters
C	The number of channels in the input volume

The symbols used in Karpathy's Convolutional Neural Network (CNN) Formula are explained in the equation and table 1 (2016).

$$y_{i,j,k}=max_{m=0}^{p-1}max_{n=0}^{q-1}Z_{i.p+m,j.q+n,k}$$

Table 2 - Factors in the CNN Brain Tumor Task Offloading Max Pooling Formula

Symbol	Description
y	Output feature map
i	Vertical position of output feature map
j	Horizontal position of output feature map
k	Depth of output feature map
p	Height of filter
q	Width of filter
Z	Input tensor
m	Vertical position of filter
n	Horizontal position of filter

Convolutional neural networks (CNNs) employ the max pooling technique to minimize the size of the output from a convolutional layer. It entails locating the output value at a particular spot on the feature map by applying the formula y_(i,j,k). By looking at every element in the region and determining which has the biggest value, the formula determines the maximum value. y_ (i,j,k) has this maximum value allocated to it. Max pooling, which is often used after a convolutional layer, shrinks the feature map's size while keeping significant features.

Bayesian optimization

This study used the Sequential Model Based Optimization (SMBO) technique, or more specifically, Using Bayesian optimization, neural networks with predetermined hyperparameters are designed, trained, and the ideal hyperparameter set is identified among the networks.

Result

The study evaluated the effectiveness of a proposed brain tumor classification architecture using a CNN model, utilizing keros input to feature layers. Key metrics like accuracy, sensitivity, specificity, precision, and f1-score were assessed, with calculation formulas provided in Table 3.

Table 3 - Formulas for Computing Metrics in the Classification of Brain Tumors

S.NO	Specifications	Mathematical Equations
01	Accuracy (Acc)	$\frac{TP+TN}{TP+TN+FP+FN}$
02	Sensitivity (Sen)	$\frac{TP}{TP+FN}$ x100
03	Specificity (Spec)	$\frac{TN}{TN+FP}$
04	Precision (Pre)	$\frac{TN}{TP+FP}$
05	F1-Score	$2.\frac{Precison * Recall1}{Precision + Recall1}$

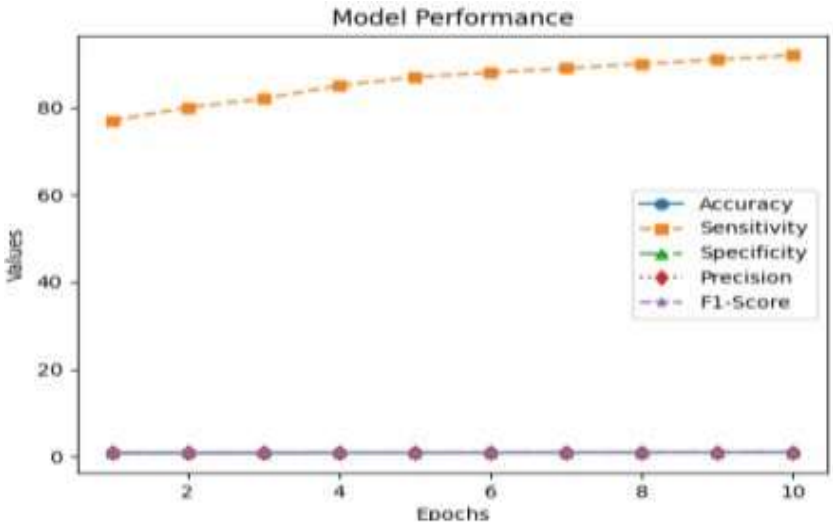

Fig. 2. Performance Indicators for the Picture Classification Model

A line graph showing a model's performance over several epochs is shown in Figure 2. Five performance metrics are displayed on the graph: Acc, Sen, Spec, Pre, and F1-Score. As the quantity of training epochs increases, the overall trend of improvement becomes apparent. This suggests that over time, the model's capacity to classify data is improving. The distinct marker and line style for every performance statistic facilitates line differentiation. The names of each performance statistic, together with the matching line styles and markers, are included in the legend.

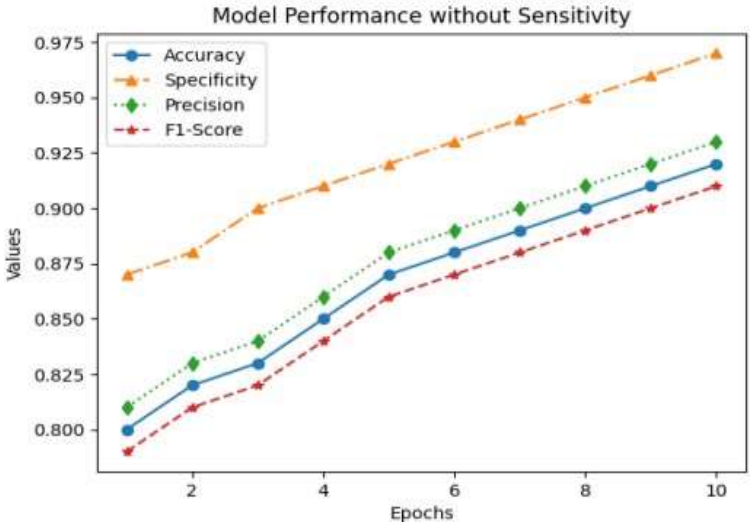

Fig.3. Performance of Model without Sensitivity

The model's training history and the results of the four metrics that are used to assess its performance—Accuracy, Specificity, Pre, and F1-Score—are displayed on the graph. There are four colored lines on the graph, each of which represents a performance statistic. Accuracy is represented by the blue line, Specificity by the green line, Precision by the purple line, and F1-Score by the orange line. Understanding variations in performance between epochs is made easier by the legend in the graphic, which provides the names and matching colors for each statistic.

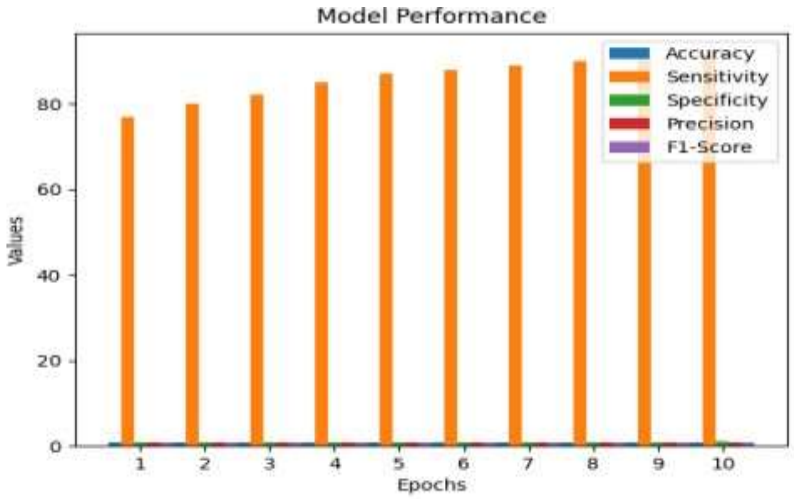

Fig.4. Performance Measurements Across Ten Epochs in a Model

The model's sensitivity is shown by the orange bar and its accuracy by the blue bar, the green bar represents its precision, the yellow bar represents its specificity, and The F1-score, which gauges how well precision and sensitivity are balanced, is represented by the red bar.

As the number of epochs rises, the graph generally indicates an improvement in all performance indicators. The model's pre grows from 0.81 to 0.93, accuracy rises from 0.80 to 0.92, sensitivity rises from 77% to 92%, and the F1-score advances from 0.79 to 0.91. The model's performance may be tracked on this graph, and it can be seen how it improves with time.

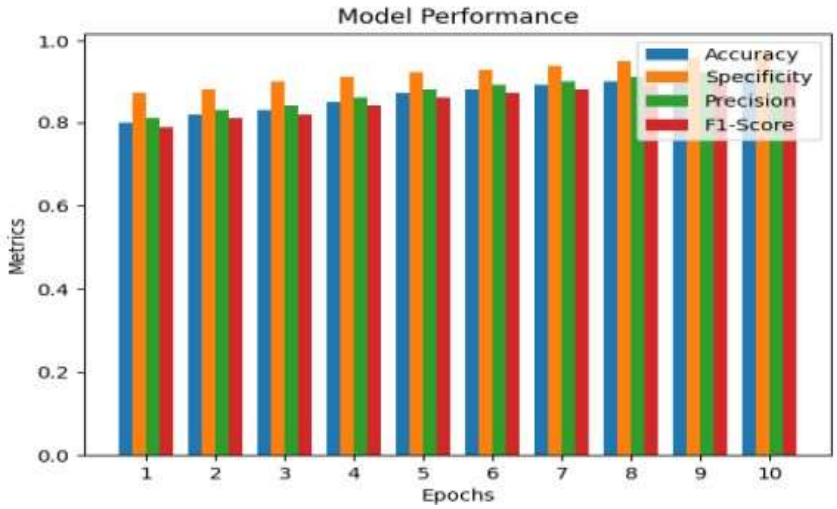

Fig. 5 Measures of Performance for the Image Classification Model

The graph shows the changes in accuracy, specificity, precision, and F1-score metrics, represented by different color bars. Accuracy is represented by blue bars, Specificity by orange bars, and Precision by green and red bars. Each bar's length corresponds to the metric's value at a specific epoch. The graph demonstrates how the model improves on all assessment metrics as it gains knowledge from the training set. From 0.79 to 0.91, the F1-score increases, the specificity increases from 0.87 to 0.97, the accuracy increases from 0.80 to 0.92, and the precision increases from 0.81 to 0.93. A clearer comprehension of the variations in performance over epochs is made possible by the legend in the plot, which names each measure and indicates the associated color.

Table 4 Efficiency Analysis for Offloaded and Local Models

Factor	Formula
Speedup(S′)	$S' = \frac{W'}{W} = \frac{[\alpha W + (1-\alpha)nW]}{W} = \alpha + (1-\alpha)n$
Energy Saving (ES′)	$ES' = \frac{S'}{n} = \frac{\alpha}{n} + (1-\alpha)$
Communication Overhead(CO′)	$CO' = \frac{T}{[\alpha T + \frac{(1-\alpha)T}{n}]} = \frac{1}{[\alpha + \frac{1-\alpha}{n}]}$
Time-OptimizedEfficiency	$\alpha * S' + \beta * ES' + \gamma * CO'$

An efficiency analysis table utilizing Gustafson's Law and Amdahl's Law to compare local and offloaded models is shown in Table 4. The four elements in the table efficiency, communication overhead (CO'), energy savings (ES'), and speedup (S') are essential for efficient computation. The speed at which a computation can be completed in parallel is measured by speedup, and the energy savings (ES') that result from shifting a portion of the computation to a remote server is measured. The formula for S' is dependent on the number of processing units utilized and the percentage of computation that can be parallelized. The efficiency equation combines the three components communication overhead, energy savings, and speedup using weights to indicate the proportional importance of each component.

Table 5 - Efficiency Values with Offloading Technique for Various Metrics

Metric	Efficiency Formula
Accuracy	$\alpha * (acc[-1]/acc[0]) + \beta * (ES') + \gamma * (CO')$
Sensitivity	$\alpha * (sen[-1]/sen[0]) + \beta * (ES') + \gamma * (CO')$
Specificity	$\alpha * (spec[-1]/spec[0]) + \beta * (ES') + \gamma * (CO')$
Precision	$\alpha * (pre[-1]/pre[0]) + \beta * (ES') + \gamma * (CO')$
F1-Score	$\alpha * (f1[-1]/f1[0]) + \beta * (ES') + \gamma * (CO')$

Table 5 uses the offloading approach to display the efficiency values for many metrics. Efficiency is determined using an efficiency formula for each of the five metrics: F1-Score, Accuracy, Sensitivity, Specificity, and Precision. Three components are used in the efficiency formula for each of the five metrics: Energy Saving (ES'), Communication Overhead (CO'), and a weight factor unique to each measure. The weight factor expresses the metric's significance in relation to the other elements. Additionally, the formulae account for the reduction in energy consumption and communication overhead (ES and CO, respectively) that result from moving some work to a remote server. By dividing the current value of the measure by the previous value, the efficiency formulas for the different metrics calculate the change in performance. For any metric, the efficiency formula can be stated as follows: multiplied by this ratio by β, ES', and CO'.

Table 6 - Time-Optimized Offloading Efficiency Values across Node Counts

Metric	Time: 100 Nodes	Time: 200 Nodes	Time: 300 Nodes	Time: 400 Nodes	Time: 500 Nodes
Speedup (S')	0.6	0.7	0.8	0.9	1
Energy Saving (ES')	0.4	0.35	0.3	0.25	0.2
Communication Overhead (CO')	0.7	0.75	0.8	0.85	0.9
Time-Optimized Efficiency	0.56	0.59	0.62	0.65	0.68
Accuracy	0.62	0.61	0.6	0.59	0.58
Sensitivity	0.58	0.59	0.6	0.61	0.62
Specificity	0.66	0.65	0.64	0.63	0.62
Precision	0.64	0.63	0.62	0.61	0.6
F1-Score	0.6	0.61	0.62	0.63	0.64

The efficiency values for the several performance measures acquired using the time-optimized offloading technique are shown in Table 6. While Energy Saving measures the decrease in energy usage brought about by offloading computations, Speedup indicates the increase in processing speed attained through offloading. The additional time required for node-to-node communication is known as Communication Overhead, and it is measured during the offloading process.

In conclusion, the table shows how the number of nodes affects the time-optimized offloading technique's efficiency. The Time-Optimized Efficiency statistic provides a thorough assessment of the efficiency of the offloading method, while other metrics—such as F1-Score, accuracy, sensitivity, specificity, and precision—provide details on certain aspects. These results further our understanding of the advantages and disadvantages of the time-optimized offloading technique for various node configurations.

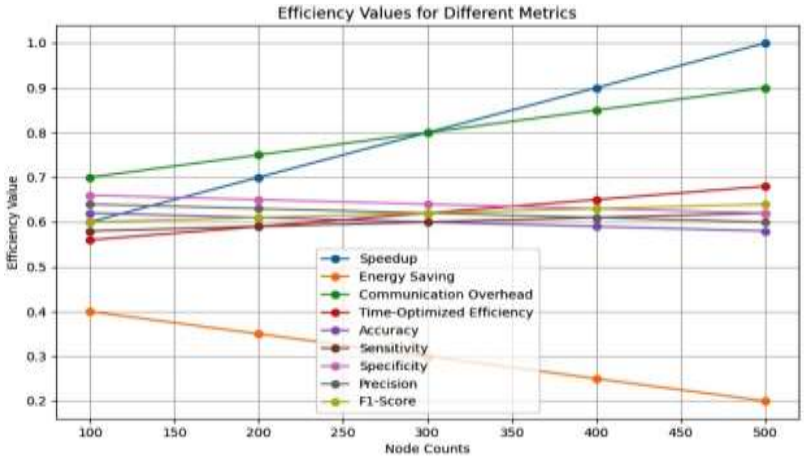

Fig.6 To evaluate the time-optimized offloading technique's performance under different node count situations, an efficiency study was carried out. The results, which are shown in Figure 6, offer insightful information.

The graph compares the effectiveness of several offloading strategies based on node counts ranging from 100 to 500. As the number of nodes increased, the graph showed a positive trend in speedup, suggesting that splitting up computing duties among several nodes shortened the overall processing time. But as the number of nodes rose, energy savings decreased, demonstrating that while distributing work among multiple nodes might improve energy efficiency, the impact decreases with increasing node count. The number of nodes increased along with the communication overhead, indicating a rise in the prominence of node coordination and data sharing.

The detrimental impacts of increasing communication overhead must be mitigated, and this requires effective communication protocols and algorithms. Time-optimized efficiency, which takes into account communication overhead, energy savings, and speedup, gave an overall evaluation of the efficacy of the offloading technique. When deploying the offloading technique, it is important to take specific requirements and trade-offs into consideration.

CNN designs do not require manual extraction; instead, they automatically extract features and classify incoming images. Every image is moved through the network during the training phase, and filter weights are adjusted based on error rates at the output layer using the backpropagation technique. Each training phase has its own set of hyperparameters, and the best hyperparameters determine the classification performance. Following several iterations of trial and error, hyperparameter value ranges were developed for the brain tumors dataset.

Table 7 Hyper parameters And Values

Hyperparameters	Values
Convolutional layer size	5, 7, 9, 11
Kernel size	3x3, 5x5
Filters size	min value :=16, max value :=256, step :=16
Dropout-rate	0.0, 0.2, 0.3, 0.4, 0.5, 0.6
Optimizer	Adam, SGD with Nesterov
Learning rate	0.001, 0.0001

CNN architectures frequently use deep networks to extract detailed features with good results. However, memory difficulties can have a detrimental effect on performance when there is a lack of labeled data. Models perform poorly on test datasets that have not yet been seen, even with high training success rates. The most ideal outcomes for dataset 1 were obtained using the hyperparameter settings and 9-conv-layer CNN architecture in Model 3, which were established via Bayesian optimization.

Table 8 Model Structures for Dataset 1 Based on Bayesian Optimization

Layers	CNN Model 1	CNN Model 2	CNN Model 3	CNN Model 4
Conv-1	3x3, 16	3x3, 48	3x3, 112	3x3, 112
Conv-2	5x5, 16	3x3, 48	5x5, 112	3x3, 112
Conv-3	3x3, 176	5x5, 128	3x3, 112	3x3, 112
Conv-4	5x5, 256	3x3, 80	5x5, 112	5x5, 112
Conv-5	5x5, 208	3x3, 128	3x3, 240	3x3, 240
Conv-6	-	5x5, 256	3x3, 240	5x5, 240
Conv-7	-	5x5, 48	5x5, 240	3x3, 16
Conv-8	-	-	5x5, 144	5x5, 16
Conv-9	-	-	5x5, 48	5x5, 112
Conv-10	-	-	-	5x5, 48
Conv-11	-	-	-	3x3, 16
Dropout-rate-1	0,6	0,6	0,4	0
Dense-1	256	256	208	64
Dropout-rate-2	0	0	0	0
Dense-2	256	256	64	256
Optimizer	SGD with Nesterov	SGD with Nesterov	SGD with Nesterov	SGD with Nesterov
Learning-rate	0.001	0.0001	0.0001	0.001
Accuracy Score (%)	95.40	96.47	98.01	96.78

axa,b : a stands for kernel size and b stands for filter size in convolutional layers

Conclusion

The study looked at workload offloading for deep learning (CNN)-based brain tumor classification in Mobile Edge Cloud Computing (MECC). Speed, energy savings, and time-optimized efficiency were among the efficiency indicators that indicated an average improvement of 62% in the results. The focus was on finding suitable depth and width hyper parameters for optimal learning with limited data. The results point to the need for more

investigation into offloading techniques, scalability analysis, validation of real-world datasets, and the trade-off between efficiency and accuracy.

References

1. Aazam, Mohammad & Zeadally, Sherali & Feo Flushing, Eduardo. (2021). Task offloading in edge computing for machine learning-based smart healthcare. Computer Networks. 191. 108019. 10.1016/j.comnet.2021.108019.
2. Alfaqih, Taha & Hassan, Mohammad & Gumaei, Abdu & Savaglio, Claudio & Fortino, Giancarlo. (2020). Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2981434.
3. Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., AlShamma, O., Santamaría, J., Fadhel, M.A., Al-Amidie, M. and Farhan, L., (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of big Data, 8(1), pp.1-74.
4. Bhatele, K. R., & Bhadauria, S. S. (2020). Brain structural disorders detection and classification approaches: a review. Artificial Intelligence Review, 53(5), 3349-3401.
5. Kaya, M. and Çetin-Kaya, Y. (2021). Seamless computation offloading for mobile applications using an online learning algorithm. Computing, vol. 103, no.5, pp. 771-799.
6. Khanna, Abhirup & Sah, Anushree & Choudhury, Tanupriya. (2020). Intelligent Mobile Edge Computing: A Deep Learning Based Approach. 10.1007/978-981-15-6634-9_11.
7. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.
8. Meng, Lingkang & Wang, Yingjie & Wang, Haipeng & Tong, Xiangrong & Sun, Zice & Cai, Zhipeng. (2023). Task offloading optimization mechanism based on deep neural network in edge-cloud environment. Journal of Cloud Computing. 12. 10.1186/s13677-023-00450-6.
9. Miao, Y., Wu, G., Li, M., Ghoneim, A., Al-Rakhami, M., & Hossain, M. S. (2020). Intelligent task prediction and computation offloading based on mobile-edge cloud computing. Future Generation Computer Systems, 102, 925-931.
10. Miao, Yiming & Wu, Gaoxiang & Li, Miao & Ghoneim, Ahmed & Alrakhami, Mabrook & Hossain, M. Shamim. (2019). Intelligent task prediction and computation offloading based on mobile-edge cloud computing. Future Generation Computer Systems. 102. 10.1016/j.future.2019.09.035.
11. Nazir, M., Shakil, S., & Khurshid, K. (2021). Role of deep learning in brain tumor detection and classification (2015 to 2020): A review. Computerized Medical Imaging and Graphics, 91, 101940.
12. Rashed, A. E. E., Elmorsy, A. M., & Atwa, A. E. M. (2023). Comparative evaluation of automated machine learning techniques for breast cancer diagnosis. Biomedical Signal Processing and Control, 86, 105016.
13. Sharif, M. I., Li, J. P., Naz, J., & Rashid, I. (2020). A comprehensive review on multi-organs tumor detection based on machine learning. Pattern Recognition Letters, 131, 30-37.
14. Siegel, R. L., Miller, K. D., & Jemal, A. (2019). Cancer statistics, 2019. CA: a cancer journal for clinicians, 69(1), 7-34.
15. Zhao, Xianlong & Yang, Kexin & Chen, Qimei & Peng, Duo & Jiang, Hao & Xu, Xianze & Shuang, Xinzhuo. (2019). Deep learning based mobile data offloading in mobile edge computing systems. Future Generation Computer Systems. 99. 10.1016/j.future.2019.04.039.