A Comparative Analysis of Convolutional Neural Network Architectures for Brain Tumor Classification

Sai Akshay Suresh
Faculty of Science, Engineering and Technology
University of Adelaide, North Terrace, Adelaide, South Australia
a1906525@adelaide.edu.au

Abstract

This paper investigates the classification of brain tumors through the utilization of a variety of convolutional neural network (CNN) architectures. On the preprocessed brain tumor sample dataset, early CNN architectures such as LeNet-5 and AlexNet, interim CNN architectures like ResNet-34 and ResNet-50, and the most recent architecture, EfficientNet, have been trained and tested. These models were subjected to hyperparameter variation of a variety of optimizers and epoch values. The architectures' performance in classifying brain tumors was examined in terms of accuracies, losses, sensitivity, specificities, precisions, and F1 scores. The efficacy of each model was evaluated by analyzing its outputs to promote a more efficient diagnosis of brain tumors by oncologists. The identification of brain tumors could be an alarming step in preventing further complications and treating the tumors when it is curable. This investigation illustrates the strengths and limitations of each architecture about its capacity to assess brain tumors.

1. Introduction

1.1 Background

The cellular units called "simple cells" and "complex cells" in animal visual cortexes are responsive to specific patterns in images, which were discovered by scientists Hubel and Wiesel in the 1950s. They suggested that these two types of cells were useful in pattern recognition. This brought an awareness of how brains visually process information [1]. This was further elevated by the development of a hierarchical, multilayered artificial neural network called "neocognitron" in the 1970s by Kunhiko Fukushima which possessed the ability to learn visual pattern recognition by unsupervised learning. Backpropagation was applied to train neural for the identification of handwritten zip codes by Yann LeCun in 1989. LeCun and his team came up with the original convolutional neural network (CNN) architecture "LeNet-5" in 1998 for document recognition [2]. The CNNs were further developed in the 2000s gradually but it was not becoming a viable option for image recognition as the computers had limited capabilities. There was a breakthrough when AlexNet in 2012 won the ImageNet Large Scale Visual Dataset (ILSVRC) which did large-scale and multiclass image classification more efficiently with a significantly low error rate which was unprecedented at that time [3]. The architectures were improved over time with the number of layers and pivotal parameters which were useful for effective deep learning tasks.

1.2 Convolutional Neural Network Architecture

A convolutional neural network simply, is a deep learning network architecture that can directly learn from the data [4]. As given in Figure 1, a conventional CNN architecture has the following parts –

i) Convolutional Layers
ii) Pooling Layers
iii) Fully Connected Layers

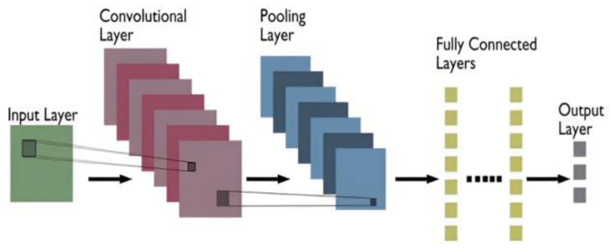

Figure 1. **A Common CNN Structure** with its layers.

1.2.1 Convolutional Layer

The input layer receives the images as raw data represented as a 3D unit (Width*Height* Channels). The convolutional layer extracts the features of the input dataset with the use of filters in multiple aspects like edges, repetitive color patterns, textures, shapes, noise, distortion, etc which will be separately represented as stacks. To reduce the number of parameters and improve efficiency, the same filter is shared across the entire image, which is called parameter sharing. A small section of the input units is connected to each neuron in the convolutional layer which helps to maintain spatial relationships [4,5].

1.2.2 Pooling Layers

Pooling layers replace the output of the network at a certain number of spots by extracting descriptive statistics of the nearest outputs. This helps in reducing the dimensional space size of the feature map representation. There are two types of pooling, namely max pooling and average pooling. Max pooling minimizes the dimensional size of features by choosing the highest value in each pooling window of a feature map. These values will be denoted as a max pooled feature map. Average pooling shortens the dimensional space by averaging all the values in a pooling window of a feature map. This will be represented as an average pooled feature map [5,6].

1.2.3 Fully Connected Layer

The outputs taken from previous layers as multiple feature maps stacked together will be transformed into a 1-D vector of values with all the features integrated that will be used for training. Final changes are made here as it performs an abstract reasoning needed for classification. The information is passed to the output layer, which predicts with an impulse of some activation function provided, eg., the softmax activation function is for multiclass prediction [6, 7].

1.2.4 Hyperparameters

Hyperparameters are the attributes of CNN, which can be altered for a machine's performance improvisation. These hyperparameters can be applied before or after convolution and fully connected layers based on requirements [8]. Examples of hyperparameters are activation functions, dropout rate, batch normalization configurations, learning rate, etc.

1.3 Applications of CNN

The CNN has spread its wings in a wide range of applications across various fields [9]. Some of them are -
i) Image Classification - Detection of types of images.
ii) Image Segmentation - Segmenting a particular region in an image for comprehensive analysis.
iii) Face Recognition - Face detection for security purposes etc.
iv) Video Analysis - Thorough checking of elements and properties in the video for identifying a theft, robbery, etc.
v) Natural Language Processing – Identifying words and letters on an image and converting it into text.
vi) Medical Image Analysis – Examining images like tumors to study and predict the seriousness of such hazardous elements on one's body by interpreting the resultant images of various scans like Computed Tomography (CT) and Magnetic Resonance Imaging (MRI).

2. Methodology

2.1. Overview

A brain tumor of 498 medical images and 9 classes of types MRI, CT, and MRI-CT fused were preprocessed by using the methods namely, rescaling, resizing, and augmentation. This preprocessed dataset was split as 75 % for training and 25 percent for validation and was experimented with CNN images classification models such as LeNet-5, AlexNet, ResNet-34, ResNet-50, and EfficientNet with the variation of hyperparameters namely, optimizers[Gradient Descent, Stochastic Gradient Descent, AdaGrad, AdaDelta, RMSProp and, Adam] and epoch values [5, 10, 20, 40, 60] with the usage of ReduceLROnPlateau for learning rate adjustments for each of the architecture. The models were validated based on their key performance indicators like training accuracy, testing accuracy, validation accuracy, training loss, testing loss, validation loss, sensitivity, specificity, precision, and F1- score as given in Figure 2. The models and hyperparameters were chosen carefully to study the evolution of CNN architectures over time.

2.2 Hardware

The experiments were conducted using the NVIDIA A100 GPU, which offered high-performance computing capabilities for deep learning applications. Key specifications of the A100 GPU used via Google Colaboratory are given as follows:

i) GPU Model: NVIDIA A100
ii) GPU Memory: 40 GB HBM2 (High Bandwidth Memory)
iii) Memory Bandwidth: 1.6 TB/s
iv) GPU Interconnect: NVLink Full Mesh
v) Interconnect Bandwidth: 600 GB/s

2.3 Models

2.3.1 LeNet-5 (1998)

The LeNet-5 is the oldest of all CNN architectures developed by Yann LeCun in 1998 for the sole purpose of handwritten digit recognition by using the Modified National Institute of Standards and Technology (MNIST) dataset. This model was primarily applied in postal services and banking fields [10].

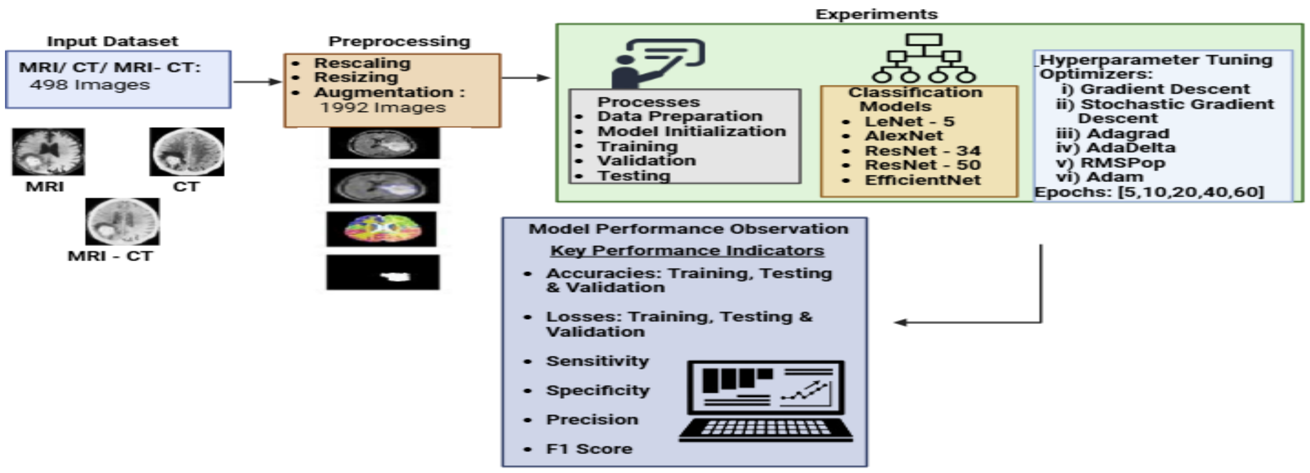

Figure 2, **The Workflow of the Experiments** with the steps performed.

LeNet-5 is a 7-layered architecture (3 convolutional, 2 subsampling, and 2 fully connected layers) that could take up an input size of 32x32 pixels of grayscale images. The convolutional layers and pooling layers were first introduced in LeNet-5 with the usage of "tan-h" as an activation function. As our brain tumor dataset had 9 classes, LeNet-5 was applied which could take up to 10 classes (digits 0 to 9). LeNet-5 has 60,000 training parameters [10].

2.3.2 AlexNet (2012)

The AlexNet was a pathbreaking architecture when it was introduced in 2012 by Alex Krizhevsky for classifying 1.2 million high-resolution images in the ImageNet Large Scale Visual Recognition Challenge (LSVRC) which contained 1000 different classes. AlexNet has 8 layers (5 convolutional and 3 fully connected) which will take three channeled RGB images with 227x227 pixel size. AlexNet has fetched up a top-5 error rate of 15.3 % with 17 % variance which was novel at that time. AlexNet was the first architecture to introduce stacked convolutional layers, the Rectified Linear Unit (ReLu) and to use a Graphical Processing Unit (GPU) to improve performance. AlexNet has around 62 million learnable parameters [11].

2.3.3 ResNet-34 (2015)

ResNet-34 was introduced in 2015 by Kaiming He and his team which had 34 layers (5 convolutional, 1 fully connected, 16 residual blocks with 2 convolutional layers each) that will allow three channeled RGB images of 224x224 pixel size. New additions called "Residual block" and "Skip connections" were introduced as the residual block was found to overcome the problem of vanishing gradient (i.e when the weight does not get updated by a significant value because of the width of neural network and activation function) and the skip connection provides additional paths for the flow of gradients which in turn preserve strong gradient signals throughout the network. Along with these, a subtle pivotal factor called batch normalization which standardizes batches of data after each convolution layer. These additions made accuracy to get improved from the existing models. ResNet-34 has around 63 million training parameters. ResNet-34 can classify up to 1000 classes [12].

2.3.4 ResNet-50 (2015)

ResNet-50 was also introduced in 2015 by the developers of ResNet-34. ResNet-50 contained 50 layers (5 convolutional, 1 fully connected, 16 bottleneck blocks with 3 convolutional layers in each block) which can absorb 3 channeled RBG images of 224x224 pixel size. The notable addition to this architecture was the bottleneck which reduces the number of parameters and matrix multiplications. Further, ResNet-50 uses 1*1 convolutions to reduce and restore the dimensional space of the architecture. This architecture can categorize images into around 1000 classes [13].

ResNet-50 is widely regarded as an ideal architecture for various computer vision applications. Additionally, the inclusion of bottlenecks makes this architecture have 23 to 25 million learning parameters which is almost half of the ResNet-34 which in turn leverages up the accuracy to a higher level with low computational cost [13].

2.3.5 EfficientNet (2019)

EfficientNet architecture was introduced in 2019 by Tom and Le. EfficientNet provides a family of models (B0 to B7), which have a range of layers from 82 to 152, with increasing complexity and performance. These models could intake images from 224x224 to 600x600 pixels corresponding to the family of models. EfficientNet is one of the top efficient models that requires the least FLOPS (floating point operations per second) which gauges the computation complexity involved in a model [14].

EfficientNet reaches state-of-the-art accuracy on common image classification learning tasks. EfficientNet incorporates the usage of mobile inverted bottleneck convolution which inverts the bottlenecks to increase efficiency and EffcientNet uses the “Swish” activation function to increase the performance. EfficientNet has around 5.3 million to 66 million parameters according to the family of models that can classify images around 1000 classes. EfficientNet of family B0 has been experimented with for this research [14].

2.4 Hyperparameter Tuning

2.4.1 Epochs

An epoch is a complete cycle of learning through the entire training dataset. In simple terms, the better the epochs are, the better the model detects the images in the right classes in most cases. However, there is a chance of obtaining diminishing returns when additional epochs may not significantly improve the accuracy. The number of epochs influences a model’s performance. In some cases, there can be underfitting with the use of a meager number of epochs, and there will be other few cases when there will be overfitting in a model due to the usage of a high number of epochs. Ideal epoch count depends on a specific dataset, and a specific model’s architecture and it requires experimentation to figure out the right epoch value that aligns well with the model [15]. In our case for the brain tumor dataset, the models were tested on the following epoch sizes: 5, 10, 20, 40, 60.

2.4.2 Optimizers

The optimizers are the functions that alter the attributes of deep learning architectures such as weights and biases to improve its performance [16]. The following optimizers, listed based on their efficiencies from low to high were varied across all the architectures: **i) Gradient Descent (GD)** - Finds the lowest value taken by a differentiable function. GD has slow convergence, mainly for complex problems and it is sensitive to learning rate change. **ii) Stochastic Gradient Descent (SGD) -** A variant of GD that learns from training data as small cohorts or separate data points rather than learning the whole data set at once. SGD is faster than GD due to frequent updates and, noise updates can lead to variation in losses. **iii) AdaGrad** – A variant of GD that gets modified to the learning rates of each parameter based on historical gradients and works well with sparse data. This optimizer allows to improve the efficiency of models. **iv) AdaDelta** – An SGD variant that can adapt to learning rates based on a sliding window of gradient updates, instead of gathering all the past gradients. This optimizer eliminates the need to set a learning rate initially which makes the architectures robust to different datasets. **v) RMSprop** – The root mean squared propagation technique, which updates the model’s parameters in the opposite direction of the gradient and computes the gradient of the loss function with parameters in turn reduces the losses. This optimizer adapts well while handling mobile label targets when the target function gets updated over time. **vi) Adam** – A variant of SGD that is based on adaptive estimation of the first-order and second-order moments. This is an optimizer based on iterations, which can reduce losses significantly during the training of datasets in neural networks. The Adam optimizer combines the benefits of both AdaGrad and RMSprop, thus making it effective for large datasets and high-dimensional parameter spaces.

2.4.3 ReduceLROnPlateau

The learning rate scheduler when an observed metric stops improving modifies the learning rates [17]. This modification is customisable, and in our case, it has been set to reduce the learning rate by half when the metric does not seem to improve for 5 consecutive epochs.

2.5 Key Performance Indicators

The key performance metrics were mostly calculated using coding. The accuracies and losses i.e cross-entropy loss (loss commonly used for multi-class classification) of all the types namely testing, training, and validation were obtained by “model.evaluate()” function of the Tensorflow/Keras library, and the precisions, sensitivity, and F1-score were obtained from confusion matrices by using the classification_report () function of the Scikit-Learn library [18, 19, 20]. The definitions and the" functions used for obtaining the metrics are given as follows:-

i) Training Accuracy – The number of correct predictions of the total number of predictions on the training dataset. **ii)Training Loss** – The number of wrong predictions made by the model obtained. **iii) Validation Accuracy** – A measure of the number of right predictions made on the data which was not seen during training. **iv) Validation Loss** – A measure of the number of wrong predictions made by the model on the data that was unseen during training. **v) Test Accuracy –** A ratio of the number of right predictions to the total number of predictions by the model on the test dataset. **vi) Test Loss –** A measure of the error when the model fails to predict the images properly on the test dataset. **vii) Precision** – The ratio of true positive predictions among all the positive predictions i.e. true positives and false positives. **viii) Sensitivity -** The number of true positive predictions among all the actual positive samples i.e. true Positives and false Negatives. **ix) F1 Score** – The harmonic mean of precision and recall. **x) Specificity** – The proportion of true negative predictions among all the negative samples i.e. true negatives and false positives. Specificity has been calculated manually with the formula :

$$\textbf{Specificity: } \frac{True\ Negatives}{True\ Negatives + False\ Positives}$$

3. Findings and Analysis

3.1 The Dataset

The brain tumor dataset taken contained 498 images in the following classes :

Source	**Size**	**Tumor Type – Scan Type**	**Number of Images**	**Class Name**
Medharvard	224 * 224 * 3	Sarcoma – MRI	96	MRI Tumor Sarcoma
Medharvard	224 * 224 * 3	Sarcoma – CT	24	CT Tumor Sarcoma
Medharvard	224 * 224 * 3	Meningioma – MRI	81	MRI Tumor Meningioma
Medharvard	224 * 224 * 3	Meningioma – CT	27	CT Tumor Meningioma
Kaggle	236 * 236 * 1	Meningioma – MRI	115	MRI Tumor Meningioma [Kaggle]
Kaggle	236 * 236 * 1	No Tumor – MRI	105	MRI No Tumor [Kaggle]
Medharvard	224 * 224 * 3	Anonymous Fused Tumors – MRI - CT	30	Graph Cut Fused Images Output
Scan Center	413 * 413 * 1	No Tumor – Dicom MRI	10	Dicom - MRI
Scan Center	413 * 413 * 1	No Tumor – Dicom CT	10	Dicom - CT
			Total: 498	

Table 1, **Summary of the Dataset.**

The brain tumor datasets were collected mainly from the websites Medharvard and Kaggle and the real biomedical images were collected from a scan center. There were 54 (6 images per class) images that were tested. The number of images with their classes and sizes is given in Table 1. An outlook of the dataset according to classes is given in Figure 3.

3.2 Preprocessing

The training image dataset is preprocessed to make sure that the data becomes more acceptable to the architectures so that there is no discrepancy concerning the dataset [21]. The images are first rescaled by dividing pixel values by 255, standardizing them to the [0,1] range. The data augmentation is performed on the training data with the usage of "ImageGenerator()" function extracted from "tensorflow.keras" library by rotating them up to 10 degrees and width, height shifts, shear transformations, and zoom were increased up to 10 % with horizontal shifts. The data has been split 75 % for training (377 images) and 25 % for validation (121 images). The images were sized to 32*32 pixels for LeNet-5 architecture and 128*128 for the remaining architectures. Data was prepared in batches of 32 images and the class mode has been set for multi-class classification. The RGBA channels found were converted to RGB. The mean values for red, green, blue, and grayscale were found. Color channel means are normalized

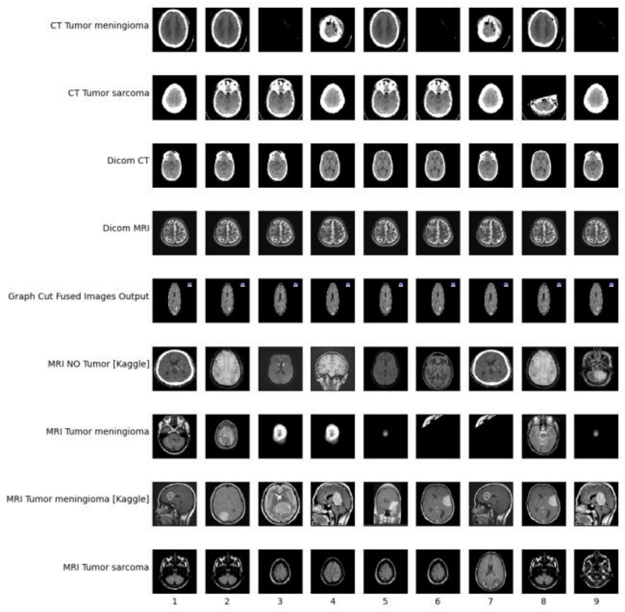

Figure 3, **A Preview of the images** in the dataset which has 9 classes.

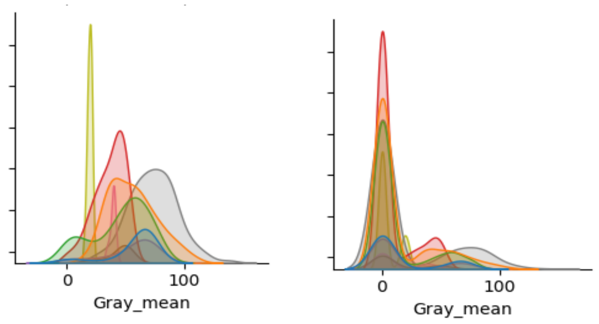

Figure 4, **Distribution Plot of Gray Mean** in dataset labels before (R) and after (L) Balancing Label.

by the grayscale mean [22]. The pair plots were projected to visualize the distribution of color features before and after augmentation as given in Figure 4. It can be observed that the grayscale intensity of the images is close to 0, as the images are dark.

3.3 Results and Interpretations

The "tensorflow.keras" library has been used for building layers of the models. The LeNet-5 model with the gradient descent optimizer and epoch alterations gave the train, validation, and test accuracies below 25% and the train, validation, and test losses were almost close to 2. LeNet-5 with stochastic gradient descent and AdaGrad optimizers gave slightly improved accuracies with minimal reduced losses compared to the GD optimizer. The sensitivities, precisions, and F1 scores were close to 0, where they must be close to 1 ideally [23]. AdaDelta gave accuracies in the range of 3 % to 12 % with losses being greater than 2 in all the cases while the sensitivity, precision and F1 score values were extremely low. There was a significant improvement in accuracies close to 67 percent and the losses were reduced to less than 1.5 with improvement in sensitivity, specificity, and precision as the epochs were increased when the RMSprop optimizer was used. Adam optimizer emerges as the winner for LeNet-5 architecture with accuracies being close to 70 and especially, the test accuracy being 72 % with a loss around 0.85 with the increment of epochs. The sensitivity, precision, and F1-score were significantly better being close to 0.75. It was noted that the specificity was close to 1 in all the optimizer types and epoch variations. LeNet-5 can be considered acceptable for good sensitivities in any given case.

The AlexNet architecture with GD and SGD optimizers gave almost identical results in terms of all the metrics and the specificity remained close to 1 like the LeNet-5 architecture with the improvement of epochs. AlexNet with AdaGrad fetched accuracies close to 71 percent which is significant and the losses being 0.8 approximately. The sensitivity, precision, and F1 score along with other metrics improved with the increase of epochs. AlexNet with AdaDelta gave results that were comparatively dissatisfying than the first two optimizers in almost every aspect. RMSprop and Adam gave surprising results in terms of accuracies which went close to 85 % and the training loss reached closer to 1. However, the validation and test losses were relatively higher than the training loss which suggests that there can be an overfitting possibly [24]. The sensitivity, precision, and F1 scores seemed to get closer to 1 as the number of epochs increased. In all the hyperparameter variation cases of AlexNet, the specificity was close to 0.9 which shows that AlexNet also can identify true negatives among all the negative samples.

The ResNet-34 with GD optimizer was seen to be performing better than the ResNet-50 with the GD optimizer in terms of all the metrics and in both types, the values were seen to be getting better with the increment of epochs. However, the accuracies were close to 50 percent overall and the losses were in the range of 1 to 2. It was observed that the test losses were higher than the training losses, which indicates that there may have been overfitting [24]. SGD optimizer for both the ResNet-34 and ResNet-50 was seen to slightly perform better than the GD but the values were almost the same for all the performance metrics. ResNet-34 with AdaGrad has shown a significant improvement with better outcomes in all aspects but SGD for ResNet-50 disappoints with a performance on par with the previous optimizers for the architecture. AdaDelta has given possibly the worst results for both the ResNet-34 and ResNet-50 architectures with a disenchantment in all the prospects, which shows that the models may have been incompetent to classify 498 images dataset as the models are meant for very high number of images. RMSprop and Adam have excelled for both architectures in all aspects giving pleasant results with accuracies close to 83 percent and losses lesser than 1 above 40 epochs, but it was observed that ResNet-50 with RMSprop has more test accuracy than the training accuracy which signifies that there may have been underfitting of the data [24]. It was seen that the same pattern of high specificity in any given case was repeated on ResNet architectures too.

The EfficientNet (B0 family) with the GD, SGD, and AdaGrad optimizers has shown results that were very low in all aspects with accuracies being lesser than 25 % in any case and the losses were between 1.8 and 3.2 which was high. The parameters like sensitivities, specificities, and F1

Arc	Opt	Epochs	Train Accuracy	Train Loss	Val Accuracy	Val Loss	Test Accuracy	Test Loss	Sensitivity	Specificity	Precision	F1 Score
LeNet-5	Adam	60	74%	0.686	64%	0.953	72%	0.871	0.718	0.964	0.801	0.734
AlexNet	Adam	60	97%	0.118	74%	0.972	83%	0.815	0.847	0.979	0.870	0.848
ResNet-34	Adam	60	93%	0.200	77%	0.881	83%	0.650	0.829	0.979	0.865	0.836
ResNet-50	Adam	60	88%	0.289	76%	0.778	76%	0.521	0.750	0.969	0.753	0.736
Efficient Net	RMSpro-p	10	23%	1.97	23%	1.95	11%	2.22	0.111	0.889	0.012	0.022

Table 2, **Overall Peaks for all the Models.**

scores were dull. EfficientNet with AdaDelta fetched results that were second-all-time low accuracies of all the time after LeNet-5 with AdaDelta. RMSprop and Adam for EfficientNet showed results on par with the first two optimizers. The specificity as observed earlier, was seen to be consistently above 0.8 in any given case. This is the model in which it was seen to be performing poorly in terms of all the hyperparameter variations. It can be noted that the model is ineffective in classifying a 498-image brain tumor dataset as the model was created with the intention to be used for datasets with a larger number of images.

The peaks observed in all the architectures are given in Table 2. It can be seen that the best performance was highlighted in yellow and the second-best performance was highlighted in pinkish red. The AlexNet and ResNet-34 fared well in classifying our dataset effectively. Although the accuracy values of both these architectures almost were similar, ResNet-34 takes a slight lead in terms of validation and test losses being lower.

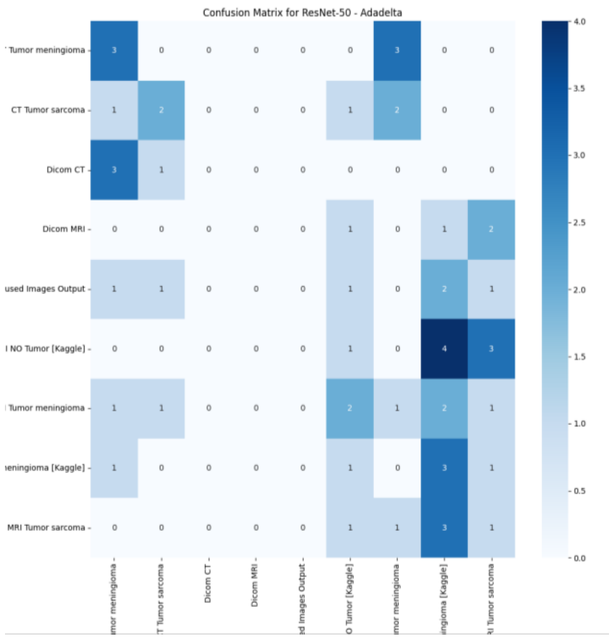

Figure 5, **A Sample Confusion Matrix** for ResNet-34 architecture with AdaDelta optimizer.

Figure 5 contains a sample confusion matrix for the ResNet-34 model which has predicted classes in the horizontal axis and true classes in the vertical axis. The performance metrics sensitivity, specificity, precision, and F1 scores were extracted by using the confusion matrices and classification reports.

The graphical analysis was done based on i) a sample of line plots of performance metrics for each architecture and optimizer, with epoch variation shown in Figure 6,
ii) a sample of bar charts of peak points for each architecture based on optimizers with respect to epochs, displayed in Figure 7, iii) a sample of line plots of overall peaks for each model's metrics, using the same kind of representation (e.g., % for accuracies), as shown in Figure 8.

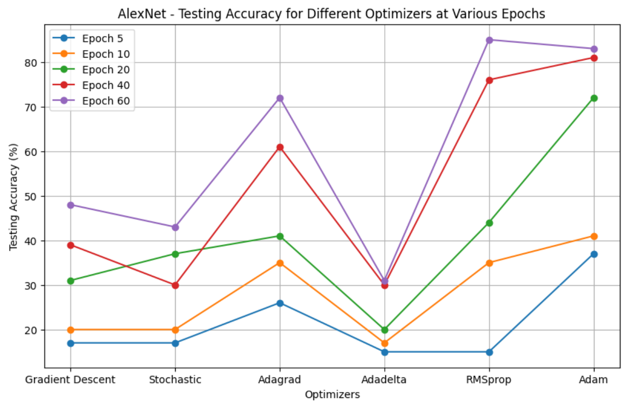

Figure 6, **A Sample Line Plot for AlexNet's Test Accuracy** based on epochs.

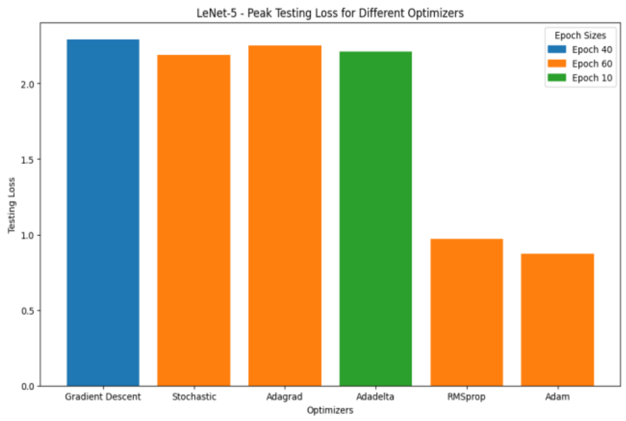

Figure 7, **A Sample Bar Chart for low Testing losses** based on different optimizers in LeNet-5.

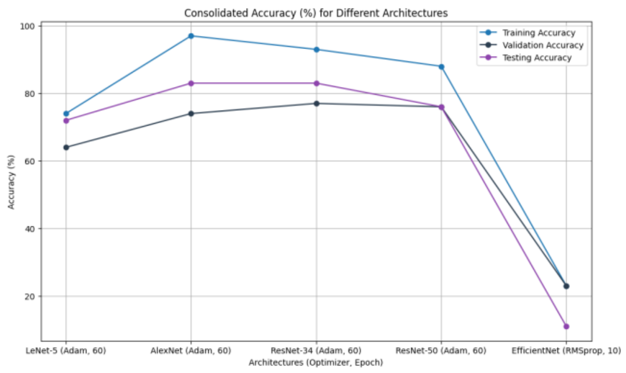

Figure 8, **A Sample Line Plot for Overall Peak Accuracies** based on Architectures, Optimizers, and Epochs.

4. Codes & Files

The codes, folders, and files used and obtained with the experiments can be viewed at: https://github.com/a1906525/CNNs.git.

5. Conclusion and Future Work

To summarize, the brain tumor images were experimented with in the architectures which were indicators of the evolution of the CNNs with time. Each architecture displayed novel outputs which were explored and analyzed for reading the capabilities of the architectures with the efficiencies of the hyperparameters. Overall, it was found that the ResNet models and the AlexNet with were mostly successful in learning and predicting the data while the LeNet-5 and EfficientNet models were indicators of what may probably be areas of concern that one must shed light upon to explore the shortcomings. The optimizers, RMSProp and Adam were comparatively successful in enhancing a model's capabilities which were reflected in the performance metrics. Furthermore, the work can be extended with a larger dataset by subjecting it to even more intense preprocessing for the poorly performed architectures with even a better hyperparameter configuration. This study can be presented to oncologists for precise detection of brain tumors which (if present) may be identified early to avoid serious health implications. The best-performed architectures namely AlexNet and ResNet 34 and 50 can be integrated with additional layers with modifications on attributes such as filters, weights, and biases and addition or decrement of layers which may serve better for reducing and overcoming the contagious health hazards caused by the maligning brain tumors.

5. References

[1] Y. Lian, A. Almasi, D. B. Grayden, T. Kameneva, A. N. Burkitt, and H. Meffin, "Learning receptive field properties of complex cells in V1," PLOS Computational Biology, vol. 17, no. 3, p. e1007957, Mar. 2021, doi: https://doi.org/10.1371/journal.pcbi.1007957. 1

[2] S. Joshi, M. Manu, and A. Mittal, "A Review of the Evolution and Applications of Convolutional Neural Network (CNN)," pp. 1109–1114, Jul. 2023, doi: https://doi.org/10.1109/icecaa58104.2023.10212250. 1

[3] Zoheb Abai, "A Decade of Deep CNN Archs. - AlexNet (ILSVRC Winner 2012)," DEV Community, Jul. 18, 2020. https://dev.to/zohebabai/alexnet-ilsvrc-winner-2012-3cfo (accessed Nov. 4, 2024). 1

[4] "Convolutional Neural Network (CNN) Architectures," GeeksforGeeks, Mar. 21, 2023. https://www.geeksforgeeks.org/convolutional-neural-network-cnn-architectures/(accessed Nov. 4, 2024). 1, 2

[5] M. Mandal, "CNN for Deep Learning | Convolutional Neural Networks (CNN)," Analytics Vidhya, May 01, 2021. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/(accessed Nov. 4, 2024). 2

[6] S. Cong and Y. Zhou, "A review of convolutional neural network architectures and their optimizations," Artificial Intelligence Review, Jun. 2022, doi: https://doi.org/10.1007/s10462-022-10213-5. 2

[7] GeeksforGeeks, "Introduction to Convolution Neural Network," GeeksforGeeks, Mar. 14, 2024. https://www.geeksforgeeks.org/introduction-convolution-neural-network/ (accessed Nov. 4, 2024). 2

[8] E. Tuba, N. Bačanin, I. Strumberger, and M. Tuba, "Convolutional Neural Networks Hyperparameters Tuning," Artificial Intelligence: Theory and Applications, pp. 65–84, 2021, doi: https://doi.org/10.1007/978-3-030-72711-6_4. 2

[9] D. R. Sarvamangala and R. V. Kulkarni, "Convolutional neural networks in medical image understanding: a survey," Evolutionary Intelligence, vol. 15, Jan. 2021, doi: https://doi.org/10.1007/s12065-020-00540-3. 2

[10] C. F. Fraser et al., "Optimizing Artificial Neural Networks to Minimize Arithmetic Errors in Stochastic Computing Implementations," Electronics, vol. 13, no. 14, pp. 2846–2846, Jul. 2024, doi: https://doi.org/10.3390/electronics13142846. 2, 3

[11] C. Hetterich, "Point Cloud Network: An Order of Magnitude Improvement in Linear Layer Parameter Count," arXiv.org, 2023. https://arxiv.org/abs/2309.12996 (accessed Nov. 5, 2024). 3

[12] Lokesh Borawar and R. Kaur, "ResNet: Solving Vanishing Gradient in Deep Networks," Lecture notes in networks and systems (Online), pp. 235–247, Jan. 2023, doi: https://doi.org/10.1007/978-981-19-8825-7_21. 3

[13] "Revisiting ResNets: Improved Training and Scaling Strategies," ar5iv. https://ar5iv.labs.arxiv.org/html/2103.07579 (accessed Nov. 5, 2024). 3

[14] C.-C. Wang, C.-T. Chiu, and J.-Y. Chang,

"EfficientNet-eLite: Extremely Lightweight and Efficient CNN Models for Edge Devices by Network Candidate Search," arXiv.org, 2020. https://arxiv.org/abs/2009.07409 (accessed Nov. 5, 2024). 4

[15] S. Shafi and A. Assad, "Exploring the Relationship Between Learning Rate, Batch Size, and Epochs in Deep Learning: An Experimental Study," Lecture notes in networks and systems, pp. 201–209, Jan. 2023, doi: https://doi.org/10.1007/978-981-19-6525-8_16. 4

[16] A. Mortazi, V. Cicek, E. Keles, and U. Bagci, "Selecting the best optimizers for deep learning–based medical image segmentation," Frontiers in Radiology, vol. 3, Sep. 2023, doi: https://doi.org/10.3389/fradi.2023.1175473. 4

[17] Ayman Al-Kababji, Faycal Bensaali, and Sarada Prasad Dakua, "Scheduling Techniques for Liver Segmentation: ReduceLRonPlateau vs OneCycleLR," pp. 204–212, Jan. 2022, doi: https://doi.org/10.1007/978-3-031-08277-1_17. 4

[18] S. Gonzalez and Risto Miikkulainen, "Improved Training Speed, Accuracy, and Data Utilization Through Loss Function Optimization," arXiv (Cornell University), Jul. 2020, doi: https://doi.org/10.1109/cec48606.2020.9185777. 4

[19] E. R. Collins, "5 Best Ways to Evaluate Your Model with Keras in Python – Be on the Right Side of Change," Finxter.com, Mar. 08, 2024. https://blog.finxter.com/5-best-ways-to-evaluate-your-model-with-keras-in-python/ (accessed Nov. 6, 2024). 4

[20] S. Thakur and A. Kumar, "X-ray and CT-scan-based automated detection and classification of covid-19 using convolutional neural networks (CNN)," Biomedical Signal Processing and Control, vol. 69, p. 102920, Aug. 2021, doi: https://doi.org/10.1016/j.bspc.2021.102920. 4

[21] M. Xu, S. Yoon, A. Fuentes, and D. S. Park, "A Comprehensive Survey of Image Augmentation Techniques for Deep Learning," Pattern Recognition, vol. 137, p. 109347, May 2023, doi: https://doi.org/10.1016/j.patcog.2023.109347. 5

[22] I. Zeger and S. Grgic, "An Overview of Grayscale Image Colorization Methods," Sep. 2020, doi: https://doi.org/10.1109/elmar49956.2020.9219019. 5

[23] S. A. Hicks et al., "On evaluation metrics for medical applications of artificial intelligence," Scientific Reports, vol. 12, no. 1, p. 5979, Apr. 2022, doi: https://doi.org/10.1038/s41598-022-09954-8. 6

[24] O. A. Montesinos López, A. Montesinos López, and J. Crossa, "Overfitting, Model Tuning, and Evaluation of Prediction Performance," Multivariate Statistical Machine Learning Methods for Genomic Prediction, pp. 109–139, 2022, doi: https://doi.org/10.1007/978-3-030-89010-0_4. 6