Fetal Brain Development Monitoring using Ultrasound Images: A U-Net based approach

Jai Tyagi[1] **Manya Goyal**[2] **Rahul Baliyan**[3]
Bhagwan Parshuram Institute Of Technology[1,2,3]

Abstract

We find out the effectiveness of the proposed ideas in the neural network architecture GoogLeNet, we modify and implement the architecture to attempt to achieve improved utilization of computing resources inside the neural network for classification for smaller number of classes without overfitting, achieving successful utilization of the neural network for niche tasks. In this paper we present the idea of using a modified version of the GoogLeNet neural network architecture and attempt make it converge early using batch normalization for applications in niche classification tasks which require almost near perfect accuracy of prediction. We find the results of training on an image dataset of MRI scans for predicting classes by making the architecture bigger and helping it converge early using batch normalization achieving satisfactory performance in classification tasks.

1 Introduction

In this work, we try to find out the effectiveness of the main ideas proposed in the original paper of GoogLeNet in [1] by analyzing the former's architecture and the changes made to the neural network architecture proposed in the same by applying the architecture for Alzheimer's Classification on a large dataset. When the first Inception architecture was published in 2014, the neural network space for image classification was progressing forward at a very dramatic pace. At the time, GoogLeNet architecture stood out as one of the most prominent ideas in image classification when the creators were able to achieve winning results in the ILSVRC 2014. The Inception architecture achieved exceptional results in the challenge with almost 12x less parameters than the already exceptional architecture [4] by Krizhevsky et. al which preceded them as the winners two years at that time.

The main assumptions of the authors were that the "biggest gains in object-detection have not come from the utilization of deep networks alone or bigger models, but from the synergy of deep architectures and classical computer vision".

In this work, we use a modified form of the Inception network with newer techniques to try and achieve almost perfect levels of accuracy on datasets which hold importance in the real world for real time applications in many domains, taking Alzheimer's classification as a case study for the same. Several prior works regarding this case study which have inspired our work and provided motivation are [18, 17, 16].

We explain the reasoning behind using the GoogLeNet as the inspiration for this modified version of the model which has been able to inspire our design of the model and also we will demonstrate the results of trying different versions of the model out on our task as experiments and gather as well as compare the performance of three main variants of the architecture names GoogLeNet Implementation, BN GoogLeNet and ExpGoogLeNet.

2 GoogLeNet as Inspiration

2.1 GoogLeNet for size handling

The dimension of a neural network plays a crucial role, impacting not just its overall functionality but also its susceptibility to overfitting on specific datasets. A larger network tends to lean towards overfitting, as enlarging the neural network involves increasing both its depth and width, subsequently upping the count of parameters and thereby elevating the likelihood of overfitting and slower training. In this study, our focus is on examining how modifications to a neural network, inspired by the original GoogLeNet, can yield outcomes in diverse and narrower application domains compared to the original.

It became imperative to seek solutions that could help decrease the training duration while permitting the neural network to adapt to a relatively smaller dataset without compromising accuracy and training efficiency. Given GoogLeNet's advocacy for a neural network with an extensive amount of parameters, our primary concern was to prevent overfitting on the provided data. To mitigate the overfitting issue and reduce training time, we experimented with the integration of batch normalization [2] to the network. Similar applications such as this one can be found in the works of [7, 9, 10].

2.2 Batch Normalization

Batch normalization can be applied to individual layers or all layers. In each training iteration, the inputs undergo normalization—subtracting their mean and dividing by the standard deviation. Following this, a scale coefficient is applied, along with an offset to restore lost degrees of freedom. The working of the Batch Normalization is as follows :

Inputs	Values of x over a mini-batch: B = $\{x_{1....m}\}$; Parameters to be learned: $\boldsymbol{\gamma}, \boldsymbol{\beta}$
Outputs	$\{y_i = \mathbf{BN}_{\gamma,\beta}(x_i)\}$
Mini-Batch Mean	$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
Mini-Batch Variance	$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
Normalize	$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$
Scale and Shift	$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathbf{BN}_{\gamma,\beta}(x_i)$

Table 1: Batch Normalizing Transform, applied to activation x over a mini-batch

It has been proven to significantly enhance convergence rates without introducing overfitting, as demonstrated in [2]. Thus, in our pursuit of training the network to achieve notable advancements in specific application domains, we turned to batch normalization as a viable technique for implementation in our network.

There are several prior works which have inspired our use of batch normalization in this work. The works of [9, 10] provided a fair incentive for the usage of batch normalization. As well as the works of [11, 12, 13, 14] provided motivation for using convolutional architectures for the problem at hand.

Our study introduces three neural network architectures, differing in the number of batch normalization layers. The three proposed architectures are as follows: we initially conduct experiments with our modified GoogLeNet, which lacks batch normalization layers. Subsequently, we repeat the experimentation with successive models, gradually increasing the number of batch normalization layers while maintaining a constant network parameter count of ∼8.3M.

A simple representation of batch normalization for a minibatch $\mathcal{B}$, and input $\mathbf{x} \in \mathcal{B}$ to Batch Normalization (BN). And thus, so the batch normalization will be defined as [3] :

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta}.$$

2.3 Points of importance and motivations

The architecture proposed just like it's motivation is one which carries quite a lot of parameters, and thus along with the increase in size of the architecture there are some complications which had to be taken care of while implementation :

1. The increased size of the resources needs increased computation.
2. Due to limited computational budget, efficient distribution of resources is required.

To solve both of the problems, the Inception module is used as a sparse alternative for the fully connected architecture of the neural network which is connected through multiple layers in between. For solving these issues, and add the element of sparsity, the Inception module is used in the network containing convolutional and pooling layers which are concatenated in the end. In the original work on the GoogLeNet both the naïve and the dimension reduction forms are present [1].

2.4 Inception module

The Inception module shown in the work [4] and used in the original GoogLeNet is a combination of convolutional and pooling layers as discussed. The module in this case consists of 1 x 1 convolutions which are done before the computationally heavy 3 x 3 and 5 x 5 convolutions. We use a similar approach with the addition of a batch normalization layer before concatenating all of the layers at the end. We use two variations of this where one module contains a single batch normalization layer each after the 3 x 3 and 5 x 5 convolutions and the second which contains one batch normalization layer after each layer before concatenation.

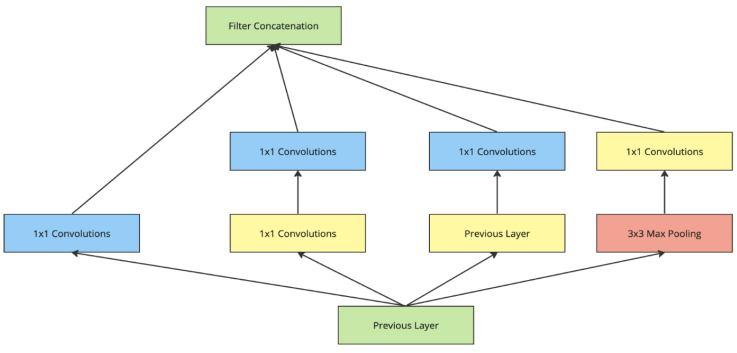

Figure 1: Inception Module

The other two different Inception Modules we have used in our experiments are implementations of the same along with layers of batch normalization. The first implementation BN GoogLeNet contains 2 batch normalization layers on top of the computaionally expensive 3 x 3 and 5 x 5 convolutions, and the ExpGoogLeNet contains 4 batch normalization layers each before the final concatenation of the layers.

Both of the Inception modules are designed as follows :

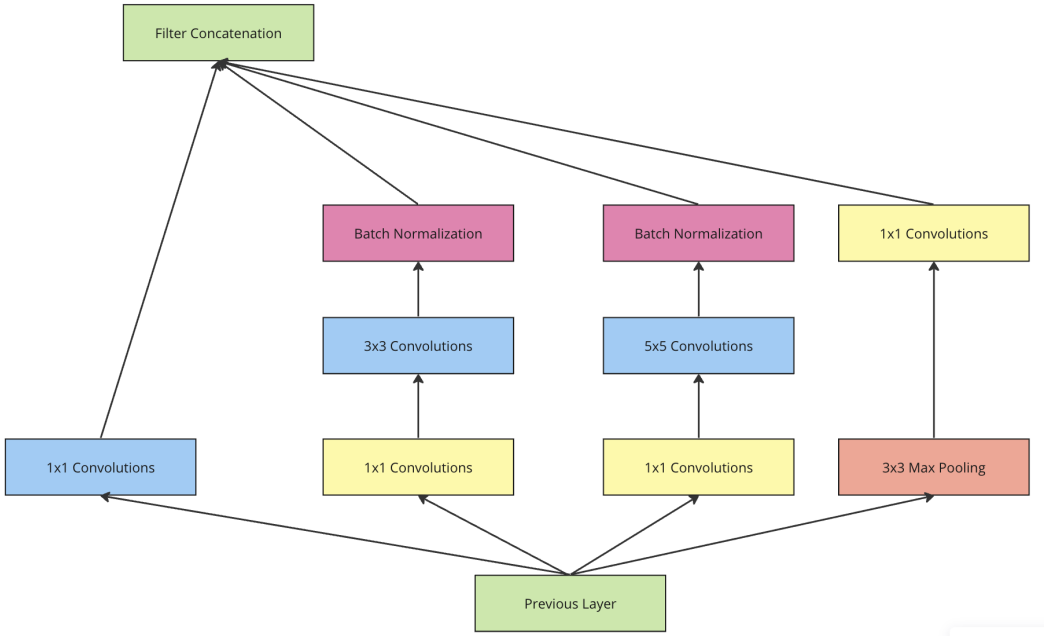

Figure 2: BN GoogLeNet Inception Module

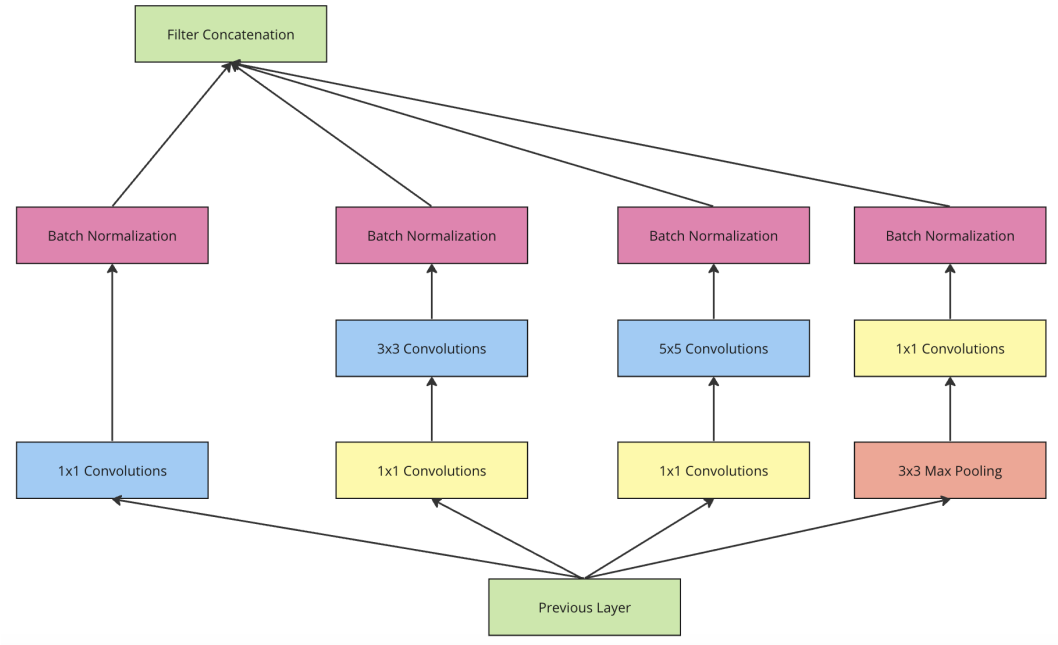

Figure 3: ExpGoogLeNet Inception Module

3 Dataset

3.1 Overview of the dataset :

The dataset used for this series of experiments is a collection of MRI images of the brain. These images have been divided into 4 classes :

- Mild Demented
- Moderate Demented
- Non Demented
- Very Mild Demented

It has a mixture of images containing both augmented and original images. The dataset is available on Kaggle for open source use [5, 6].

The classification of the number images and classes in the dataset are as follows :

Class	**Number Of Images**
Mild Demented	8960
Moderate Demented	6464
Non Demented	9600
Very Mild Demented	8960

Table 2: Overview Of Distribution Of MRI Images

Along with this a bar chart describing the classes and images is shown as follows :

Figure 4: Distribution Of Images in the dataset

Along with this, some sample images from the dataset are as follows :

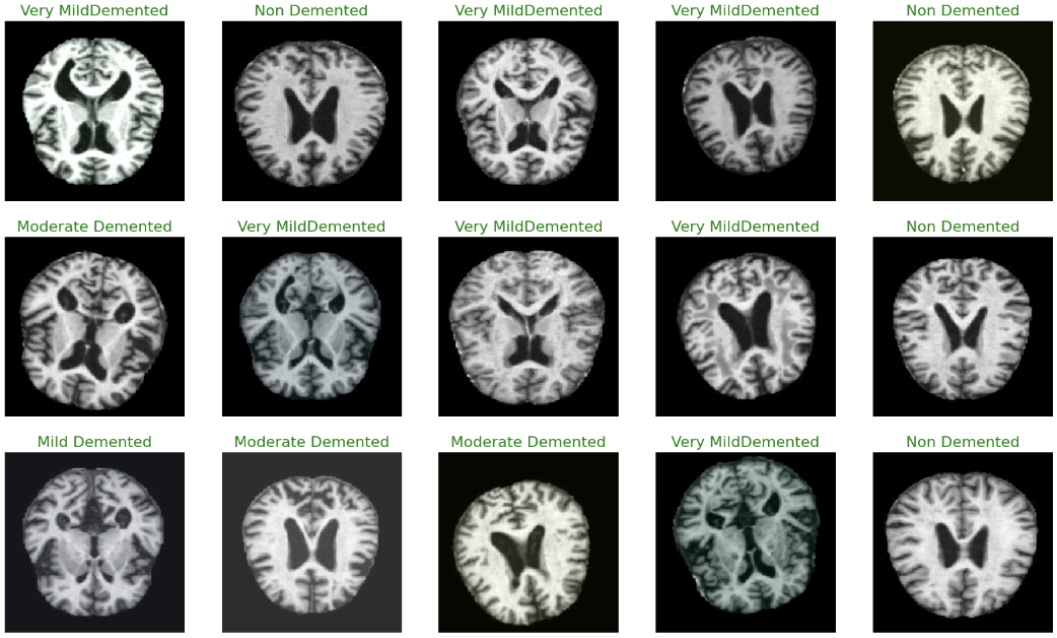

Figure 5: Sample images of classes in the dataset

This is a representation of the images sampled randomly from the dataset. It depicts the classes and the types of images in the classes present.

3.2 Data Splits

The dataset is split into three parts for training, testing and validation each containing a certain number of images. The dataset is distributed such that both the augmented and the original images which have been used for eliminating class imbalance are sampled together from the dataset.

The training set is sampled with the intention to keep about 62% of the images, along with the validation and the testing set containing images from the remaining as 15% and 23% respectively.

A table containing brief description of the images is shown as follows in the form of a table :

Sets	**Number Of Images**
Training Set	27187
Validation Set	6797
Testing Set	10196

Table 3: Split of data into training, testing and validation

3.3 Data Preprocessing and Ingestion

The TensorFlow image data generator is used for the ingestion and most preprocessing functions. The process is as follows :

- Images are rescaled by a factor of 1/255.
- Then the images are resized to a 244 x 244 x 3 resolution.
- The color mode is kept RGB and the images are sampled without shuffling.
- Batch size is kept 32.

The images when ingested into the image data generator is rescaled by the factor of 255. This rescaling is done with keeping in consideration the fact that we would like to keep all of the images in the RGB color space. Owing to the fact that the RGB color space consists of three channels, the rescaling performs a normalization of the image to be between 0 and 1.

After normalizing the images are then rescaled to be 224 x 224, mainly because of the fact that it matches well with the functioning of the image data generator and that most image processing applications prefer that 224 x 224 x 3 scale and thus can be considered a norm to be followed as it helps to be aligned with most practices followed currently.

We sample the images wthout any shuffling from the directories and we process them in a batch of size 32.

A description of the pre-processing process is described as follows :

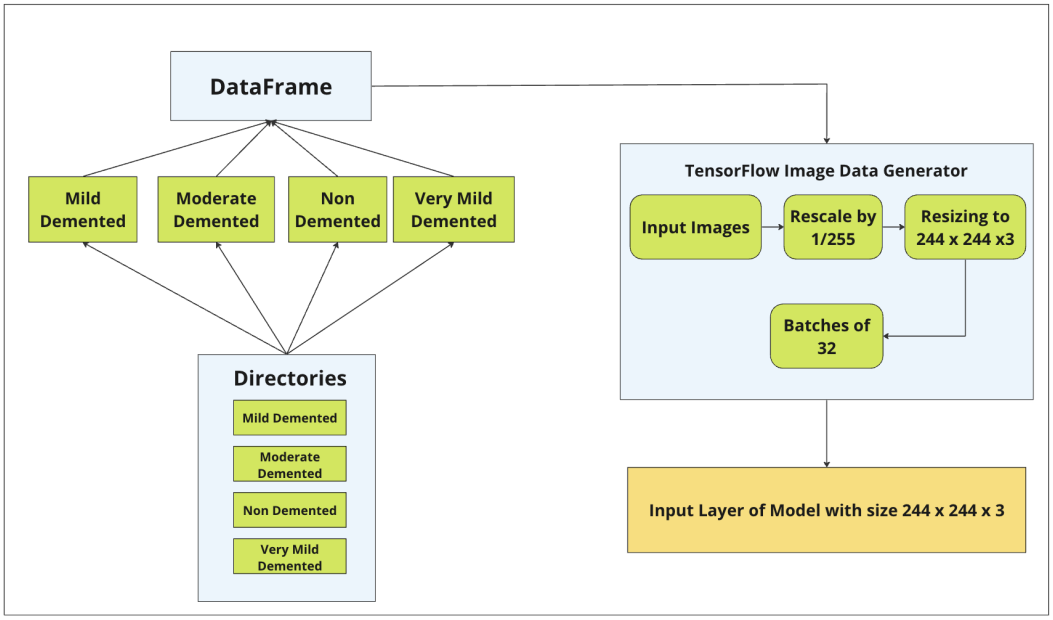

Figure 6: Pre-processing of Image Data

The process is as follows :

- The images in the respective directories of the classes are first matched with the labels and then kept in a dataframe for retrieval.
- From the dataframe, the images along with the labels are then retrieved.
- The images are then rescaled, resized and made into batches of 32.

4 Model and training specifications

4.1 Model Specifications

The architecture of the ExpGoogLeNet is designed with keeping in mind the sparsity of the network as well as keeping the number of parameters to an optimum value of $\sim$8.3M. The network is 21 layers deep and takes inspiration from the original [1], conataining "1 x 1", "3 x 3", "3 x 3 Reduce", "5 x 5 reduce" and "5 x 5" convolutions in the Inception module of the ExpGoogLeNet which are followed by the several pooling layers present.

The layers of batch normalization, convloutions and max pooling, act as intermediates between the modules as connections between them. After the connections, a global average pooling layer followed by a dropout of 0.4, a linear layer and softmax.

The amount of parameters and the overall description of the number of parameters in the layers is illustrated as follows :

Type	Patch size/stride	#1 x 1	# 3 x 3 Reduce	# 3 x 3	# 5 x 5 Reduce	# 5 x 5	Pool proj
convolution	7 x 7 / 2						
max pool	3 x 3 / 2						
convolution	1 x 1						
convolution	3 x 3 / 1		128	512			
batchnorm							
max pool	3 x 3 / 2						
Inception 1(a)		64	96	128	16	32	32
Inception 1(b)		256	256	192	32	96	64
max pool	3 x 3 / 2						
Inception 2(a)		192	96	208	16	48	64
Inception 2(b)		160	112	224	24	64	64
Inception 2(c)		256	256	512	24	64	64
Inception 2(d)		112	144	288	32	64	64
Inception 2(e)		512	160	320	32	128	128
max pool	3 x 3 / 2						
Inception 3(a)		256	160	320	32	128	128
Inception 3(b)		384	192	384	48	128	128
global avg pool							
dropout (0.4)							
linear							
softmax							
Total Parameters					8,348,020		
Trainable Parameters					8,331,028		

Figure 7: Overview of Layers and Parameters

An additional change in the network is the removal of auxiliary classifiers. In the original paper the auxiliary classifiers are used as a means to propagate gradients back to the layers. Given the large depth of the network, it was important that the gradients of the later layers also be propagated back again since they may be able to produce features in the middle of the network which may be discriminative and so the auxiliary classifiers were used as a means to calculate the loss along with taking in consideration the losses produced by the auxiliary classifiers as well, taking them weighted by 0.3 and at the time of inference the auxiliary classifiers were discarded.

In our case, we eliminated the requirement of auxiliary classifiers given the fact that batch normalization takes care of all the considerations related to the gradients not being propagated and also helps in the classifier being able to converge in mere 40 epochs with a testing accuracy of 99.1% as opposed to the network without batch normalization which ceases to converge and stops early after 16 epochs with just about $\sim$28%.

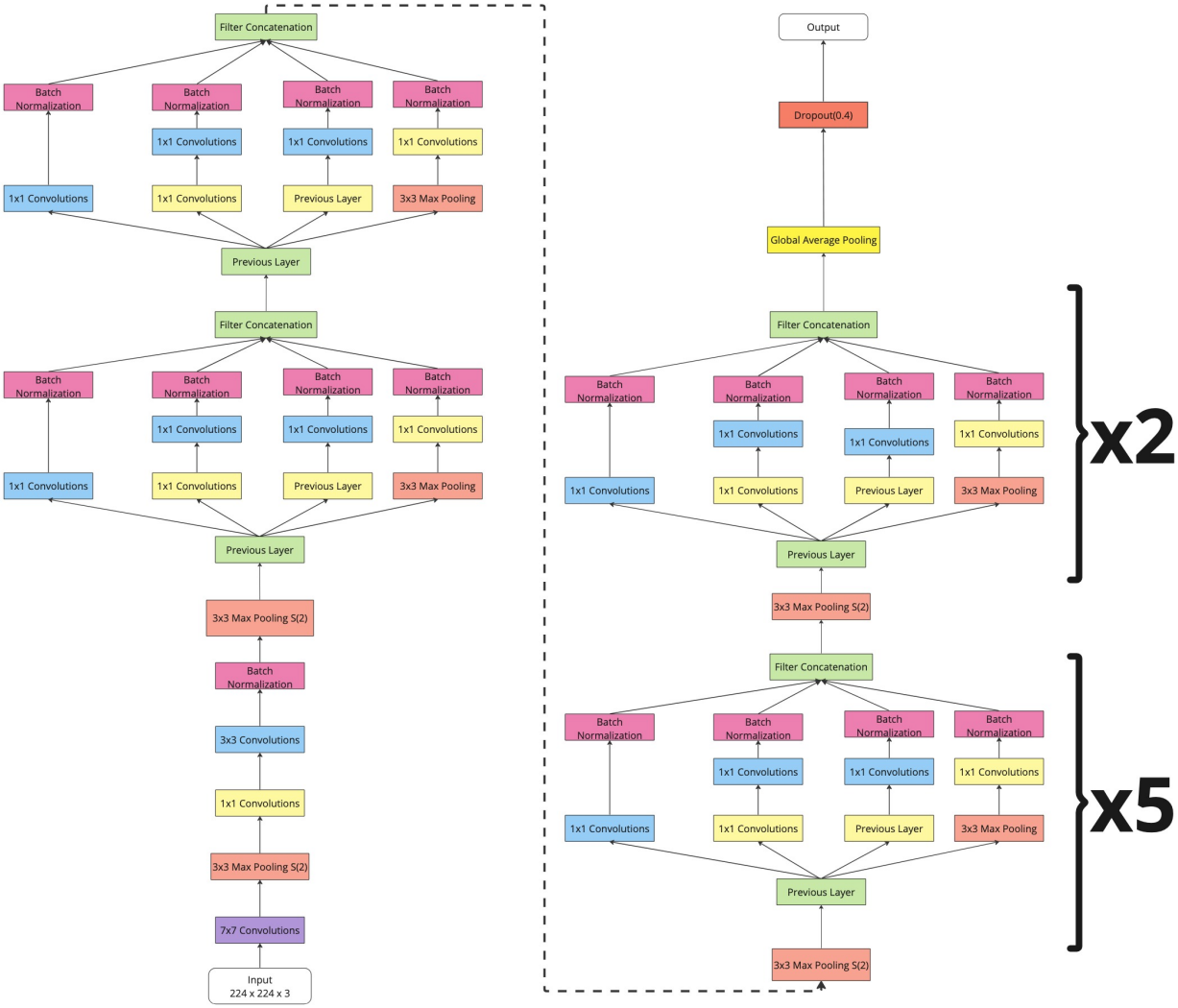

Figure 8: ExpGoogLeNet Network

4.2 Training Specifications

For the training, the categorical cross entropy loss is used, given as :

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c} \cdot \log(\hat{y}_{i,c})$$

where the $\mathcal{L}_{\text{CE}}$, N is the number of samples in the dataset, C is the number of classes, $y_{i,c}$ is the true label for sample i and class c, and $\hat{y}_{i,c}$.

The optimizer used is Adam, with a learning rate of 0.001. A brief description of Adam is as follows:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{1}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{2}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \tag{5}$$

Along with a callback list parameter which monitors max validation accuracy with a patience of 15 and stops.

4.3 Other Comparative Models and Results

Along with the described final model ExpGoogLeNet, we tried two other models with similar architectures, one was our implementation of GoogLeNet and the other was GoogLeNet along with batch normalization which we named BN GoogLeNet architecture. We trained all three of the models with the same training specifications and the results of the three are as follows :

Model	Classes	Precision	Recall	F1 Score	Support	Accuracy
GoogLeNet without BN	Mild Demented	-	-	-	-	~28%
	Moderate Demented	-	-	-	-	
	Non Demented	-	-	-	-	
	Very Mild Demented	-	-	-	-	
BN GoogLeNet	Mild Demented	0.96	0.99	0.98	2693	96.7%
	Moderate Demented	1.00	0.99	0.99	1977	
	Non Demented	0.93	0.98	0.96	2811	
	Very Mild Demented	0.99	0.92	0.95	2715	
ExpGoogLeNet	Mild Demented	1.00	0.99	1.00	2693	99.2%
	Moderate Demented	1.00	1.00	1.00	1977	
	Non Demented	0.99	0.99	0.99	2811	
	Very Mild Demented	0.99	0.99	0.99	2715	

Table 4: Comparison between models and result

5 Conclusion

Through this series of experiments leveraging multiple deep neural networks using sparse representations for achieving better performance on niche tasks such as Alzheimer's class prediction. In these experiments we achieved results by modifying the GoogLeNet architecture using some newer techniques such as batch normalization which helped us fit the data on the neural network efficiently in a considerably lesser amount amount of time. The experimental changes introduced in our implementations of the architecture have led us to realise that deeper neural networks with sparse representations are good learners in presence of techniques such as batch normalization which help to fit such data to a very large network while being computationally inexpensive.

References

[1] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 07-12-June-2015. https://doi.org/10.1109/CVPR.2015.7298594

[2] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. 32nd International Conference on Machine Learning, ICML 2015, 1.

[3] Batch Normalization `http://d2l.ai/chapter_convolutional-modern/batch-norm.html`

[4] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 2.

[5] Alzheimer's Dataset (4 class of Images) `https://www.kaggle.com/datasets/tourist55/alzheimers-dataset-4-class-of-images`

[6] Augmented Alzheimer MRI Dataset `https://www.kaggle.com/datasets/uraninjo/augmented-alzheimer-mri-dataset`

[7] Darma, A. S., & Mohamad, F. S. B. (2021). The Regularization Effect of Pre-activation Batch Normalization on Convolutional Neural Network Performance for Face Recognition System Paper. International Journal of Advanced Computer Science and Applications, 12(11). https://doi.org/10.14569/IJACSA.2021.0121135

[8] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11). https://doi.org/10.1109/5.726791

[9] Thakkar, V., Tewary, S., & Chakraborty, C. (2018). Batch Normalization in Convolutional Neural Networks - A comparative study with CIFAR-10 data. Proceedings of 5th International Conference on Emerging Applications of Information Technology, EAIT 2018. https://doi.org/10.1109/EAIT.2018.8470438

[10] Wang, S. H., Tang, C., Sun, J., Yang, J., Huang, C., Phillips, P., & Zhang, Y. D. (2018). Multiple sclerosis identification by 14-layer convolutional neural network with batch normalization, dropout, and stochastic pooling. Frontiers in Neuroscience, 12(NOV). https://doi.org/10.3389/fnins.2018.00818

[11] Antunes, A., Ferreira, B., Marques, N., & Carriço, N. (2023). Hyperparameter Optimization of a Convolutional Neural Network Model for Pipe Burst Location in Water Distribution Networks. Journal of Imaging, 9(3). https://doi.org/10.3390/jimaging9030068

[12] Han, X., Hu, Z., Wang, S., & Zhang, Y. (2023). A Survey on Deep Learning in COVID-19 Diagnosis. In Journal of Imaging (Vol. 9, Issue 1). https://doi.org/10.3390/jimaging9010001

[13] Jasim, R. M., & Atia, T. S. (2023). An evolutionary-convolutional neural network for fake image detection. Indonesian Journal of Electrical Engineering and Computer Science, 29(3). https://doi.org/10.11591/ijeecs.v29.i3.pp1657-1667

[14] Li, H., Tan, Y., Miao, J., Liang, P., Gong, J., He, H., Jiao, Y., Zhang, F., Xing, Y., & Wu, D. (2023). Attention-based and micro designed EfficientNetB2 for diagnosis of Alzheimer's disease. Biomedical Signal Processing and Control, 82. https://doi.org/10.1016/j.bspc.2023.104571

[15] Lin, H. I., Mandal, R., & Wibowo, F. S. (2024). BN-LSTM-based energy consumption modeling approach for an industrial robot manipulator. Robotics and Computer-Integrated Manufacturing, 85. https://doi.org/10.1016/j.rcim.2023.102629

[16] Saeedi, S., Rezayi, S., Keshavarz, H., & R. Niakan Kalhori, S. (2023). MRI-based brain tumor detection using convolutional deep learning methods and chosen machine learning techniques. BMC Medical Informatics and Decision Making, 23(1). https://doi.org/10.1186/s12911-023-02114-6

[17] Priyatama, A., Sari, Z., & Azhar, Y. (2023). Deep Learning Implementation using Convolutional Neural Network for Alzheimer's Classification. Jurnal RESTI (Rekayasa Sistem Dan Teknologi Informasi), 7(2). https://doi.org/10.29207/resti.v7i2.4707

[18] Zhao, Z., Chuah, J. H., Lai, K. W., Chow, C. O., Gochoo, M., Dhanalakshmi, S., Wang, N., Bao, W., & Wu, X. (2023). Conventional machine learning and deep learning in Alzheimer's disease diagnosis using neuroimaging: A review. In Frontiers in Computational Neuroscience (Vol. 17). https://doi.org/10.3389/fncom.2023.1038636

A APPENDIX A : BN GoogLeNet and GoogLeNet Implementation

A.1 BN GoogLeNet

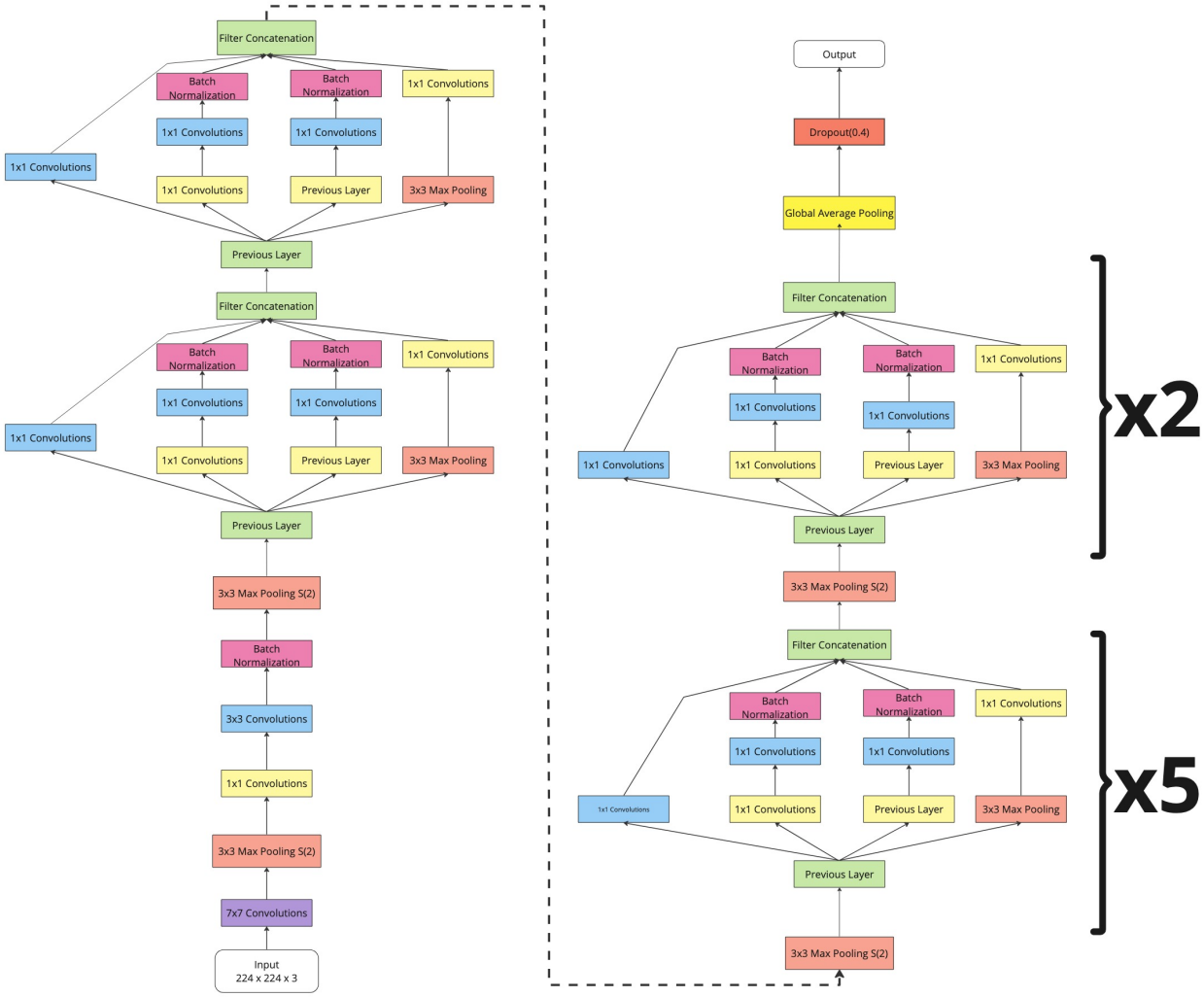

Figure 9: BN GoogLeNet

In this implementation the Inception module used is the one referenced here 2 and is represented with a " ' " on the top of each of the modules. Model specifications are as follows :

Type	Patch size/stride	#1 x 1	# 3 x 3 Reduce	# 3 x 3	# 5 x 5 Reduce	# 5 x 5	Pool proj
convolution	7 x 7 / 2						
max pool	3 x 3 / 2						
convolution	1 x 1						
convolution	3 x 3 / 1		64	192			
batch norm							
max pool	3 x 3 / 2						
Inception 1'(a)		64	96	128	16	32	32
Inception 1'(b)		256	256	192	32	96	64
max pool	3 x 3 / 2						
Inception 2'(a)		192	96	208	16	48	64
Inception 2'(b)		160	112	224	24	64	64
Inception 2'(c)		256	256	512	24	64	64
Inception 2'(d)		112	144	288	32	64	64
Inception 2'(e)		512	160	320	32	128	128
max pool	3 x 3 / 2						
Inception 3'(a)		256	160	320	32	128	128
Inception 3'(b)		384	192	384	48	128	128
global avg pool							
dropout (0.4)							
linear							
softmax							

Table 5: Model specifications for BN GoogLeNet

A.2 GoogLeNet Implementation

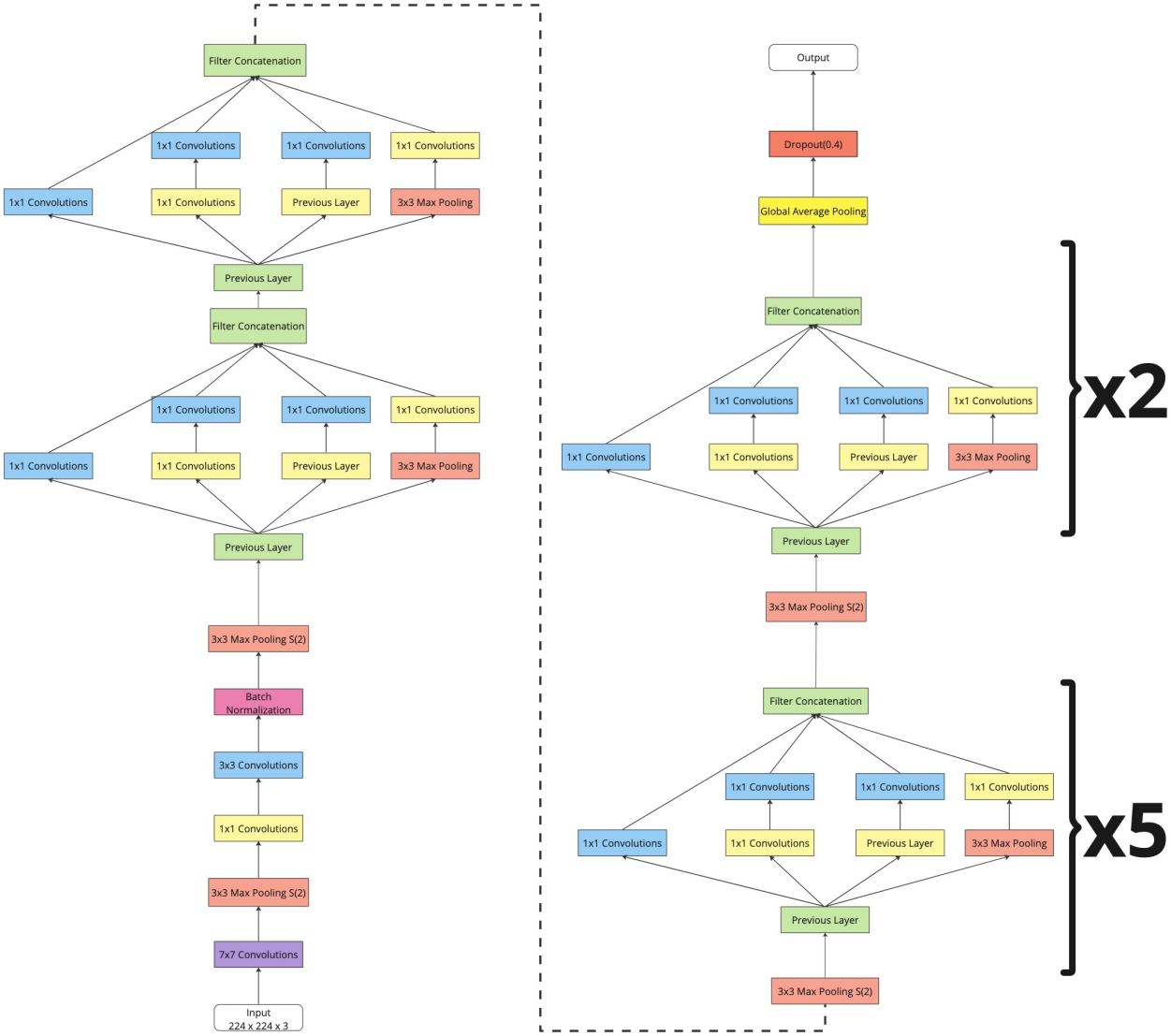

Figure 10: GoogLeNet Implementation

Similar to the BN GoogLeNet the GoogLeNet Implementation this implementation uses the original Inception module referenced here in Fig 1. Model specifications are as follows :

Type	Patch size/stride	#1 x 1	# 3 x 3 Reduce	# 3 x 3	# 5 x 5 Reduce	# 5 x 5	Pool proj
convolution	7 x 7 / 2						
max pool	3 x 3 / 2						
convolution	1 x 1						
convolution	3 x 3 / 1		64	192			
max pool	3 x 3 / 2						
Inception 1"(a)		64	96	128	16	32	32
Inception 1"(b)		256	256	192	32	96	64
max pool	3 x 3 / 2						
Inception 2"(a)		192	96	208	16	48	64
Inception 2"(b)		160	112	224	24	64	64
Inception 2"(c)		256	256	512	24	64	64
Inception 2"(d)		112	144	288	32	64	64
Inception 2"(e)		512	160	320	32	128	128
max pool	3 x 3 / 2						
Inception 3"(a)		256	160	320	32	128	128
Inception 3"(b)		384	192	384	48	128	128
global avg pool							
dropout (0.4)							
linear							
softmax							

Table 6: Model specifications for GoogLeNet Implementation