Convolutional Neural Network-Based Natural Image and MRI Classification Using Gaussian Activated Parametric (GAP) Layer

BIJEN KHAGI AND GOO RAK KWON, (Member, IEEE)
Department of Information and Communication Engineering, Chosun University, Gwangju 61452, South Korea
Dementia Care 3D Medical Image Center, South Korea
Corresponding author: Goo Rak Kwon (grkwon@chosun.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) under Grant NRF-2019R1A4A1029769 and Grant NRF-2021R1I1A3050703.

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the Institutional Review Board (or Ethics Committee) of in IRB Committee of Chosun University under Application No. 2-1041055-AB-N-01-2019-45 / 2020-72, and performed in line with the Declaration of Helsinki.

ABSTRACT We propose a novel gaussian activated parametric (GAP) layer for deep neural networks specifically for CNN. This layer normalizes the feature vector using the Gaussian filter-based un-sharpening technique. The goal of the proposed method is to normalize and activate the initial and intermediate feature layers of a deep CNN so that a customized layer can make the feature more distinguish and layers are smoothly tuned for the target-domain classification. Our experiment shows the use of the proposed layer for normalization gives a result almost similar to that of batch normalization and in few cases, the result is slightly better at the cost of higher training time. To demonstrate the result of the proposed layer we are using 4 layers of an encoder-based network as a base architecture for classification in which the first two normalization layers are GAP layers or BN layers whereas the remaining are BN layers. The result of the experiment is better in classification with two GAP layer as normalization layer in a bulkier dataset like ADNI MRI (3D CNN accuracy: 93.58% vs. 91.89%) and CIFAR-10 (2D CNN accuracy: 75.21% vs. 75.11%), whereas the result is better with replacement of first BN layer with GAP in a smaller dataset like 5-animals dataset (2D CNN accuracy: 62.92% vs 58.48%). Therefore, we suggest the use of single GAP layer normalization for smaller datasets and two GAP normalization layers for bigger dataset training. Also, the proposed method produced better results than the cross-channel normalization-based AlexNet network under scratch training.

INDEX TERMS Activation, convolutional neural network (CNN), normalization layer, correlation, classification, deep neural network (DNN), normalization layer.

I. INTRODUCTION

Deep neural network (DNN) has been the dark horse in the field of machine learning (ML) and deep learning after the success of LeNet-5, an emerging Convolutional neural network (CNN), in 1989 for handwriting recognition [1], [2]. The massive success behind the use of DNN is because of its capacity to accommodate a larger number of trainable model parameters which contributes to the accurate extraction of feature for pattern recognition, image-classification as in ImageNet using AlexNet [3], GoogleNet [4], ResNet [5], object recognition (R-CNN [6], [7]), scene segmentation (SegNet [8], [9]) and other tedious tasks for human perception. The commonly used DNN in image classification or recognition is CNN, with convolution filter as the key feature detector from primary level features like edge, color, corner, line to higher-level features like texture, pattern, shape, etc. for its class identification [10]–[12], [31]. Hence, CNN is basically an image feature extractor. In CNN, the weights of convolution filters are the key parameter to train, and it determines how a particular filter works. Besides the convolution filters, many other learnable layers also participate in weight update during training via backpropagation, so that they all work conjointly to produce final down-sampled features with class-label properties. The training algorithm or the objective function for optimization and the initialization techniques...
are the key components to escort the cross-entropy loss to minima, so that the training can be stopped, i.e., the weights can no more be changed and technically the model/network is said to achieve convergence. Achieving optimal convergence is our primary goal however, only reaching the lowest minima does not guarantee a high success rate in test (unseen) sample classification. i.e., the ‘test error’ can be high with a low ‘training error’. This familiar problem in ML is called overfitting or generalization error, which is the most serious issue in any DNN architecture. Many regularization techniques along with dropout, early stopping, random sampling, etc. are introduced to reduce this generalization error. However, being the state-of-the-art algorithm in several benchmark datasets, numerous ML algorithms are still not fully understood and working as a black box in many tasks, it can be realized by the fact that many standard DNN fails to generalize [13], [36], [37].

Besides, any NN under scratch training is affected by the randomness in the parameter initialization, which brings some short of disparities in the input node distribution of layer during training time and makes it extremely difficult to train networks with saturating nonlinearities considered as covariant shift [14]. This phenomenon considered specifically as an internal covariant shift is some sort of curse for convergence and generality as well. To overcome this, batch normalization (BN) was introduced to minimize the effect of this covariant shift. It also speeded up the training process and cope up with a higher learning rate without exploding gradient. BN uses layer-wise whitening technique image i.e., mean zero and unit variance for normalization and decorrelation, with only two extra parameters per activation one for scaling and the other for shifting. It helps to reduce training time and preserves the interpretation capacity of the network [15]. It might be because the convolution weights are updated so quickly that the weights tend to move towards convergence faster while the validation accuracy still lags far behind. BN was mainly designed for speedy training and reducing internal covariant shift also, the claimed regularization technique was said to eliminate the use of dropout. Because of this BN also claims to reduce the generalization error as dropout does the same. However, at the same time, we observed a prominent overfitting problem with the use of BN in our base architecture. Along with BN, many other lately used activation functions to address these issues of overfitting and covariant shift. For experiments, we are using 4 layers of an encoder-based network as a base architecture with a normalization layer in each encoder. Also, AlexNet with local response normalization [3] which is similar in architecture to our base model is also used for scratch training all dataset to compare the classification result. The proposed method has been mathematically explained via matrix equivalency with the CNN normalization layer. We use three parameters per activation in the proposed layer, one for scaling the original feature, the other for scaling the masked feature, and the last one as offset or bias to shift the output. Our contribution in this paper is:

i. We are proposing a novel Gaussian filter-based normalization layer to integrate with deep neural networks, which we call Gaussian activated parametric (GAP) layer. Also, instead of performing minibatch averaged scaling, we are proposing the same channel mean scaling for normalization within the layer.

ii. Comparative analysis is performed for the proposed GAP layer in alternate to BN in the base architecture to study the feature extraction, histogram analysis, internal covariance problem via correlation test, and overfitting. Along with a comparison of the final classification results.

II. BACKGROUND AND MOTIVATION

It has been observed that the convolutional filter weights in the initial layers of CNN tend to mimic some image filters once the network is fully trained. In Krizhevsky et al. [3] the trained convolutional weights/parameters were mostly the edge detectors and color filters. The edge filters could detect horizontal, vertical, and diagonal edges, besides these filters are translationally invariants in nature and so works spatially for all the input images. Kaimming et al. [16] report the filters of the first convolutional layers were mostly Gabor-like filters such as edge or texture detectors, and the results after full training showed that both positive and negative responses of the filters are revered. With these previous results pertaining to the basic image filters and negative responses being equally important during training, we were interested in designing a layer for early feature detection and quick imitation. For this instead of manually initializing different filters, we were interested in finding filters to smooth the overall result not only the single features like edge, color, blob, etc. Our experiment showed the use of edge detection filters like Sobel, Prewitt, Robert, etc. [17]–[20] works unidirectionally without bringing feature variance and the classification result was also poor. Hence, we need a universal filter to boost up the learned features and can work marginally and cleverly for all types of feature attributes like edge, blob, color, corner, etc. For this, we selected the Gaussian filter. The Gaussian filter can be used as an average smoothing function along with the normalized input. However, using Gaussian smoothing with batch normalization seems inappropriate, so to normalize the features, we use filter/channel mean normalization. In this normalization, the mini-batch properties in not entertained,
instead the mean features from all the filter weights (i.e., convolution weights in all channels) are normalized channel-wise for each batch dimension as described later in detail in section IV. So, the obtained mean feature vectors work only for its channel images without mixing the mini-batch properties. In doing this the injection of noise is reduced, and hence single image property is only summed up. Since we are using the proposed layer as a feature enhancer, it follows the convolution layer to operate on the activated image output.

In an experiment [21] to understand the effect of randomness in NN, the Gaussian function was used to add noise to the input and bring non-uniformity in the signal by adding random pixels to the source image. This destroys the relationship of training pixels with its label. Further to make the experiment more random, the image pixels were re-shuffled and re-sampled from a gaussian distribution. However, the NN was not heavily affected by this and was able to fit the test set. Thanks to the stochastic gradient descent algorithm [22] used during the training. In our case, we have used the gaussian function not as a noise generator, but as the mask generator for the un-sharpening process and the work in channel normalization [41] has motivated us in doing so. We aimed to obtain good training convergence without destroying the correlation property between the input and output of each layer. Instead of performing Gaussian smoothing on the whole image, we have used the Gaussian kernel to generate matching mean and variance to the original image kernel. Kernel size is determined by the size of the preceding convolution filter kernel, to avoid mixing of filter properties during the smoothing process.

III. GAUSSIAN FILTER AND UN-SHARPENING PROCESS

Gaussian filter [23] is a spatially weighted average filter, which works as a point-spread function for any image-pixel distribution. It operates as a non-uniform low pass filter with a higher weight to the central pixel producing a normal distribution of pixels. And its rotationally symmetric nature provides directional unbiasedness for image morphological operation. This property of the Gaussian filter also helps to marginally preserve edges and brightness, while producing smoothing results for image attributes with averaging values. However, the input standard deviation (SD) (σ) has some effect in blurring the image. T. Lindeberg [24] derived the optimal discretized Gaussian kernel and proved Gabor functions can look very much like Gaussian derivatives. In order to operate in the CNN feature matrix, we need a discrete approximation of its kernel. This is done as in equation (1) where i, j, and k represents the matrix row, column, and depth and σ^2 is the input variance which is equal to 1 for standard Gaussian output. The output for 3 x 3 x 3 and 5 x 5 x 5 matrix is shown in plot Figure 1.

$$h(i, j, k, \sigma) = \frac{1}{(\sqrt{2\pi} \sigma)^3} e^{-\frac{(i^2+j^2+k^2)}{2\sigma^2}}$$  \hspace{1cm} (1)

The term \(1/\sqrt{2\pi}\) is a normalization constant that comes from the fact that the integral over the exponential function is not unity. It is shown that the convolution with a Gaussian kernel is a linear operation [23] so in order to bring non-linearity in the system, we need to add some non-linear functions like Leaky ReLU for rectification. A 3D kernel is used in designing GAP layers for both 2D and 3D CNN, in 2D CNN the filter operates with the mean of all convolution-based images, whereas in 3D CNN the mean value is operated in the whole volume itself. However, in both cases, the learning coefficient parameters are updated separately acknowledging the mean effect from its channel filters for the final output.

The un-sharpening process involves three steps i) blurring of image i.e. correlation operation of input image volume with the Gaussian kernels ii) subtraction of original input image volume with its blurred version to generate masking kernel iii) adding the masked version to the original input. Figure 3 illustrates the operation involved inside the layer in detail.

IV. PROPOSED GAUSSIAN ACTIVATED PARAMETRIC (GAP) NORMALIZATION LAYER

A. ARCHITECTURE AND TRAINING

Let us consider the output from the first convolution layer as in base architecture (see Table 1), which acts as input for

![Figure 1. Visualization of Gaussian 3D kernel in the linear plot (a) 3*3*3 kernel with 27 weights matrix (b) 5*5*5 kernel with 125 weights matrix. Please note the higher central value for each corresponding peaks.](image-url)
**TABLE 1.** The base architecture is used for testing the proposed method against BN for classification. Please note that 2D and 3D architecture is different with different activation size. Note: here ‘g1g2b3b4’ architecture indicates 1\textsuperscript{st} normalization GAP (g1), 2\textsuperscript{nd} GAP (g2) and 3\textsuperscript{rd}, 4\textsuperscript{th} both BN as b3 and b4 respectively. Similarly, b1b2b3b4 means all BN and so on. The selection of hyperparameters and activation function is based on our previous work [12].

| S. N | LAYERS NAME | DESCRIPTION (2D BASE ARCHITECTURE) | 2D ACTIVATION SIZE | 2D LEARNABLE PARAMETERS | DESCRIPTION (3D BASE ARCHITECTURE) | 3D ACTIVATION SIZE | 3D LEARNABLE PARAMETERS |
|------|-------------|-----------------------------------|-------------------|------------------------|-----------------------------------|-------------------|------------------------|
| 1    | Input Image | 227×227×3 images with 'zero center' normalization | 227×227×3 | 64×64×64 images with 'zero center' normalization | 64×64×64 | 64×64×64 | 64×64×64 |
|      | Convolution | 32 3×3 convolutions with stride [1 1] and padding 'same' | 227×227×32 | Weights-3×3×3×1 convolutions with stride [1 1] and padding 'same' | 64×64×64 | 64×64×64 | 64×64×64 |
|      | Batch       | Normalization (g1)                 | 227×227×32       | Offset-1×1×1×64 Normalization or activated Normalization | 64×64×64 | 64×64×64 | 64×64×64 |
|      | or          | Leaky ReLU with scale 0.01         | 227×227×32       | Leaky ReLU with scale 0.01 | 64×64×64 | 64×64×64 | 64×64×64 |
|      | Max Pooling | 2×2 max pooling with stride [2 2] and padding [0 0 0 0] | 113×113×32 | 64×5×5×64 convolutions with stride [1 1] and padding [0 0 0 0] | 63×63×63×64 | 63×63×63×64 | 63×63×63×64 |
|      | Convolution | 64 5×5 convolutions with stride [1 1] and padding 'same' | 113×113×64 | Weights-5×5×32×64 convolutions with stride [1 1] and padding 'same' | 63×63×63×64 | 63×63×63×64 | 63×63×63×64 |
|      | Batch       | Normalization (g2)                 | 113×113×64       | Offset-1×1×1×64 Normalization or activated Normalization | 63×63×63×64 | 63×63×63×64 | 63×63×63×64 |
|      | or          | Leaky ReLU with scale 0.01         | 113×113×64       | Leaky ReLU with scale 0.01 | 63×63×63×64 | 63×63×63×64 | 63×63×63×64 |
|      | Max Pooling | 2×2 max pooling with stride [2 2] and padding [0 0 0 0] | 56×56×64 | 64 7×7×7×64 convolutions with stride [1 1] and padding 'same' | 31×31×31×64 | 31×31×31×64 | 31×31×31×64 |
|      | Convolution | 64 7×7 convolutions with stride [1 1] and padding 'same' | 56×56×64 | Weights-7×7×64×64 convolutions with stride [1 1] and padding 'same' | 31×31×31×64 | 31×31×31×64 | 31×31×31×64 |
|      | Batch       | Normalization (b3)                 | 56×56×64          | Batch normalization with 64 channels | 31×31×31×64 | 31×31×31×64 | 31×31×31×64 |
|      | or          | Leaky ReLU with scale 0.01         | 56×56×64          | Leaky ReLU with scale 0.01 | 31×31×31×64 | 31×31×31×64 | 31×31×31×64 |
|      | Max Pooling | 2×2 max pooling with stride [2 2] and padding [0 0 0 0] | 28×28×64 | 2×2 max pooling with stride [3 3] and padding [0 0 0 0] | 10×10×10×64 | 10×10×10×64 | 10×10×10×64 |
TABLE 1. (Continued.) The base architecture is used for testing the proposed method against BN for classification. Please note that 2D and 3D architecture is different with different activation size. Note: here 'g1g2b3b4' architecture indicates 1 \textsuperscript{st} normalization GAP (g1), 2 \textsuperscript{nd} GAP (g2) and 3 \textsuperscript{rd}, 4 \textsuperscript{th} both BN as b3 and b4 respectively. Similarly, b1b2b3b4 means all BN and so on. The selection of hyperparameters and activation function is based on our previous work [12].

| Encoder-4 | Convolution | Weights-9×9×64 | Weights-9×9×64 | Weights-9×9×9×64 | Weights-9×9×9×64 |
|-----------|-------------|----------------|----------------|-------------------|-------------------|
|           | 128 9×9 convolutions with stride [1 1] and padding 'same' | 28×28×128 | 1×1×1×64 | 1×1×1×64 |

| Batch Normalization (b4) | Batch normalization | Offset-1×1×64 | Batch normalization with 64 channels | Offset-1×1×64 |
|-------------------------|---------------------|-------------|-------------------------------|-------------|
|                         | 28×28×128           | Scale-1×1×64 | 10×10×10×64                  | Scale-1×1×64 |

| Leaky ReLU | Leaky ReLU with scale 0.01 | Leaky ReLU with scale 0.01 |
|------------|----------------------------|----------------------------|
|            | 28×28×128                 | 10×10×10×64               |

| Max Pooling | 15×14×128 | 3×3×3×64 |
|-------------|-----------|----------|
| 2x2 max pooling with stride [2 2] and padding [0 0 0 0] | 1×1×1×1152 | 1×1×1×1728 |
| 2x2 max pooling with stride [4 4] and padding [0 0 0 0] | 1×1×1×1152 | 1×1×1×1728 |

| Fully Connected | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|-----------------|----|----|----|----|----|----|----|
| 1152 fully connected layer | 1×1×1×1152 | 1×1×1×1152 | 1×1×1×576 | 1×1×1×1152 | 1×1×1×N | 1×1×1×N | 1×1×N |
| 1728 fully connected layer | 1×1×1×1152 | 1×1×1×1152 | 1×1×1×576 | 1×1×1×1152 | 1×1×1×N | 1×1×1×N | 1×1×N |
| 1728×1 | 864 fully connected layer | 1×1×1×1152 | 1×1×1×1152 | 1×1×1×576 | 1×1×1×1152 | 1×1×1×N | 1×1×1×N | 1×1×N |
| 864×1 |

| SoftMax | Classification | Cross-entropy with 'AD' and 2 other classes | 1×1×3 |
|--------|---------------|------------------------------------------|-----|
| 1×1×N |                |                                          |     |

| N fully connected layer where 'N' is the number of trained class | 1×1×N | 3 fully connected layer | 1×1×3 |
|---------------------------------------------------------------|-------|------------------------|-------|
| 1N×N | 3×864 | 1×1×3 |

Here, the mean value $X_m$ is calculated based on its 3\textsuperscript{rd} dimension i.e., the number of filter or channel. (for 3D CNN it is calculated on the 4\textsuperscript{th} dimension) hence can be represented as a matrix:

$$X_m = \begin{bmatrix} X_1^1 & \cdots & X_1^{32} \\ \vdots & \ddots & \vdots \\ X_{64}^1 & \cdots & X_{64}^{32} \end{bmatrix}$$ (2)

Here, the column matrix on the right side contains the mean value of each batch from different training samples. The dimension now for $X_m = [227 \ 227 \ 32 \ 64]$ which gives the averaged value of ‘same images’ activated ‘differently’ with convolutional filters. It is an empirical mean value of all channels so contains 64 different images for each batch updated during training, and is not used post-training, hence an empirical mean. Similarly, standard deviation is also calculated for $X_s = [227 \ 227 \ 32 \ 64]$. Now we re-center and re-scale $X$ using $X_m$ and $X_s$ as, $X_f = (X - X_m) / X_s$ which produces the averaged mean-centered and 1/Xs scaled output, $X_f$. Please note all the arithmetic operation in the matrix is an element-wise identical dimension operation. Hence the size of $X_f = [227 \ 227 \ 32 \ 64]$. Here in equation (3), $X_f^i$ is the mean from the activated output of the first training images.
FIGURE 2. Training and validation accuracy curve using different normalization schemes for same training environment for (a) 5-animals dataset (b) Caltech-102 (c) CIFAR-10 (d) 3D MRI baseline. In all the graphs training accuracy reaches convergence (100%) faster with 100% BN whereas the validation curve does not follow the training curve for a longer time, which means the weight update process ends sooner and cannot generalize better. In the case with 50% BN and 75% BN, the training curve achieves convergence lately and so does the validation. This helps the network to update weight slowly (at the same learning rate) and hence might reduce the overfitting (although we have a clear case of overfitting due to a large gap in training and validation accuracy in all cases, better architectures might change the result, however in our case we are using our base architecture to test the proposed idea). Here, the CNN with GAP layers tends to reach 100% accuracy slowly so that validation accuracy is addressed for a longer time whereas with BN layers the training accuracy shoots up quickly causing a higher gap (coincidentally we have named GAP for the proposed layer) between the training and validation layer during the early stages of training. Hence the overfitting problem is still not completely tackled in batch normalization. Even though with a low learning rate and higher learn-rate-drop factor the problem still prevails. Concerning this, we want the training convergence to generalize better with the proposed GAP layer. Our result suggests the testing result remains the same or slightly improved in most of the cases and feature extraction is also enhanced.

\[ X_1, X_2, \ldots, X_32 \] and so on. Hence, the variance is nominally very less, i.e., in general, \( \text{var}(X'_f) \approx \text{var}(X^n_c) \). In BN, once the training is finished each BN layer has \( n \) number of trained mean and variance per activation stored in the trained network, which is later used to normalize the input during prediction. However, in our method, the layer does not store the trained mean and S.D instead, during prediction, the weight of convolutional kernels is passed below so the empirical mean for each image is calculated from each channel output, hence no need for pretrained mean and S.D.

Channel normalization [41] standardizes each channel in a CNN, independently for every training example, and scales and shifts the resulting input with a (learnable) scalar. It is comparable to instance normalization [42] and BN for a single training sample. In our case \( X_f \) is the normalized unsharpened version of its original minibatch input, which is the shifted and scaled version of the input \( X \). \( \alpha, \beta \) works as a separate channel standardizing factor for the input and the unsharpened version respectively. However, the pretrained value of \( \alpha, \beta, \) and \( \gamma \) scales each channel separately. \( X_f \) is now spatially correlated with a Gaussian kernel, i.e., filtering is done with the kernel of a size equivalent to the size of the previous convolution filter (here \( 3 \times 3 \times 3 \)) for the first GAP layer as in equation (4), where \( h_{\text{new}} \) represents the Gaussian filter weights for the 3D kernel, calculated as equation (1) for \( i, j \) and \( k \) as \( w = 1 \) to 3 or 5. For the second GAP normalization i.e., \( g_2 \) layer in \( g_1g_2b_3b_4 \) architecture, the used Gaussian kernel is obtained via convolution of the discrete Gaussian kernel with each other to get a second order filter response of the second normalization layer, different than the first normalization layer. Please note that we have used a 3D kernel in all GAP layers to accommodate the normalization of feature in the activation map (the \( 3^{rd} /\text{channel dimension} \)) and the filtering kernel moves throughout the whole \( X_f \) vector.
FIGURE 3. Schematic representation of proposed layer, along with input and output histogram for comparison. The input signal is represented as a ramp signal to demonstrate the edge detection process. However, in our experiment input X to the layer is the activated image matrix from the preceding convolution layer. The input passes through the normalization unit to produces a scaled and shifted version of X having a narrow range of feature values. Later, the Gaussian smoothing function transforms the feature vector $X_n$ in a weighted-average fashion to produce a sharpened version of images i.e., $X_g$. The difference of X and $X_g$ produces a masking vector $X_{mask}$, which is again added with the original X to produce Z. The learnable parameters $\alpha$, $\beta$, and $\gamma$ scales $X$, $X_{mask}$ and offset respectively.

In equation (4), the symbol ‘o’ represents the correlation operation which is similar to spatial filtering operation in image processing. Mathematically correlation process is the same as a convolution in the time domain, except that the signal is not reversed before the multiplication process. The idea here is to perform filtering to all images in a replicated manner so that all $32 \times 64$ images are covered. However, the blurring or smoothing effect works differently for each image from $X_{1}^{1}$, $X_{1}^{2}$, ……,$X_{1}^{32}$ on the first batch and so on. The obtained gaussian images can be represented in a matrix as in equation (4), where each $X_{g}^{n}$ represents an image after the Gaussian filter. The applied Gaussian kernel is the same size as the preceding convolution kernel to perform linear correlation. Hence, our initial aim of combining normalization and activation into a single unit seemed not practicable, so we need to use an extra non-linear (activation) function to further support the classification.

$$X_g = \begin{bmatrix} h_{11w} & \cdots & h_{1ww} \\ \vdots & \ddots & \vdots \\ h_{ww1} & \cdots & h_{www} \end{bmatrix} \circ \begin{bmatrix} X_{f1}^{1} & \cdots & X_{f1}^{32} \\ \vdots & \ddots & \vdots \\ X_{f64}^{1} & \cdots & X_{f64}^{32} \end{bmatrix} = \begin{bmatrix} X_{g1}^{1} & \cdots & X_{g1}^{32} \\ \vdots & \ddots & \vdots \\ X_{g64}^{1} & \cdots & X_{g64}^{32} \end{bmatrix}$$ (4)

And the mask is obtained as $X_{mask} = X - X_g$ that subtracts the normalized Gaussian signal from its original version, for each channel output without inheriting the batch properties. Hence finally we forward the output from the layer as:

$$Z = \alpha.X + \beta.X_{mask} + \gamma$$ (5)

Here $\alpha$, $\beta$ and $\gamma$ are learnable parameters each of size [1 1 N], which have unique values for each ‘N’ filters and
help to optimize the output values. \( \alpha \), \( \beta \) and \( \gamma \) are initially selected between 0 and 1 and acts as scaling co-efficient to control the gradient output during backpropagation of weight update. And the second parametric term in equation (5) i.e. \( \beta \) is initially less than 1, however with weights update during backpropagation, the value tends to be \( \beta > 1 \) in which case acts as a high boosting filter, emphasizing the contribution of unsharp masking, and when \( \beta < 1 \) the sharpening mask is more emphasized (please see figure 3). The other learnable parameters \( \alpha \) and \( \gamma \) are also updated during backpropagation, \( \alpha \) works as scaling coefficient for making output equivalent as \( \beta.Xmask \) and \( \gamma \) works as bias with no effect in the layer gradient loss. The MATLAB code implementation along with supporting codes are included in the Appendix.

Stochastic gradient descent (SGD) is the training algorithm for updating the learnable parameter (weights, bias, offset, coefficients) values computed using a mini-batch, instead of using the whole training set error at once as in standard ones [22], [25]. Here the loss is calculated on a mini-batch set by updating the network’s parameters toward a negative gradient of loss at each iteration as in equation (6)

\[
w^{t+1}_i = w^t_i - \gamma^t \cdot \alpha^t \frac{dL}{dw^t_i} + r \left( w^{t-1}_i - w^{t-1}_i \right)
\]  

Here, weights or bias or offset \( w^{t+1}_i \) update in layer ‘l’ at each iteration ‘t + 1’, uses the weights of the previous iteration \( w^t_i \), \( \alpha^t \) is the learning rate hyperparameter for the parameters of layer \( l \), kept at value > 0, and initially 0.001 in our experiments. Since we are also using the SDG with the momentum it reduces the oscillation of the parameter weight update and finds the path of steepest descent towards the optimal value. For this, the hyper-parameter \( r \), known as the rate of momentum is set for 0.95. Also, the negative term represents the gradient of the loss function in the layer updated as in back-propagation after every epoch.

The backpropagation calculates the derivative of loss with respect to (w.r.t) all trainable parameters in CNN. The layer gradient loss w.r.t to input \( X \) i.e., \( \frac{dl}{dz} \) and other parameters \( \alpha \), \( \beta \), and \( \gamma \) are updated as follow:

\[
k = Xmask/X;
\]

From equation (5) we get,

\[
\frac{dl}{dX} = \alpha \frac{dl}{dz} + \beta \frac{k}{dz}
\]  

In our experiment, it was found that ‘\( k \)’ value, when used, produces a small gradient value causing vanishing gradient problem, hence \( k \) was selected to be 1. Hence for the GAP layer with 32 activations and 64 minibatch sizes we define gradient loss as follow:

\[
\frac{dl}{d\alpha} = \sum_{b=1}^{64} \sum_{n=1}^{32} \frac{dL}{d\alpha} X^b_n \frac{dl}{dz} \]

\[
\frac{dl}{d\beta} = \sum_{b=1}^{64} \sum_{n=1}^{32} \frac{dL}{d\beta} Xmask^b_n \frac{dl}{dz}
\]

\[
\frac{dl}{d\gamma} = \sum_{b=1}^{64} \sum_{n=1}^{32} \frac{dl}{dz}
\]
FIGURE 7. Layer-wise filter response visualization and histogram plot for a sample image from the 5-animals dataset during the testing phase. 7.1(a) Histogram of all filter input to layer b1 of b1b2b3b4. 7.1(b) Mean filter response of 32 filters in layer b1. 7.1(c) Histogram of all feature output from layer b1. 7.2(a) Histogram of all filter input to the layer g1 of g1g2b3b4. 7.2(b) Mean filter response of 64 filters in layer g1. 7.2(c) Histogram of all feature output from layer g1. 7.3(a) Histogram of all filter input to layer b2 of b1b2b3b4. 7.3(b) Mean filter response of 64 filters in layer b2. 7.3(c) Histogram of all feature output from layer b2. 7.4(a) Histogram of all filter input to the layer g2 of g1g2b3b4. 7.4(b) Mean filter response of 64 filters in layer g2. 7.4(c) Histogram of all feature output from layer g2. In 7.1 (c) and 7.3 (c) the output of the batch normalization layer histogram shows the feature distribution is completely different from its input i.e., 7.1 (a) and 7.3 (a) respectively, the frequency of weight value was more in the mean range in input whereas in output, the weight around mean is reduced. This drastic change in distribution is also shown in the mean response plot, 7.1 (b) and 7.3 (b) with correlation coefficients 90.46 and 84.73, respectively. However, the output pixel/weights distribution is not completely changed while using GAP normalization. The output histogram using GAP layers, 7.2 (c) and 7.4 (c) follows the input pattern of 7.2 (a) and 7.4 (a) respectively. Also, the mean plot response shows a very high correlation with its input filter mean, and the correlation value is 94.6 and 91.3, respectively. Please see the appendix for code implementation.

Training in minibatch has a significant effect in achieving convergence time, i.e., higher minibatch size makes the training quicker by reducing the number of iterations per epoch, on the other hand also affects the training accuracy. We are also training the network in mini-batch i.e., the entropy loss is calculated based on mini-batch input however,
FIGURE 8. Correlation value plot between input and output in the normalization layer for all test images in the 5-animals dataset. Here in Layer 2, the BN (b1) layer produces a correlation value of around 90% for all test sets, whereas the GAP layer (g1) has a slightly higher correlation value than b1. Whereas in the second normalization layer i.e., Layer 6, the BN layer (b2) produces drastically low correlated output with its input, and in wide ranges for all test sets, i.e., ranging from 94% to as low as 22%, however, the output from g2 is not highly decorrelated with its input, hence in the range of around 90% correlation with its input. Input X is the output from the preceding convolution layer, and output Z is the output from the normalization layer. If the layer correlation value comes out to be very low, it means the layer has decorrelated the feature. However perfect correlation is also useless.

the normalization process is not the batch normalization if the used normalization layer is GAP. In normalization generally, the whitening process is preferred where the input is linearly transformed to zero mean and unit variance i.e., the input will have a fixed distribution, which is also considered to remove the ill effects of the internal covariate shift. While activation process is like a filtering process where the activation function determines the weight output. Similarly in our method, the Gaussian filter works as an activation function along with a learned parameter to create a normalized mask. This mask works as the additional extracted feature ($\beta X_{mask}$) with the original input so that the original signal is slightly boosted with its filter mean responses. So, later when linearly added to the original input adds the value according to the mask. If the masked value is only used, we would lose the entire input image property. Hence mask is added to the original input to bring a calculated variance, without losing the linear property of its input.

Few hyperparameters like initial learn rate, learn-drop rate, etc. affect the training time and learning ratio however in the longer run, results are not significantly different, the lately achieved convergence slightly affects the testing result only. Fully connected layers act as a single-layered feed-forward network with all parameters connected from input to output. Because of this nature, FCL is blamed for causing overfitting in the network, and potentials regularization techniques like dropout are used in between them [26], [30]. So, to reduce the number of parameters and eventually overfitting, we have reduced the number of FCL from 4 to 3 in our base architecture [12].

B. CLASSIFICATION PERFORMANCE AND DISCUSSION
We performed several classification experiments using the base architecture with different normalization layers used in the first 2 encoder part. Our main goal was to compare the classification result, with the first two BN layers replaced with GAP layers. Benchmark 2D datasets were used for this purpose. The used datasets are CIFAR-10 [10], Caltech-102 [27], 5-animals, and MRI images from OASIS [28] for 2D classification purposes. Among these, 5-animals and MRI images are prepared privately and are made available in a public repository, whereas others are already publicly available. The 5-animals dataset consists of around 700 images per class of five animals viz, tiger, lion, dog, cat, and fox. OASIS MRI consists of a total of 5220 images of four classes categorized based on clinical
TABLE 2. Detailed experiment results using different normalization techniques in the same base architecture as shown in Table 1. The bold number signifies the best performance result. The training environment was identical in the case for all datasets in natural images and similarly for medical MRI. The AlexNet model was obtained using the MATLAB deep learning tool. In the case of the same dataset, the training, validation, and testing materials were also identical, so the result could not be biased in any case. Accuracy represents the % of correctly classified samples during prediction, whereas average test recall and precision are calculated by taking the mean of class-wise recall and precision. 95% CI error represents the error with a 95% confidence score, the one with a score above 95% is only calculated for a min-error value and one with a score below 95% is only calculated for max-error value.

| Natural Dataset | STRUCTURE | # OF TRAINING IMAGE | # OF VALIDATION IMAGE | # OF TESTING IMAGE | TRAINING TIME (MIN) | VALUATION TIME (MIN) | TESTING TIME (MIN) | AVERAGE TEST RECALL | AVERAGE TEST PRECISION | 95% CI ERROR (MIN) | 95% CI ERROR (MAX) |
|-----------------|-----------|---------------------|----------------------|-------------------|---------------------|----------------------|---------------------|----------------------|----------------------|--------------------|--------------------|
| Caltech-102     | g1b2b3s4  | 100                 | 5481                 | 1824              | 1829                | 113                  | 100                 | 69.6                 | 66.59                | 0.5714              | 0.5078             | 31.24              | 35.57              |
|                  | g1g2b3s4  | 91                  | 100                  | 69.1               | 66.54               | 0.5642              | 0.5178              | 31.33                | 35.62                |                    |                    |
| Weightless AlexNet (WA) |              |                     |                      |                   |                     |                     |                     |                      |                     |                    |                    |
|                  | g1b2b3s4  | 84                  | 100                  | 60.36              | 59.69               | 0.4317              | 0.426               | 38.16                | 42.65                |                    |                    |
| 5-animals        | g1g2b3s4  | 120                 | 100                  | 59.1               | 58.16               | 0.6037              | 0.5801              | 37.99                | 45.69                |                    |                    |
|                  | b1b2b3s4  | 30                  | 100                  | 57.1               | 58.48               | 0.5941              | 0.5899              | 37.68                | 45.37                |                    |                    |
|                  | WA        | 24                  | 90                   | 48.09              | 42.47               | 0.4151              | 0.421               | 53.67                | 61.38                |                    |                    |
| CIFAR-10         | g1g2b3s4  | 851                 | 100                  | 74.55              | 74.68               | 0.7465              | 0.7461              | 24.54                | 26.24                |                    |                    |
|                  | b1b2b3s4  | 2300                | 100                  | 75.3               | 75.21               | 0.7502              | 0.7521              | 23.94                | 25.64                |                    |                    |
|                  | WA        | 390                 | 100                  | 74.68              | 75.11               | 0.7565              | 0.7513              | 24.02                | 25.72                |                    |                    |
|                  | 500       | 1044                | 1566                 |                   |                     |                     |                     |                      |                     |                    |                    |
| OASIS MRI CDR (2D MRI) | g1g2b2s4  | 76                  | 100                  | 99.42              | 99.74               | 0.9974              | 0.9982              | 0.01                 | 0.51                 |                    |                    |
|                  | g1g2b3s4  | 180                 | 100                  | 99.67              | 99.66               | 0.9966              | 0.9967              | 0.08                 | 0.69                 |                    |                    |
|                  | b1b2b3s4  | 41                  | 100                  | 99.71              | 99.81               | 0.9976              | 0.9987              | 0.01                 | 0.51                 |                    |                    |
|                  | WA        | 13                  | 100                  | 98.66              | 99.75               | 0.9951              | 0.995              | 0.2                  | 0.95                 |                    |                    |
| ADNI MRI small (3D CNN) | g1b2b3s4  | 196                 | 100                  | 71.65              | 80.1                | 0.8247              | 0.7895              | 9.43                 | 30.57                |                    |                    |
|                  | g1g2b3s4  | 551                 | 100                  | 73.36              | 81.82               | 0.8197              | 0.8092              | 7.99                 | 27.38                |                    |                    |
|                  | b1b2b3s4  | 81                  | 100                  | 73.68              | 83.64               | 0.8349              | 0.8288              | 6.59                 | 26.14                |                    |                    |
| ADNI MRI baseline (3D CNN) | g1g2b3s4  | 670                 | 100                  | 94.92              | 92.23               | 0.9438              | 0.906               | 4.72                 | 10.82                |                    |                    |
|                  | g1g2b3s4  | 1780                | 100                  | 94.42              | 93.58               | 0.9346              | 0.9256              | 3.63                 | 9.21                 |                    |                    |
|                  | b1b2b3s4  | 244                 | 100                  | 94.92              | 91.89               | 0.9134              | 0.9079              | 5.11                 | 11.22                |                    |                    |

dementia rating (CDR) level of participants. Details of participants are described in our previous work on 2D CNN [29].

To test 3D CNN performance, we used 3D-MRI volumes obtained from the ADNI database (http://adni.loni.usc.edu/).

To compare the result on bulky and small training samples we have prepared 2 types of datasets for 3D CNN, MRI_baseline is the bulkier one with 988 MRI samples and MRI_small with 187 MRI samples. These MRIs belong to one of the 3 classes viz AD (Alzheimer’s disease), MCI (Mild cognitive impairment), and NC (Normal controls). Details on these medical databases can be found in our previous work [12].

Table 2 shows the detailed condition for the experiment and the obtained results. Overall, it shows the use of the proposed layer for normalization gives results almost similar to that of batch normalization, and in few cases, the result is slightly better with higher training time. The result of the experiment is better in classification with double GAP layer as normalization layer in a bulkier dataset like MRI_baseline with overall test accuracy 93.58% against 91.89% using 100% batch normalization. Similarly, CIFAR-10 test accuracy improved from 75.11% to 75.21% by replacing the GAP normalization layers with the BN layer in the first and second encoder.

While the result is better with the replacement of the first BN layer with GAP in a smaller dataset like the 5-animals dataset (2D CNN accuracy: 62.92% vs. 58.48%), which supports the use of the proposed method for normalization and activation.

C. FEATURE VISUALIZATION AND ANALYSIS

D. CORRELATION AND GENERALIZATION

Correlation is the measurement of the similarity between two signals. The correlation coefficient of two random variables measures the linear dependency between the input feature matrix \((X)\) and output \((Z)\) with \(N\) scalar observations as the Pearson correlation coefficient \(r\):

\[
r(X, Y) = \frac{1}{N-1} \sum_{i=1}^{N} \left( \frac{X_i - \mu_X}{\sigma_X} \right) \left( \frac{Y_i - \mu_Y}{\sigma_Y} \right)
\]

\[
r(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} \tag{11}
\]
The correlation coefficient also measures the covariance ‘cov’ between any two vector matrices. As BN is stated to reduce the internal covariant shift in its layer, we have measured the covariance between output and input in this layer using the Pearson correlation coefficient (r). Here in our case, X is the 3D feature matrix generated from the preceding convolution layer and Z is the output from the normalization layer with same dimensions as X. And the value of ‘r’ is a single value for all filters for a single image, hence correlating average characteristics of the activation layer. (Please note for the filter-wise response we have plotted the mean response plot as in figure 7.1 (b) to 7.4(b)). Perfect correlation brings an identical result between input and output without any variance in which case the layer becomes useless, however very low correlation is also dangerous, as it brings very high variability and shift between layer input and output.
FIGURE 10. Comparison of feature detection heatmap using various visualization algorithms for natural images. The used techniques to generate heat maps on test images in successive order are LIME [33], Occlusion [34], and Grad-Cam [35]. Overall, AlexNet [3] has a narrow heat map area i.e., the region of influence for classification, and similarly, the heat map area of g1g2b3b4 and g1b2b3b4 is wider and accurate than one using BN only i.e., b1b2b3b4. It signifies the better feature detection process done using GAP normalization.

which makes the layer suspicious to accidents like vanishing gradients. So, it is still unclear if a high correlation is good or not, which in our case, we expected a higher correlation value than the BN result, which turned out to be true.

Overfitting is one of the major adversity in ML that brings disparities in test performance and training performance within a trained network. This can be verified when the error on the training set is very less, but when an unseen similar data is predicted via the network the error is large. It means the network has ‘memorized well’ but has ‘not learned well’. It is majorly the number of parameters that decide the fate of the network to overfitting. Keeping in mind that if the number of parameters in the network is much smaller than the total number of points in the training set, then there is little or no chance of overfitting, which simply means increasing the parameter of the training network increases the chance of overfitting. To check the generalization error, we computed the range of prediction error on ‘N’ test samples for a confidence score over 95% against the standard test error (STE) = 1 – accuracy. This test of error margin is also called the Wald test and represents the minimum and maximum error in the range of 95% confidence interval as shown in Table 2 and plotted in Figure 6 along with the accuracy graph. Besides, we plotted the T-SNE projection for test images and found out the errored distribution as shown in figure 9.

V. CONCLUSION

To conclude we have experimented with our proposed idea of the GAP layer as a normalization layer in 2D and 3D architecture CNN. Our experiments show the use of GAP layers produces a result similar or slightly better in the case of the bulkier 3D dataset and lighter 2D dataset. We studied the phenomenon of overfitting via training and validation graph, normalization layer covariance via mean response plot and
correlation plot, and feature representation via TSNE and histogram plots. To summarize we have listed the finding as below:

1. With b1b2b3b4 architecture, the training accuracy shoots higher quickly, indicating an overfitting condition as the validation accuracy during the mid-training was still too low causing a higher gap between training and validation accuracy. Hence, with the GAP layer, we delayed the faster convergence of weights during training (please see Figure 2).

2. The weights of convolutional filters in early layers seems to change abruptly from convolution to normalization process during BN, (please see Figure 4) this might suggest the feature property of the input image becomes highly uncorrelated with its input after normalization. Consequently, the correlation coefficient between input and output of BN is quite low. However, using GAP the filter weights are only slightly changed and the input-output correlation value is higher (Please see Figure 7 and 8), indicating less distortion of image property.

3. Due to the minibatch mean value used for scaling in BN, the output from normalization layers is scaled with the minibatch mean, i.e., the minibatch properties of images are mixed, causing a higher squeezing in its feature value (Please see histogram plot in Figure 7). This might have brought higher variability in BN output, as discussed in point 2. On the other hand, in GAP normalization the scaling mean coefficient is calculated from the activated channels from the same image i.e., equivalent to minibatch = 1, so the image property from the same image is only mixed up, without spoiling its feature attributes.

4. Scaling mean and variance are empirically calculated as in 3. for each input image, so no need to pass the trained mean and variance value as in BN during the testing phase. Also, the BN’s error is higher for small batch size, due to imprecise batch statistics estimation.

5. The activation region for decision-making is wider and accurate in most cases using g1g2b3b4 or g1b2b3b4 than the BN-based b1b2b3b4 and AlexNet (please see Figure 10).

6. In most of the experiment, the result is slightly better (Table 2) and can also visualize via T-SNE projection (see Figure 9).

Our proposed layer is itself not a replacement for batch normalization (BN), since it is beneficial only if we use those layers in the first one or two convolution layers. Thus, it works as a good alternative for batch normalization in the early layers although not the ultimate layers. Moreover, another serious drawback in our idea is the use of the filter function itself, as it consumes a lot of time to perform a 3D filtering operation, because of this our method requires 2 (1 GAP) to 5 (2 GAPS) times more training duration to generate trained model. More the layers and training sample used; more will be the delay in training. To overcome this, if we can perform a weight-wise operation like convolution or BN in the layer itself without using an external filter function, might reduce the operation time. To do so, we need a more precise algorithm inside the network with parameters for Gaussian filter operation. This will be our future work. We hope our work will help the researcher working in the field of DNN to achieve better results in their application task.

APPENDIX

MATLAB implementation code is presented as C1, C2, C3. Confusion matrix of performance (as of Table 2) is also presented. The prepared dataset can be downloaded from https://drive.google.com/drive/folders/1G1fsK2VxaHkvttJf

C1: MATLAB code implementation for GAP layer used after convolutional layer as an alternative to Batch normalization layer.

```matlab
encoder_1 = [convolution2dLayer([3 3], 32, 'Padding', 'same', 'WeightsInitializer', 'glorot');
test_gauss(32, 'g1', 3); % batchNormalizationLayer for BN
leakyReluLayer('Name', 'relu_s1');
maxPooling2dLayer(poolSize, 'Stride', 2);
]
The definition for test_gauss is as below:
classdef test_gauss < nnet.layer.Layer
    properties(Learnable)
    Alpha
    Beta
    Gamma
    end
    properties
    hsize
    end
    methods
    function layer = test_gaus(numChannels,name,hsize)
        layer.Name = name;
        layer.hsize = hsize;
        layer.Description = "Scale factor" + numChannels + "channels";
        layer.Alpha = rand([1 1 numChannels]);
        layer.Beta = rand([1 1 numChannels]);
        layer.Gamma = rand([1 1 numChannels]);
    end
    function Z = predict(layer, X)
        X_mean = double(mean(X,3));
        X_std = std(X,0,3); % For 3D 4th dimension is used for mean and std
        X_F = (X - X_mean) / X_std;
        h_size = layer.hsize;
        h = fspecial3('gaussian',h_size);
        h_f = [h]; % h_f = convn(h,h,'same') for 2nd gap layer
        X_test = imfilter(X_F, h_f, 'replicate');
        Z = layer.Alpha.*X + layer.Beta.* (X - X_test) + layer.Gamma;
    end
end
```

B. Khagi, G. R. Kwon: CNN-Based Natural Image and MRI Classification
FIGURE 11. Confusion matrix for dataset (architecture).
FIGURE 11. (Continued.) Confusion matrix for dataset (architecture).

function [dLdX, dLdAlpha,dLdBeta,dLdGamma] = backward(layer, X, ∼, dLdZ, ∼)
  h_size = layer.hsize;
  h = fspecial3('gaussian',h_size);
  X_mean = double(mean(X,3)); %X(w1,w2, fiternumber, batchsize)
  X_std = std(X,0,3);
  X_F = (X-X_mean)./X_std;
  h_f = [h];
  X_test = imfilter(X_F,h_f,'replicate');
  dLdX = (layer.Alpha.*dLdZ + layer.Beta.*dLdZ);
  dLdAlpha = X.* dLdZ;
  dLdBeta = sum(dLdBeta,[1 2 3]);
  dLdAlpha = sum(dLdAlpha,4); % for 3D
  dLdBeta = sum(dLdBeta,5);
  dLdBeta = sum(dLdBeta,[1 2]);
  dLdBeta = sum(dLdBeta,4);
  dLdGamma = dLdZ;
  dLdGamma = sum(dLdGamma,4);
  dLdGamma = sum(dLdGamma,4);
end
end
end

C2: Code snippet for feature visualization and mean response as in Figure 7 and Figure 8.
To visualize the feature and get mean response of the proposed layer as in Figure 7.1(b) to 7.4(b), where X and Z represent the input and output to the predict function, respectively.

... Load net, X, Z % Need to save net, X and Z layer as .mat file previously.
feature = gather(X);
x(:, :) = gather(mean(X,[1 2]));
x = x';
z(:, :) = gather(mean(Z,[1 2]));
z = z';
plot(x)
hold on
plot(z)
hold off
Dsize = size(X,4); % for 3D CNN size(X,5)
for i = 1:Dsize
A = gather(X(:, :, :, :, i)); % for 3D CNN 'i' in 5th dimension i.e minibatch representing test samples
B = gather(Z(:, :, :, :, i));
R(:, :, :, :, i) = corrcoef(A, B);
plot(R)
end
...
C3: Code snippet implementation for generating feature map using Deep dream as shown in Figure 5.
...
load net; % Load Pretrained Network layer = 22; % Last FC layer of the network trained_net.Layers(end).ClassNames(channels)
% Generate detailed images that strongly activate these classes.
I = deepDreamImage(trained_net.layer, channels, ...
    'Verbose', false, ... 
    'NumIterations', 50);
figure
montage(I)
name = net.Layers(layer).Name;
title([{'Layer', name, 'Features'}])

See Figure 11.

REFERENCES

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. “Backpropagation applied to handwritten zip code recognition,” Neural Comput., vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.

[2] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. “Handwritten digit recognition with a back-propagation network,” in Proc. Adv. Neural Inf. Process. Syst., 1990, pp. 396–404.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 25, 2012, pp. 1097–1105.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2015, pp. 1–9.

[5] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 770–778.

[6] R. Girshick, “Fast R-CNN,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 1440–1448, doi: 10.1109/ICCV.2015.169.

[7] S. Ren, K. He, R. Girshick, and J. Sun. “Faster R-CNN: Towards real-time object detection with region proposal networks,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, Jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[8] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A deep convolutional encoder-decoder architecture for image segmentation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 12, pp. 2481–2495, Dec. 2017, doi: 10.1109/TPAMI.2016.2644615.

[9] A. Kendall, V. Badrinarayanan, and R. Cipolla. “Bayesian SegNet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding,” in Proc. Brit. Mach. Vis. Conf. (BMVC), 2017, pp. 57.1–57.12, doi: 10.5244/c.31.57.

[10] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Tech. Rep., 2009, pp. 1–60.

[11] A. Mordvintsev, C. Olah, and M. Tyka. (2015). Inceptionism: Going Deeper Into Neural Networks. Google AI Blog. Accessed: Feb. 10, 2021. [Online]. Available: https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

[12] B. Khagi and G.-R. Kwon, “3D CNN design for the classification of Alzheimer’s disease using brain MRI and PET,” IEEE Access, vol. 8, pp. 217830–217847, 2020, doi: 10.1109/ACCESS.2020.3040486.

[13] C. Gulcehre and Y. Bengio, “Knowledge matters: Importance of prior information for optimization,” J. Mach. Learn. Res., vol. 17, pp. 1–32, Jan. 2016.

[14] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in Proc. 32nd Int. Conf. Mach. Learn. (ICML), vol. 1, Jul. 2015, pp. 448–456.

[15] S. Wiesler and H. Ney, “A convergence analysis of log-linear training,” in Proc. Adv. Neural Inf. Process. Syst., vol. 24, 2011, pp. 657–665.

[16] K. He, X. Zhang, S. Ren, and J. Sun. “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 1026–1034, doi: 10.1109/ICCV.2015.123.

[17] J. M. Prewitt, “Object enhancement and extraction,” Picture Process. Psychopictorics, vol. 10, no. 1, pp. 15–19, 1970.

[18] R. M. Lewitt and S. Matej, “Overview of methods for image reconstruction from projections in emission computed tomography,” in Proc. IEEE, vol. 91, no. 10, pp. 1588–1611, Oct. 2003, doi: 10.1109/JPROC.2003.8179882.

[19] C. Rafael, Gonzalez, Richard E. Woods, Digital Image Processing, Upper Saddle River, NJ, USA: Prentice-Hall, 2002.

[20] I. Sobel, “History and definition of the Sobel operator,” in Proc. Retrieved World Web, vol. 1505, 2014.

[21] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning (still) requires rethinking generalization,” Commun. ACM, vol. 64, no. 3, pp. 107–115, Mar. 2021, doi: 10.1145/3446776.

[22] H. Robbins and S. Monro, “A stochastic approximation method,” Ann. Math. Statist., vol. 22, pp. 400–407, 1951.

[23] T. Gaussian, “The Gaussian kernel,” in Front-End Vision and Multi-Scale Image Analysis, 2003, pp. 37–51, doi: 10.1007/978-1-4020-8840-7_3.

[24] T. Lindeberg, “Scale-space for discrete signals,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 12, no. 3, pp. 234–254, Mar. 1990.

[25] K. P. Murphy, Machine Learning: A Probabilistic Perspective. Cambridge, MA, USA: MIT Press, 2012.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” J. Mach. Learn. Res., vol. 15, no. 1, pp. 1929–1958, 2014.

[27] L. Fei-Fei, R. Fergus, and P. Perona, “Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories,” in Proc. Conf. Comput. Vis. Pattern Recognit. Workshop, Jun. 2004, p. 178.

[28] D. S. Marcus, T. H. Wang, J. Parker, J. G. Csernansky, J. C. Morris, and R. L. Buckner, “Open access series of imaging studies (OASIS): Cross-sectional MRI data in young, middle aged, nondemented, and demented older adults,” J. Cognit. Neurosci., vol. 19, no. 9, pp. 1498–1507, Sep. 2007.
B. Khagi, G. R. Kwon, and R. Lama, “Comparative analysis of Alzheimer’s disease classification by CDR level using CNN, feature selection, and machine-learning techniques,” Int. J. Imag. Syst. Technol., vol. 29, no. 3, pp. 297–310, Sep. 2019, doi: 10.1002/ima.22316.

J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, “Striving for simplicity: The all convolutional net,” in Proc. 3rd Int. Conf. Learn. Represent. (ICLR), 2015, pp. 1–14.

R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Region-based convolutional networks for accurate object detection and segmentation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 38, no. 1, pp. 142–158, Jan. 2016, doi: 10.1109/TPAMI.2015.2437384.

J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, “Understanding neural networks through deep visualization,” 2015, arXiv:1506.06579. [Online]. Available: http://arxiv.org/abs/1506.06579.

M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should I trust you?” explaining the predictions of any classifier,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2016, pp. 1135–1144.

M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in Proc. Eur. Conf. Comput. Vis. (Lecture Notes in Computer Science Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8689. Cham, Switzerland: Springer, pp. 818–833, 2014, doi: 10.1007/978-3-319-10590-1_53.

R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual explanations from deep networks Via gradient-based localization,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 618–626.

A. Shrestha and A. Mahmood, “Review of deep learning algorithms and architectures,” IEEE Access, vol. 7, pp. 53040–53065, 2019.

L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaria, M. A. Fadhel, M. Al-Amide, and L. Farhan, “Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions,” J. Big Data, vol. 8, no. 1, pp. 1–74, Dec. 2021.

S. Singh and S. Krishnan, “Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 11234–11243, doi: 10.1109/CVPR42600.2020.01125.

Y. Wu and K. He, “Group normalization,” Int. J. Comput. Vis., vol. 128, no. 3, pp. 742–755, Mar. 2020, doi: 10.1007/s11263-019-01198-w.

J. Lei Ba, J. Ryan Kiros, and G. E. Hinton, “Layer normalization,” 2016, arXiv:1607.06450. [Online]. Available: http://arxiv.org/abs/1607.06450.

Z. Dai and R. Heckel, “Channel normalization in convolutional neural networks avoids vanishing gradients,” 2019, arXiv:1907.09539. [Online]. Available: https://arxiv.org/abs/1907.09539.

D. Ulyanov, V. Vedaldi, and V. Lempitsky, “Instance normalization: The missing ingredient for fast stylization,” 2016, arXiv:1607.08022. [Online]. Available: http://arxiv.org/abs/1607.08022.