International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Efficiency-Enhanced Densenet Architectures: An Exploration of Multi-Kernel, Multi-Branch Structures for Achieving Optimal Trade-Off Between Parameters and Accuracy

Shaikh Abdus Samad Shaikh Aga Mohammad[1], Gitanjali J.*[2]

Submitted: 25/01/2024 **Revised**: 03/03/2024 **Accepted**: 11/03/2024

Abstract: In this work, we present intricately woven investigative one-layer structure designs for the Dense-Block of the DenseNet, intending to improve performance in visual recognition tasks. Developing a robust representation for accurate visual recognition is a critical challenge that requires more than just increasing the depth and width of neural networks. Therefore, we have devoted significant effort to developing new One-Layer Structures (OLSs) for the Dense-Block. Our proposed OLSs are comprising multiple branches of stacks of 1×1, 3×3, and 5×5 convolutional layers. We recommend replacing the standard OLS of Dense-Block with one of these proposed OLSs. Our proposed OLSs are lightweight, simple, and optimally arranged, making them an ideal choice for optimizing network performance. We organized them into three families: 1.0 and 1.1, 2.0 to 2.3, and 3.0 to 3.3. To evaluate the effectiveness of our proposed models, we conduct experiments on the three benchmark datasets: Imagenette, CIFAR-10, and CIFAR-100. The investigation of DenseNet models enhanced with OLSs up to version 3.3 provides a nuanced understanding of the intricate relationship between model complexity, computational efficiency, and accuracy. Through meticulous analysis on multiple datasets, a consistent pattern of parameter and FLOP reduction is observed, indicating progressive refinement in model architecture. OLS 2.X versions achieve accuracy values ranging from 94.84% to 95.31% on CIFAR-10, 80.33% to 80.69% on CIFAR-100, and 93.325% to 93.478% on Imagenette demonstrating that the integration of OLSs contributes positively to model performance.

Keywords: *One-Layer Structure, DenseNet, Dense-Block, Optimized Network, Convolutional Neural Network*

1. Introduction

Convolutional Neural Networks (CNNs) have recently achieved impressive breakthroughs in a diverse range of computer vision tasks and have excelled beyond the numerous conventional feature-driven approaches in this realm. Its remarkable capability of extracting underlying nonlinear structures of image data is the basis upon which deep CNN is successful in image/video analytics. For increasing the performance of CNN models, the two most widely used steps are increasing the depth and width of the network (in terms of the number of layers and feature maps at each layer respectively) and using more training data if feasible. In the meantime, efforts have been made to minimize the number of floating-point operations (FLOPs) and the number of model parameters. Especially when the model has many layers, designing architectures becomes more intricate as more hyperparameters are added, such as kernel size or filter size, activation function, stride, learning rate, etc.

The VGG [1] network, characterized by its simplistic yet effective architecture consisting of stacked convolutional layers, ReLU [2] activation functions, and a limited number of pooling layers, has proven to be a seminal CNN model in the field of image recognition. In recent years, there has been a trend toward increasing the depth of CNNs. However, this increase in depth also leads to optimization and learning challenges and does not always result in better performance. Therefore, research has shifted towards the development of well-architected models such as Inception [3][4][5], ResNet [6], and DenseNet [7], which have been able to achieve high accuracy with fewer parameters. Empirical evaluations on a wide range of visual recognition tasks [6][8][9][10][11][12][13] have demonstrated the robustness of VGG networks, DenseNets, and ResNets.

It was noted that better performance of a model in image classification tasks would likely be transferred into quality improvements in a diverse spectrum of other vision applications. So, incorporating architectural improvements in deep CNN architectures helps to elevate the quality of other computer vision tasks that are strongly reliant upon high-quality, learnable features. In VGGNet, layers are stacked on top of each other, which results in a volumetric number of parameters. If convolutional layers are simply stacked on top of one another, the gradients of the layers start losing their intensity. ResNets and DenseNets are designed to assist in solving the vanishing gradient problem with novel skip connection strategies. Through the development of carefully designed topologies, the Inception model family has demonstrated its ability to deliver high

[1] *Vellore Institute of Technology, Vellore – 632014, Tamil Nadu, INDIA*
ORCID ID : 0009-0001-8908-9683
[2] *Vellore Institute of Technology, Vellore – 632014, Tamil Nadu, INDIA*
ORCID ID : 0000-0002-2296-800X
** Corresponding Author Email: gitanjalij@vit.ac.in*

accuracy at low theoretical complexity. It has been common practice in Inception models to split layers into multiple branches, transform them, and then merge them. Inception modules make use of 1×1 convolution to reduce the number of channels of input volume of split branches. The output of lower-dimensional embedding is then transformed using different size filters, such as 3×3 and 5×5. It then merges the outputs of the branches by concatenating them. Modules for Inception will be able to approximate dense and large layers with the same degree of representational power at a lower computational complexity by splitting, transforming, and merging.

In this work, we present the ten meticulously woven alternate one-layer structures that inherit the properties of the Inception model into DenseNet which utilizes the easy-to-implement split-transform-merge approach of Inception. Our approach is simple: we distribute input to multiple branches and aggregate the features which are transformed with different and similar topologies. A large number of transformations can be carried out with this design without requiring any specialized layers. In the proposed models, a simplified version of Inception like modules is used in a Dense-Block where it does not contain a max pool layer and a direct layer with the convolution of 1×1 filters. We make use of 3×3 and 5×5 filters in the proposed models. In some of the proposed models, we employ two consecutive 3×3 filters to replace one 5×5 filter. The same receptive field can be covered by two consecutive 3×3 filters as by one 5×5 filter. The section 3 discusses the ten proposed instances of split-transform-merge approach modules in Dense-Block that are used as the replacement structure strategies for One-Layer. Figure Fig. 1 shows the standard OLS of a Dense-Block. Although there are a few methods that increase accuracy while maintaining (or reducing) complexity while going deeper and wider, it is far easier to increase accuracy by enhancing capacity, since capacity can increase accuracy simultaneously. The methodology adopted in our study reveals that the distribution of input across two or four branches within a Dense-Block is a significant and concrete dimension, in addition to the established dimensions of length and width. Empirical investigations conducted demonstrate that, in terms of accuracy, bifurcating a layer within a Dense-Block into two or four branches, as opposed to mere depth augmentation, leads to superior accuracy outcomes.

In summary, we have made the following contributions:

We propose an intricately constructed split-transform-merge strategy for One-Layer of the Dense-Block of DenseNet, which leads to improved performance of the network. Specifically, we present ten exquisitely knit methodological OLS designs that are divided into three families.

We adopt a multi-branch topology in the development of our architectures, which includes multi-scale convolutions with kernel sizes of 1×1, 3×3, and 5×5. This approach allows for a more extensive feature space, as it includes different receptive fields and multiple paths of varying complexity.

To evaluate the effectiveness of our proposed architectures, we conducted extensive experiments on various benchmark datasets. The results of our experiments confirm that DenseNet with some of our proposed OLSs outperforms the standard DenseNet in terms of accuracy with fewer parameters.

Our study demonstrates that our OLS with a cardinality (number of simultaneously occurring transformations) of two and four for Dense-Block is superior to the standard DenseNet despite having fewer parameters and reduced complexity. OLS 2.0 and 2.1, for instance, are designed to keep the number of parameters and FLOPs complexity low. Although capacity (depth or width of the model) can raise accuracy quite easily, it is rather difficult to achieve the desired result while simultaneously reducing complexity (or keeping it the same).

We demonstrate that more use of 1×1 convolution can provide a means to improved accuracy. For example, OLSs from 3.0 to 3.3 show how to utilize more use of 1×1 convolution in the model.

Our neural networks outperform several other CNN models [6][7][14][15]. Proposed OLSs of the second family from 2.0 to 2.3 are exceptionally effective in achieving comparable accuracy with fewer parameters. Third-family models achieve better accuracy but with few extra parameters.

2. Related Work

Improved network architecture designs are the most effective means of improving deep network processing efficiency. By designing network architecture, one aims to describe backbone structures that can be used to improve performance between a wide range of tasks. The measure of the efficiency of architecture is the accuracy of classification on Imagenette and CIFAR datasets, in addition to other criteria.

Lightweight CNN architectures with efficient building units are introduced to improve deep network processing efficiency. In MobileNets and recent variations [16][17], the design of network architecture is based on linear bottlenecks and inverted residuals. In MobileNeXt [18], Sandglass blocks are created by reversing their inverted residuals. It is the cheap operation that is used in GhostNet [19] to create novel feature maps. Moreover, ShuffleNet [20][21] and CondenseNet [22] both employ shuffle layers along with Learned Group Convolutions (LGC) for generating new feature maps, respectively. Acknowledging the potential trade-off in accuracy, it is worth noting that these models

are designed to be parameter-efficient so that they can be deployed on hardware with limited resources. Their focus on efficiency may result in some compromise in accuracy.

In order to make CNNs as accurate as possible, spatial attention and channel attention are usually incorporated into the network. A feature map for SK-Net [23] is built in such a way that it incorporates a feature-based attention mechanism across parallel branches of the network. Applications that have been successful in the past that make use of attention modules include image classification [24][25], image captioning [26], face verification [27], semantic segmentation [28][29], and video frame interpolation [30]. Models with attention modules result in additional parameters.

Various modules can be applied to existing architecture networks to enhance their performance, such as normalization modules [31], attention modules [32], group convolution [33], and skip connection strategy [6]. Normalization can be done in several different ways, and batch normalization [31] is the most common. In most deep learning models, this is the default method used to improve performance as well as speed up convergence time. Moreover, random sampling contributes to improved generalization capability in batch normalization. It is important to note that while these techniques have shown promise in improving performance, achieving optimal performance in machine learning is an ongoing endeavour.

Many studies have been conducted on deep neural networks aimed at reducing the computational cost of these networks. Researchers have focused on pruning non-essential connections in neural networks in order to reduce computational costs [34][35][36] since these networks contain a significant amount of redundant information. Quantization [37][38], on the other hand, focuses on reducing the memory usage and bit-width of floating-point weights. These techniques can be applied post-training of a model and it can be time and resource consuming.

Researchers have begun to look at systematically searched neural networks instead of manually designed ones, due to increased processing power. By utilizing neural architecture search (NAS) convolutional neural networks [39][40][41][42][43] are capable of providing improved performance, however, they consume an enormous amount of computing resources. Manually designing [44] the CNN network space would require a great deal of manpower.

A deep neural network (DNN) can be re-parameterized in a way that has no residual connections in the network. Re-parameterization [45][46] refers to the process of replacing the parameters of a structure with those of another set of parameters. It is the case that the residual connections are to be removed at inference based on a re- parameterization process after the network with residual connections is trained. RepVGG [47] is a CNN model that resembles a multi-branch network, which can be transformed into a VGG-like model. Structural re-parametrization can be used to do so with stacks of 3×3 convolutions successively with ReLU. At inference, it produces the same results. The simplicity of RepVGG, while advantageous in many contexts, may be suboptimal for highly challenging tasks, such as training exceptionally deep neural networks.

3. Proposed One-Layer Structures

The present research study employs a network design pattern that bears close resemblance to DenseNet, using aesthetically similar configured components. To build the model, we employed multiple Dense-Blocks in conjunction with a transition layer. Our objective was to create a model that is more accurate, with fewer parameters, by presenting precisely intertwined designs of layers for one One-Layer of a Dense-Block. Our work presents sophisticated design approaches for the one-layer structure of a Dense-Block concerning two key factors: 1) To enhance the one-layer structure's exposure to a dual receptive field, our approach incorporates two kernel sizes - 3×3 and 5×5. The former maintains familiarity with the standard receptive field, while the latter provides access to an expanded receptive field. This duality in receptive field sizes enables the model to capture both fine-grained and larger contextual information, thereby enhancing its ability to comprehend complex data. 2) We propose parallelizing the convolution operations within the one-layer Dense-Block by using 3×3 and 5×5 kernel sizes. This strategic approach coordinates these operations in two and four parallel branches, allowing the model to benefit from both 3×3 and 5×5 convolutions simultaneously. This fosters a more nuanced and comprehensive feature extraction process, resulting in more effective and efficient operations.

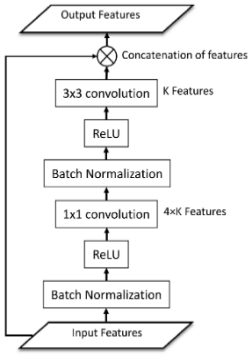

Fig 1. Standard OLS of Dense-Block. 'K' in the figure refers to the growth rate.

3.1. Revisiting One-layer structure of DenseNet

Traditional CNN models typically have no more than one connection with the adjacent upper and adjacent lower layers. DenseNet is made up of multiple Dense-Blocks and a transition layer in between every two Dense-Blocks. In a Dense-Block, every layer gets input information from all the

previous layers. Within this context, information refers to feature maps. In a Bottleneck Dense-Block, information received from all the preceding layers goes through Batch Normalization (BN), then Rectified Linear Unit (ReLU), and then a convolutional layer with 1×1 kernel size which produces $4\times k$ feature maps. Here k is the 'growth rate' which denotes the number of newly generated feature maps. For all our proposed OLSs, we fixed the value of $k=32$. These $4\times k$ feature maps then go through BN, ReLU, and 3×3 convolutional layer which produces only k feature maps. All these operations take place within a Dense-Block, which is considered to be one-layer. The standard OLS of the Dense-Block is depicted in the figure Fig. 1.

To evaluate the effectiveness of our proposed OLSs in reducing the number of learnable parameters and computations, we must first determine the parameter and multiplication count of the original OLS. This analysis will provide a baseline for comparison with our proposed approach, allowing us to quantify the extent of parameter and computation reduction achieved by our proposed OLSs. The original OLS gets m number of input feature maps at 1×1 convolutional layer and generates $4\times k$ feature maps. Kernel width and height are denoted by w and h. So, the total number of learnable parameters at this layer is given by the equation:

$$m \times 4 \times k \times h \times w \tag{1}$$

as w and h are 1, therefore it becomes: $4\times m\times k$. The 4×k feature maps generated by 1×1 convolutional layer is then fed to 3×3 convolutional layer which in turn produces k feature maps. Then the number of learnable parameters 3×3 convolutional layer is given as:

$$m \times k \times k \times h \times w \tag{2}$$

Here h and w both are 3, so the total number of learnable parameters at this layer is given as $36\times k^2$. So total number of learnable parameters for standard OLS can be calculated as:

$$4 \times m \times k + 36 \times k^2 = 4k(m + 9k) \tag{3}$$

The number of multiplications occurring in standard OLS can be calculated as filter size multiplied by the output dimensions of generated feature maps. One filter volume size can be calculated as $h\times w\times m$. For 1×1 convolutional layer, volume size is $1\times 1\times m$. For 3×3 convolutional layer volume size can be calculated as $h\times w\times 4\times k$, but here h=3 and w=3, therefore, the filter volume size for 3×3 convolutional layer is $36\times k$. Dimension of generated feature maps after convolved with 1×1 convolutional layer is given as:

$$4 \times k \times H \times W \tag{4}$$

where H and W are the height and width of generated feature maps respectively. Because values of H and W are the same, therefore, the dimension of generated feature maps is $4\times k\times H^2$. Dimensions of generated feature maps after convolved with 3×3 convolution is given as $36\times k\times H^2$. Therefore, total number of multiplications needed after 1×1 convolution are:

$$4 \times k \times H^2 \tag{5}$$

and total number of multiplications needed after 3×3 convolution are:

$$36 \times k^2 \times H^2 \tag{6}$$

For the remaining parts of this paper, we will continue using the same notations to calculate the total number of parameters and the total number of multiplications for the proposed OLSs as we have done here.

Table 1. Comparison of the total number of parameters and the total number of multiplication operations required for standard and proposed OLSs of Dense-Block. 'k' is the growth rate of feature maps. 'm' is input feature maps to OLS. 'H' is the height of generated feature map.

Model Family	Total Parameters	Total Multiplications
Standard OLS	4k(m+9k)	4kH2(m+9k)
OLS 1.0	4k(m+17k)	4kH2(m+17k)
OLS 1.1	4k(m+14.625k)	4kH2(m+14.625k)
OLS 2.0	4k(m+8.5k)	4kH2(m+8.5k)
OLS 2.1	4k(m+7.87k)	4kH2(m+7.87k)
OLS 2.2	4k(m+4.25k)	4kH2(m+4.25k)
OLS 2.3	4k(m+3.938k)	4kH2(m+3.938k)
OLS 3.0	4k(m+17.25k)	4kH2(m+17.25k)
OLS 3.1	4k(m+11.5k)	4kH2(m+11.5k)
OLS 3.2	4k(m+17.25k)	4kH2(m+17.25k)
OLS 3.3	4k(m+13.75k)	4kH2(m+13.75k)

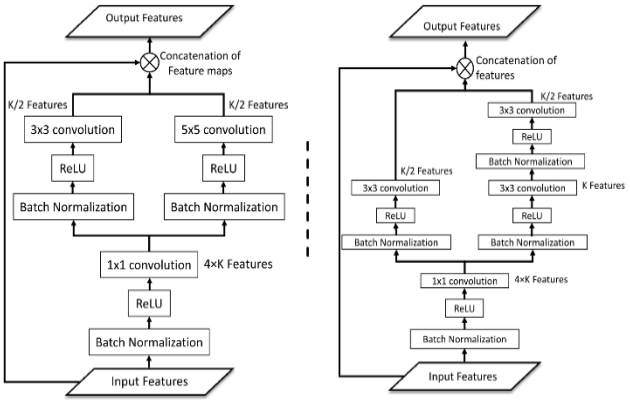

Fig 2. Proposed OLS 1.0 (left) and OLS 1.1 (right).

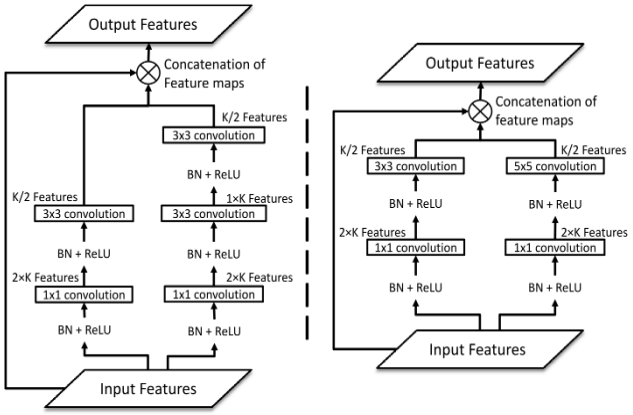

Fig 3. Proposed OLS 2.0 (right) and OLS 2.1 (left).

3.2. OLS 1.0 and OLS 1.1 (first family)

We employ the two key factors described in section 3 to design the one-layer structure for Dense-Block. In this proposed design, the cardinality of parallelism is two. Hereafter, we refer to cardinality as the degree of parallel layers in a one-layer structure. Figure Fig. 2 (left) depicts the first proposed one-layer structure. This OLS is referred to as 'OLS 1.0'. The first 1×1 convolution operation here generates 4×k (4k) feature maps. As mentioned earlier here the cardinality is two, therefore we have two parallel branches of convolution operations operating on 4k feature maps. One with

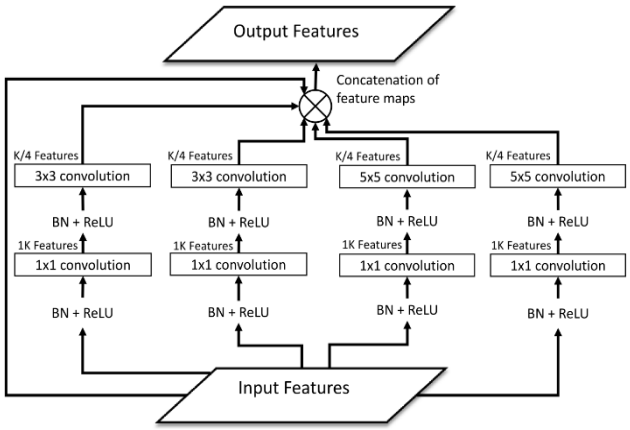

Fig 4. Proposed OLS 2.2.

3×3 kernel size and another with 5×5 kernel size. Each branch generates k/2 feature maps. The input feature maps to this OLS are then concatenated with the output of these two branches. We mentioned in the previous section that whenever we refer to input feature maps being processed through convolutional layers, this means they have passed through BN and ReLU processes prior to being processed through convolutional layers. We have followed this series of sequential operations throughout our proposed OLSs. The total learnable parameter counts and multiplication count required by OLS 1.0 are mentioned in the Table 1.

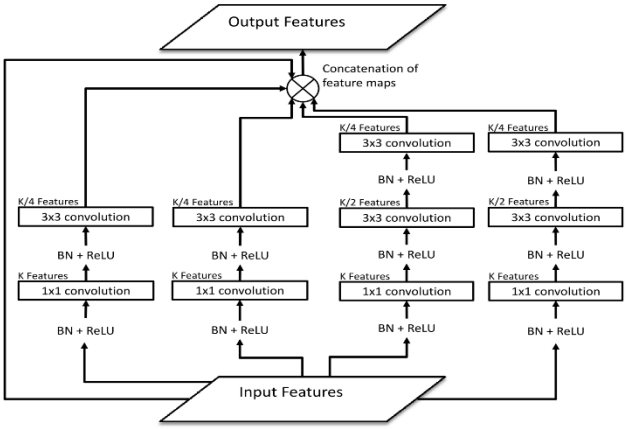

Fig 5. Proposed OLS 2.3.

Another model in this family, namely 'OLS 1.1', is proposed as a further extension of OLS 1.0. In the 'OLS 1.1', figure Fig. 2 (right), we propose to use two consecutive 3×3 convolutional layers instead of one 5×5 convolutional layer. The two consecutive convolutional layers of 3×3 kernel sizes cover the same receptive field as one 5×5 kernel. Comparing two consecutive 3×3 kernels to one 5×5 kernel has two advantages. The first advantage is that it requires fewer parameters, and the second is that it involves fewer computations.

The total learnable parameter counts and multiplication count required by OLS 1.1 are mentioned in the Table 1. It is evident that the structure of OLS 1.1 is lighter than the structure of OLS 1.0.

3.3. OLS 2.0, 2.1, 2.2, and 2.3 (second family)

It is quite evident from the proposed OLS 1.0 and OLS 1.1 that the number of parameters and computing operations is significantly greater than with the original DenseNet's OLS as a basis for comparison. We propose an evolving second family of one-layer structures for Dense-Block with cardinality two and four that is cost-effective, comparatively accurate, and lightweight. As shown in figure Fig. 3, OLS 2.0 and OLS 2.1 are configured in a simplified one-layer structure. The received feature maps are processed in parallel on two distinct pathways. In every first branch, they are passed through a convolutional layer of kernel size 1×1. The 2k feature maps are produced separately for each of the 1×1 convolutional layer on two parallel processing lines. In the structure of OLS 2.0, the output feature maps of 1×1 convolutional layer from the first branch are further processed by stacking a 3×3 convolution layer, an operation that generates k/2 feature maps. It is in the second branch that the output feature maps of the 1×1 convolutional layer are further processed with a 5×5 convolutional layer. This also results in the output of k/2 feature maps. Afterward, output feature maps of this OLS are generated by concatenating feature maps made from the two branches and the received input feature maps to this OLS. In the structure of OLS 2.1, we essentially maintained the first branch of

OLS 2.0. However, we replaced one 5×5 convolution with two 3×3 convolutional layers in the second branch. The output features from the 1×1 convolutional layer of the second branch is passed through the first 3×3 convolutional layer that produces 1*k* feature maps. The *k*/2 feature maps are then created using the second 3×3 convolutional layer with the intent of enriching these feature maps. To generate output feature maps, feature maps from the two branches and actual input feature maps received to this OLS are concatenated. Based on the calculations in Table 1, we conclude that these two structures are lighter than the structures of the first proposed family, as indicated by their lower number of learnable parameters and multiplications.

To make them more lightweight and enhance these OLSs further, we modelled two more one-layer structures in this family where cardinality is set to four. In the proposed model for OLS 2.2, all four parallel 1×1 convolutional layers process input feature maps and produce *k* feature maps. Parallel 3×3 convolutional layers from 2 left branches process *k* input feature maps independently, generating *k*/4 feature maps for each branch. Likewise, the remaining two parallel 5×5 convolutional layers from the right branches process *k* feature maps individually and generate *k*/4 feature maps for each branch. The output feature maps are generated by concatenating feature maps from all four branches and the input feature map received at this OLS. Figure Fig. 4 illustrates these details of the OLS 2.2 structure in a clear and concise manner. Later, an enhanced version of OLS 2.2 is developed, namely OLS 2.3, in which the 5×5 convolutional layer from OLS 2.2 model is broken down into two successive 3×3 convolutional layers. The first 3×3 convolutional layer generates *k*/2 feature maps and the second 3×3 convolutional layer generates *k*/4 feature maps. This OLS is outlined in figure Fig. 5. Our calculations over these two structures, as presented in the Table 1, show that OLS 2.2 and OLS 2.3 have lighter structures than the previous ones, as demonstrated by their lower number of learnable parameters and multiplications.

3.4. OLS 3.0, 3.1, 3.2, and 3.3 (third family)

We have designed four further new OLSs, labelled OLS 3.0 to OLS 3.3. These are enhanced versions of the OLSs ranging from 2.0 to 2.3. Two of the structures have a cardinality of two, while the other two have a cardinality of four. Models 3.0 and 3.1 are implemented as one-layer structures similar to those of 2.0 and 2.1 respectively with a single additional layer before the concatenation of feature maps. That additional layer is a 1×1 convolutional layer in each pathway. Figure Fig. 6 illustrates the one-layer structure designs for OLS 3.0 and OLS 3.1. Similarly, one-layer structure designs of models 3.2 and 3.3 are developed as those of 2.2 and 2.3

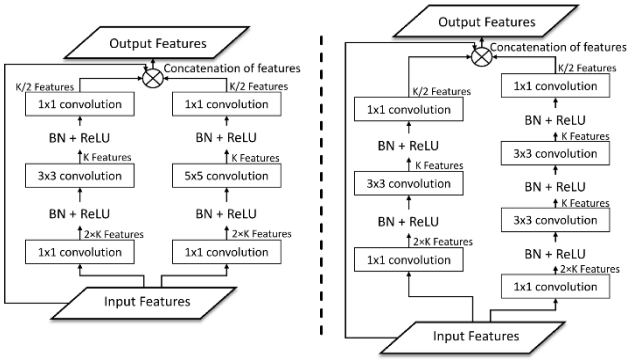

Fig 6. Proposed OLS 3.0 (left) and OLS 3.1 (right).

respectively, with an additional one 1×1 convolutional layer in each branch. The illustration of OLS 3.2 and 3.3 are depicted in figures Fig. 7 and Fig. 8 respectively. Comparison of total required parameters and total multiplication operations by all these OLSs are presented in the Table 1.

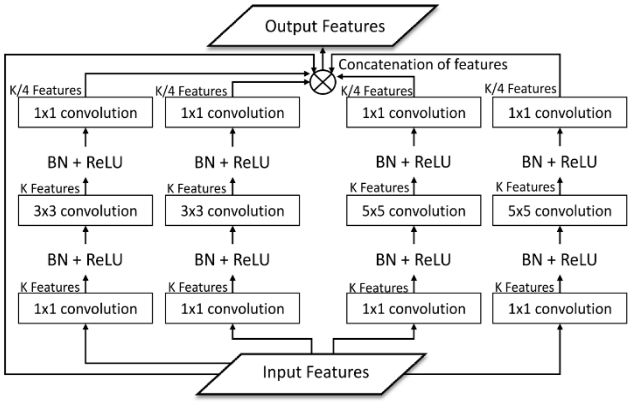

Fig 7. Proposed OLS 3.2.

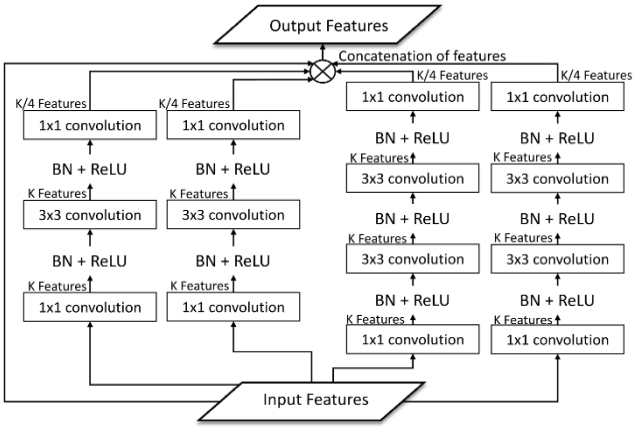

Fig 8. Proposed OLS 3.3.

3.5. Architecture designs

We used the same design architecture framework as the original DenseNet-BC except for some minor modifications in layers and growth rate. The growth rate of every model is fixed at 32 (*k*=32). We developed our first three families of models for the CIFAR dataset where the input shape is 32×32×3. A second three families of models have been built for the Imagenette dataset, which has an input shape of

224×224×3. Models for CIFAR-10 and CIFAR-100 are almost identical with the exception of the last layer. The output layer of the CIFAR-10 model has 10 neurons, while the output layer of the CIFAR-100 model has 100 neurons. Models for a 32×32×3 input shape composed of three Dense-Blocks. There are 6, 12, and 24 one-layer structures within each Dense-Block. Two consecutive Dense-Blocks are separated by an intermediate layer called a transition layer. Any of the OLSs that we proposed could be used as the one-layer structure. Table 2 has the details of the model for 32×32×3 shape of inputs. In the case of the input shape 224×224×3, there are initially two convolutional layers followed by four Dense-Block layers. Each Dense-Block contains 6, 12, 24, and 18 one-layer structures. From our proposed designs, we can use any one of the OLS. In addition to this, there is a transition layer in between each set of consecutive dense blocks. The transition layer consists of one 1×1 convolutional layer and a 2×2 average pool layer. Architectural details for input shape 224×224×3 is mentioned in the Table 3.

Table 2. Architecture details for CIFAR-10 and CIFAR-100. 'D' is the depth of a model. 'OLS' stands for 'one-layer structure'. 'P' denotes padding and 'S' denotes stride.

Layers	Output Size	k=32, D=47
Convolution	32 × 32	3 × 3 Conv, P=1, S=1
Dense Block–1	32 × 32	Proposed OLS × 6
Transition Layer–1	16 × 16	1 × 1 Conv, 2 × 2 Avg. Pool
Dense Block–2	16 × 16	Proposed OLS × 12
Transition Layer–2	8 × 8	1 × 1 Conv, 2 × 2 Avg. Pool
Dense Block–3	8 × 8	Proposed OLS × 24
Average Pool	1 × 1	Global Average Pooling
Classification Layer	10-D or 100-D fully connected linear layer	

Table 3. Architecture details for Imagenette. 'D' is the depth of a model. 'OLS' stands for 'one-layer structure'. 'P' denotes padding and 'S' denotes stride.

Layers	Output Size	k=32, D=47
Convolution	112 × 112	7 × 7 Conv, P=3, S=2
Convolution	56 × 56	7 × 7 Conv, P=3, S=2
Dense Block–1	56 × 56	Proposed OLS × 6
Transition Layer–1	28 × 28	1 × 1 Conv, 2 × 2 Avg. Pool
Dense Block–2	28 × 28	Proposed OLS × 12
Transition Layer–2	14 × 14	1 × 1 Conv, 2 × 2 Avg. Pool
Dense Block–3	14 × 14	Proposed OLS × 24
Transition Layer – 3	7 × 7	1 × 1 Conv 2 × 2, Avg. Pool
Dense Block – 4	7 × 7	Proposed OLS × 18
Average Pool	1 × 1	Global Average Pooling
Classification Layer	10-D fully connected linear layer	

4. Experimental Results

4.1. Datasets

4.1.1. CIFAR-10 & CIFAR-100

CIFAR-10 [48] dataset consists of 10 class types and 60000 RGB images of a 32×32 resolution, which is broken down into training data and testing data. There are a total of 50,000 images in the training dataset, while the rest of the images appear in the testing dataset. A CIFAR-100 dataset [48] also consists of 50,000 training images and 10,000 test images distributed across 100 categories within the dataset. For the purposes of augmentation, the input images were cropped into 32×32 squares and horizontally flipped as part of the process. Images are cropped at random from the input images after they have been padded by four pixels.

4.1.2. Imagenette

Images in the Imagenette [49] dataset, a subset of the ImageNet dataset, are classified into ten categories (classes). Each of the ten categories in the Imagenette consists of approximately 950 images. A total of 9469 images were used for the training set, while 3925 were used for the validation set, and all of them have been grouped into 10 categories. There are approximately 400 images in each category of the Imagenette in the validation set. We have applied the techniques that were used in the papers [6,50,51] in order to enhance training images in this work. We applied a center crop of 224×224 pixels to our dataset in order to test the efficiency of our model for test data. The presence of color jitter on the training set is determined by adjusting a

jitter value of 0.4. It randomly adjusts brightness, contrast, and saturation. As a regularization technique, a label smoothing technique using an epsilon of 0.1 is employed.

4.2. Experiments on CIFAR-10, and CIFAR-100

Dense-Block includes one of the proposed OLSs, creating 10 models with different configurations. In order to demonstrate the progressive innovation of our model with respect to the original DenseNet, we have conducted extensive experiments with our proposed models. Our networks are first trained on the CIFAR-10 dataset. We used Stochastic Gradient Descent (SGD) as an optimization method to train all the models. The momentum value is set to 0.9. All the models have been trained for 200 epochs. Cosine Annealing has been used for scheduling the learning rate, and initially, the learning rate is set to 0.1. The value for weight decay is 0.0001. Weight matrices are initialized as per the recommendation of this [52] paper. Mini-batch size is 64 for all the models. The same hyper-parameters and training strategy we used for models designed for the CIFAR-100 dataset. To ensure a fair comparison, the original DenseNet model was also trained under an identical training process. In consequence, the original performance was marginally downgraded. In both datasets, the validation set was used to evaluate the models.

4.3. Experiments on Imagenette

The architecture specifications for the Imagenette dataset as an input to our model are mentioned in the Table 3. Any one of the proposed OLSs of the same type can be incorporated into that. Therefore, ten different models are possible. Each of these models was trained using SGD. The momentum value is 0.9. We trained all the models for 310 iterations, 10 of which were chosen as cool-down epochs. We used these 10 epochs to achieve stability of the learning rate at a minimum value. For weight decay, this value is 0.00002. There are 64 samples in a mini-batch. We are using a cosine annealing learning rate scheduler with an initial learning rate value of 0.05. Kernel weight matrices are initialized with the values suggested by the [52] paper.

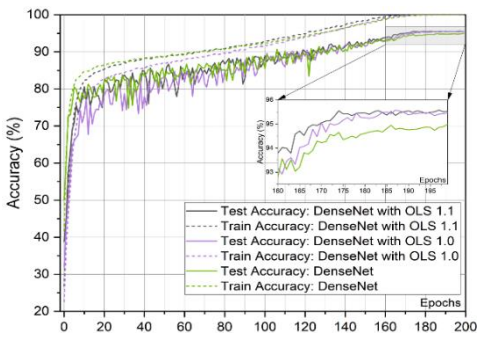

Fig 9. Train and Test set accuracies on CIFAR-10 of DenseNet with OLS 1.0, OLS 1.1, and standard DenseNet.

4.4. Experimental results

4.4.1. Results on CIFAR-10

DenseNet, the state-of-the-art architecture in computer vision, serves as the benchmark base model for our proposed models. With k=32 and 4.36 million parameters, DenseNet demonstrates an accuracy rate of 94.99% on the CIFAR-10 benchmark dataset. In an effort to surpass the performance of DenseNet, we proposed the

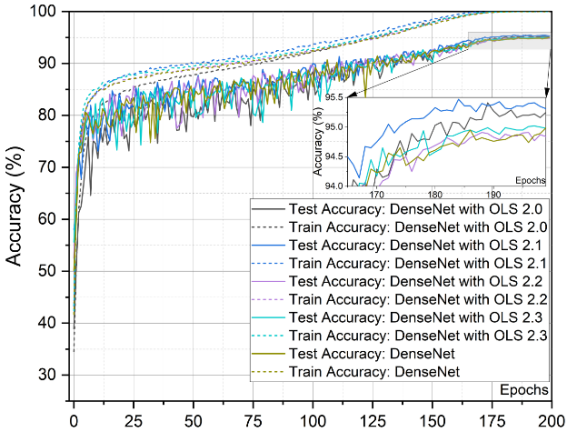

Fig 10. Train and Test set accuracies on CIFAR-10 of DenseNet with OLS 2.0 to OLS 2.3, and standard DenseNet.

first model of our family, designated as 'DenseNet with OLS 1.0', which incorporated additional parameters, specifically a total of 5.74 million parameters. Through extensive experimentation and evaluation, DenseNet with OLS 1.0 achieved an accuracy rate of 95.56%, surpassing the performance of the base model. However, we recognized that the number of parameters can have a significant impact on the overall efficiency of the model. Therefore, the second model DenseNet with OLS 1.1 incorporated a reduced number of parameters, specifically a total of 5.33 million parameters. Despite the reduction in parameters, DenseNet with OLS 1.1 achieved an accuracy rate of 95.53%, which is comparable to the accuracy of DenseNet with OLS 1.0 and higher than the base model DenseNet. Fig. 9 shows the learning curves of DenseNet and DenseNet with OLS 1.0, and 1.1.

The first two models in the second family, designated as 'OLS 2.0' and 'OLS 2.1', incorporate a unique architecture with two parallel branches. Despite having a lower number of parameters, specifically a total of 4.27 and 4.17 million respectively, both models demonstrate a high level of accuracy, achieving rates of 95.25% and 95.31% respectively on the CIFAR-10 dataset. These results suggest that a reduction in parameters may not necessarily result in a decrease in performance and may even potentially lead to improved accuracy.

In an effort to further optimize the performance of our proposed models, we introduced the last two models in this

family, designated as 'OLS 2.2' and 'OLS 2.3'. These OLSs incorporate a more complex architecture, with four parallel branches. OLS '2.2' has 3.54 million parameters, while OLS '2.3' has 3.49 million parameters. Both models achieved an accuracy rate of 94.84% and 94.98% respectively. While the accuracy is marginally lower than the original DenseNet model, the number of parameters is considerably lower as well. In fact, OLS '2.3' has been proven to be almost as accurate as the original DenseNet model. Graph Fig. 10 shows the learning curves of DenseNet with a second family and standard DenseNet.

We proposed a third family of models to further investigate the relationship between model architecture, number of parameters, and accuracy. The first two models in this family, incorporate a unique architecture with two parallel branches, while the last two models have a more complex architecture with four parallel branches. Despite having a higher number of parameters than the original DenseNet model and our earlier proposed models, all models within this family achieved a remarkably high level of accuracy on the CIFAR-10 dataset. These results are presented in the Table 4 of our paper.

These results of proposed OLSs with DenseNet suggest that an increase in the number of parameters does not necessarily result in a decrease or saturation in performance and that the architecture of the model plays a crucial role in determining the accuracy of the model. Moreover, the high accuracy achieved by these models, despite having more parameters in the third family than the previous models, highlights the potential of our proposed models to outperform the state-of-the-art DenseNet model.

4.4.2. Results on CIFAR-100

The original DenseNet model achieves 80.53% accuracy on CIFAR-100. The first family of proposed OLS models, namely OLS 1.0, and OLS 1.1 achieves 81.07%, and 81.24% accuracy respectively. OLS 1.0 and OLS 1.1 exhibit the same pattern of results as CIFAR-10, i.e., a greater number of parameters than standard DenseNet, and a higher level of accuracy. Among the first two models of the second family, OLS 2.0 and OLS 2.1, provide improved accuracy for fewer parameters than the original DenseNet. The next two models from the same family, OLS 2.2 and OLS 2.3 achieve almost similar accuracy with considerably fewer parameters than the original DenseNet model. In all four models that are part of the proposed third family, OLS 3.0 to OLS 3.3, an improvement in accuracy is observed over all the previously proposed models and the baseline DenseNet model. Since the number of convolutions per 1×1 is higher in the third family, the parameters are higher as well. All the results are shown in Table 4. The train and test accuracy curves of DenseNet and DenseNet with the third family are shown in the Fig. 11.

Table 4. Results on CIFAR-10 and CIFAR-100.

Model	Parameters (Million)	FLOPs (GMAC)	CIFAR-10 Accuracy	CIFAR-100 Accuracy
SWRN 28-10-1 [13]	12	-	95.99%	80.77%
WRN 28-10 [13]	36	-	96%	80.75%
CCT-7 / 3 × 1 [12]	3.76	0.95	94.78%	77.05%
DenseNet	4.36	0.83	94.99%	80.53%
DenseNet with OLS 1.0	5.74	1.19	95.46%	81.07%
DenseNet with OLS 1.1	5.33	1.08	95.53	81.24%
DenseNet with OLS 2.0	4.27	0.81	95.25%	80.60%
DenseNet with OLS 2.1	4.17	0.79	95.31%	80.69%
DenseNet with OLS 2.2	3.54	0.64	94.84%	80.33%
DenseNet with OLS 2.3	3.49	0.62	94.98%	80.44%
DenseNet with OLS 3.0	5.78	1.2	95.73%	81.72%
DenseNet with OLS 3.1	4.8	0.95	95.45%	81.58%
DenseNet with OLS 3.2	5.79	1.21	95.89%	81.78%
DenseNet with OLS 3.3	5.19	1.06	95.58%	81.64%

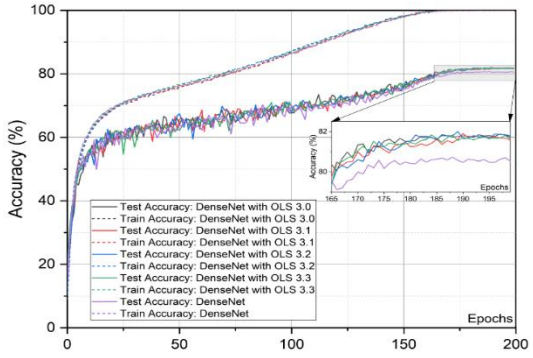

Fig 11. Train and Test set accuracies on CIFAR-100 of DenseNet with OLS 3.0, OLS 3.1, OLS 3.2, OLS 3.3, and standard DenseNet.

4.5. Results on Imagenette

Our empirical investigation on the Imagenette dataset has yielded noteworthy findings, particularly in the context of the ARViT [53] and SmoothNet [54] architectures. ARViT, boasting a parameter count of 10 million, has demonstrated an impressive accuracy level, reaching 92.05%. Conversely, the reduction in parameter complexity within the SmoothNet model has led to a diminished accuracy performance, registering at 69.7%.

With 7.93 million parameters, the standard DenseNet model on Imagenette achieves 92.7% accuracy. The proposed OLS designs from each family achieve higher accuracy than standard DenseNet; however, not all the models have fewer parameters. The proposed models from the second family have a much smaller number of parameters, especially DenseNet with OLS 2.2 and DenseNet with OLS 2.3. DenseNet with OLS 3.2 has 9.97 million parameters and delivers 94.01% accuracy. In contrast, DenseNet with OLS 3.3 has 9.12 million parameters and achieves 94.36% accuracy. Results are mentioned in the Table 5 and figure Fig. 12 shows learning curves of test accuracies on Imagenette. Figure Fig. 13 shows the trade-off between number of parameters, number of floating-point operations, and accuracies of the models. Figure Fig. 13 shows GradCAM++ results with respective softmax scores of plain DenseNet and DenseNet with second family of OLS.

Table 5. Results on Imagenette

Model	Parameters (Million)	FLOPs (GMAC)	Imagenette Accuracy
ARViT	10	-	92.05%
SmoothNet	3.4	-	69.7%
DenseNet	7.93	4.42	92.713%
DenseNet with OLS 1.0	9.9	5.54	93.936%
DenseNet with OLS 1.1	9.32	5.21	93.885%
DenseNet with OLS 2.0	7.808	4.37	93.427%
DenseNet with OLS 2.1	7.66	4.29	93.478%
DenseNet with OLS 2.2	6.76	3.83	93.325%
DenseNet with OLS 2.3	6.69	3.79	93.427%
DenseNet with OLS 3.0	9.97	5.59	94.166%
DenseNet with OLS 3.1	8.56	4.8	93.987%
DenseNet with OLS 3.2	9.97	5.64	94.013%
DenseNet with OLS 3.3	9.12	5.16	94.369%

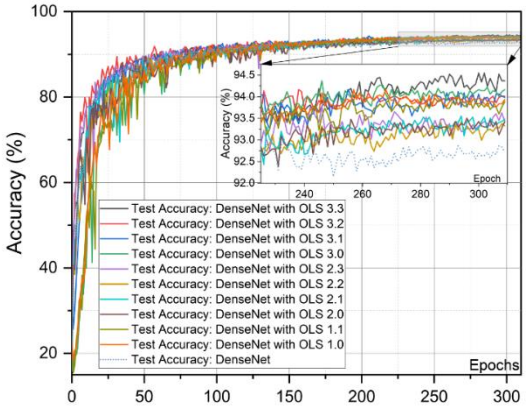

Fig 12. Test accuracies of DenseNet with all the proposed OLSs and standard DenseNet on Imagenette.

5. Observations

5.1. 3×3 and 5×5 kernel size in split branch

We employed an architecture that utilizes a combination of two parallel branches, one with a 5×5 kernel and the other with a 3×3 kernel. This architecture was implemented in our proposed OLSs 1.0 and 1.1 for the CIFAR and Imagenette datasets. The results of our experiments demonstrate that this architectural design leads to a significant improvement in the accuracy of the models on these datasets, resulting in a better overall model performance.

5.2. Two 3×3 kernels instead of one 5×5

In order to improve the efficiency of our proposed OLSs, we employed a unique architectural design that utilizes multiple smaller kernel sizes in place of a single large kernel. Specifically, instead of using a single 5×5 filter, which would require 25 parameters to be learned, we used two successive 3×3 kernels, which only require 18 parameters to be learned, resulting in a 28% reduction in the number of parameters to be learned. Through extensive experimentation and evaluation, we have demonstrated that this architectural design leads to results that are similar or have slightly better accuracy compared to models that utilize a single large kernel.

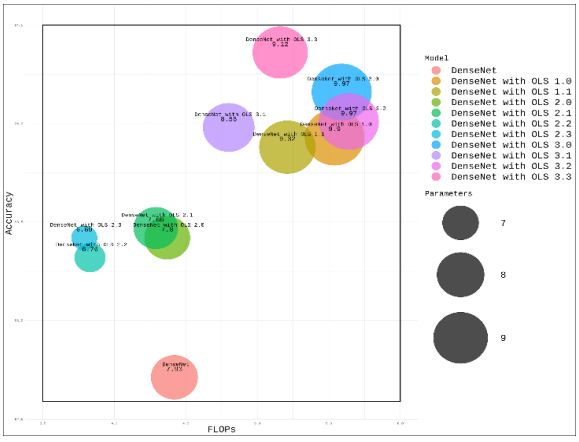

Fig 13. Relationship between test accuracies, parameters, and FLOPs of models on Imagenette.

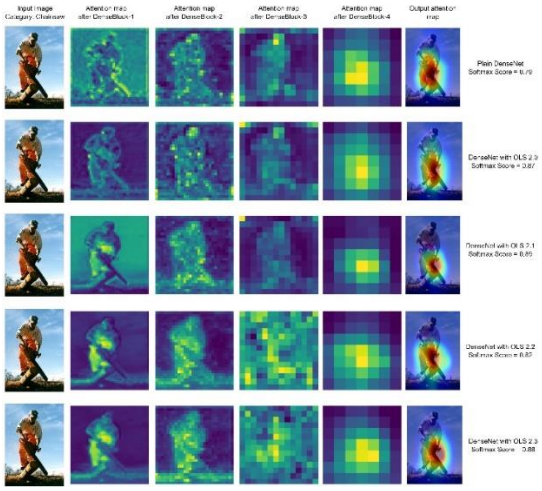

Fig 14. GradCAM++ and softmax score results of second family of OLS on Imagenette input image of chainsaw category.

5.3. More parallel branches results in less parameters and better accuracy

The two parallel branches were used to construct OLSs 2.0, 2.1, 3.0, and 3.1. While OLSs 2.2, 2.3, 3.2, and 3.3 were constructed with 4 parallel branches. Empirical evaluations have been conducted to demonstrate that the latter versions with four branches exhibit a reduction in the number of parameters while concurrently achieving improved accuracy.

5.4. Using more 1×1 convolutions give better results

The third family of models under consideration incorporates a greater number of 1×1 convolutional layers. Empirical evaluations have demonstrated that the incorporation of additional 1×1 convolution operations lead to a marked improvement in accuracy, although this performance gain comes at the cost of an increase in the number of parameters.

6. Conclusion and Future Work

By leveraging DenseNet as a framework, we suggested ten intricately constructed One-Layer Structure architecture designs for Dense-Block categorized into three families. In this research work, we examined whether the use of a kernel of 5×5 in a split branch of Dense-Block leads to increased accuracy. Rather than using one 5×5 kernel, two successive 3×3 kernels and breaking a branch into multiple parallel branches help to achieve a greater degree of accuracy and reduce the number of parameters that are to be learned. It is more advantageous to use multiple split branches and two consecutive 3×3 kernels rather than a 5×5. It reduces the number of parameters considerably. Along with these strategies, in the third family of models, there is a straightforward method that uses more 1×1 convolution at the end in the split branch of the Dense-Block. As a result, the accuracy increases, but at the expense of adding more parameters. In spite of this, we do not make any claims that our models are state-of-the-art; rather they serve as a starting point for further optimization through a collaborative effort between the architecture and training techniques. Our research highlighted the potential of utilizing split branches and various kernel sizes in improving the accuracy of image classification models and the importance of balancing the number of parameters with the desired level of accuracy.

Our forthcoming research will tackle the issue of reducing the amplified memory needs that ensue from dividing a single branch into multiple branches and then merging them. Inadequate hardware support for these increased memory demands may lead to suboptimal training performance. Our future work will aim to optimize memory usage for multiple branches to further enhance the efficiency of our proposed method.

References

[1] Liu S, Deng W. Very deep convolutional neural network based image classification using small training sample size. In2015 3rd IAPR Asian conference on pattern recognition (ACPR) 2015 Nov 3 (pp. 730-734). IEEE.

[2] Nair V, Hinton GE. Rectified linear units improve restricted boltzmann machines. InProceedings of the 27th international conference on machine learning (ICML-10) 2010 (pp. 807-814).

[3] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. InProceedings of the IEEE conference on computer vision and pattern recognition 2015 (pp. 1-9).

[4] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 2818-2826).

[5] Szegedy C, Ioffe S, Vanhoucke V, Alemi A. Inception-v4, inception-resnet and the impact of residual connections on learning. InProceedings of the AAAI conference on artificial intelligence 2017 Feb 12 (Vol. 31, No. 1).

[6] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).

[7] Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. Densely connected convolutional networks. InProceedings of the IEEE conference on computer vision and pattern recognition 2017 (pp. 4700-4708).

[8] Girshick R. Fast r-cnn. InProceedings of the IEEE international conference on computer vision 2015 (pp. 1440-1448).

[9] Girshick R, Donahue J, Darrell T, Malik J. Rich feature

hierarchies for accurate object detection and semantic segmentation. InProceedings of the IEEE conference on computer vision and pattern recognition 2014 (pp. 580-587).

[10] Bendou Y, Hu Y, Lafargue R, Lioi G, Pasdeloup B, Pateux S, Gripon V. Easy—ensemble augmented-shot-y-shaped learning: State-of-the-art few-shot classification with simple components. Journal of Imaging. 2022 Jun 24;8(7):179.

[11] Ridnik T, Sharir G, Ben-Cohen A, Ben-Baruch E, Noy A. Ml-decoder: Scalable and versatile classification head. InProceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision 2023 (pp. 32-41).

[12] Jin C, Liang J, Fan C, Chen L, Wang Q, Lu Y, Wang K. Study on segmentation of blasting fragment images from open-pit mine based on U-CARFnet. Plos one. 2023 Sep 14;18(9):e0291115.

[13] Shen X, Wang H, Wei B, Cao J. Real-time scene classification of unmanned aerial vehicles remote sensing image based on Modified GhostNet. Plos one. 2023 Jun 7;18(6):e0286873.

[14] Hassani A, Walton S, Shah N, Abuduweili A, Li J, Shi H. Escaping the big data paradigm with compact transformers. arXiv preprint arXiv:2104.05704. 2021 Apr 12.

[15] Savarese P, Maire M. Learning implicitly recurrent CNNs through parameter sharing. arXiv preprint arXiv:1902.09701. 2019 Feb 26.

[16] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861. 2017 Apr 17.

[17] Chiang HY, Frumkin N, Liang F, Marculescu D. MobileTL: on-device transfer learning with inverted residual blocks. InProceedings of the AAAI Conference on Artificial Intelligence 2023 Jun 26 (Vol. 37, No. 6, pp. 7166-7174).

[18] Zhou D, Hou Q, Chen Y, Feng J, Yan S. Rethinking bottleneck structure for efficient mobile network design. InComputer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16 2020 (pp. 680-697). Springer International Publishing.

[19] Han K, Wang Y, Tian Q, Guo J, Xu C, Xu C. Ghostnet: More features from cheap operations. InProceedings of the IEEE/CVF conference on computer vision and pattern recognition 2020 (pp. 1580-1589).

[20] Zhang X, Zhou X, Lin M, Sun J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. InProceedings of the IEEE conference on computer vision and pattern recognition 2018 (pp. 6848-6856).

[21] Zhang H, Zhu X, Li B, Guan Z, Che W. LA-ShuffleNet: A Strong Convolutional Neural Network for Edge Computing Devices. IEEE Access. 2023 Oct 16.

[22] Huang G, Liu S, Van der Maaten L, Weinberger KQ. Condensenet: An efficient densenet using learned group convolutions. InProceedings of the IEEE conference on computer vision and pattern recognition 2018 (pp. 2752-2761).

[23] Li X, Wang W, Hu X, Yang J. Selective kernel networks. InProceedings of the IEEE/CVF conference on computer vision and pattern recognition 2019 (pp. 510-519).

[24] Wang F, Jiang M, Qian C, Yang S, Li C, Zhang H, Wang X, Tang X. Residual attention network for image classification. InProceedings of the IEEE conference on computer vision and pattern recognition 2017 (pp. 3156-3164).

[25] Fu J, Liu J, Tian H, Li Y, Bao Y, Fang Z, Lu H. Dual attention network for scene segmentation. InProceedings of the IEEE/CVF conference on computer vision and pattern recognition 2019 (pp. 3146-3154).

[26] Chen L, Zhang H, Xiao J, Nie L, Shao J, Liu W, Chua TS. Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. InProceedings of the IEEE conference on computer vision and pattern recognition 2017 (pp. 5659-5667).

[27] Chen S, Liu Y, Gao X, Han Z. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. InBiometric Recognition: 13th Chinese Conference, CCBR 2018, Urumqi, China, August 11-12, 2018, Proceedings 13 2018 (pp. 428-438). Springer International Publishing.

[28] Li H, Xiong P, An J, Wang L. Pyramid attention network for semantic segmentation. arXiv preprint arXiv:1805.10180. 2018 May 25.

[29] Jiang Y, Cheng T, Dong J, Liang J, Zhang Y, Lin X, Yao H. Dermoscopic image segmentation based on Pyramid Residual Attention Module. Plos one. 2022 Sep 16;17(9):e0267380.

[30] Choi M, Kim H, Han B, Xu N, Lee KM. Channel attention is all you need for video frame interpolation. InProceedings of the AAAI Conference on Artificial Intelligence 2020 Apr 3 (Vol. 34, No. 07, pp. 10663-10671).

[31] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate

shift. In International conference on machine learning 2015 Jun 1 (pp. 448-456). pmlr.

[32] Hu J, Shen L, Sun G. Squeeze-and-excitation networks. InProceedings of the IEEE conference on computer vision and pattern recognition 2018 (pp. 7132-7141).

[33] Zhang T, Qi GJ, Xiao B, Wang J. Interleaved group convolutions. InProceedings of the IEEE international conference on computer vision 2017 (pp. 4373-4382).

[34] Ullrich K, Meeds E, Welling M. Soft weight-sharing for neural network compression. arXiv preprint arXiv:1702.04008. 2017 Feb 13.

[35] Han S, Mao H, Dally WJ. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149. 2015 Oct 1.

[36] Zhang X, Colbert I, Kreutz-Delgado K, Das S. Training deep neural networks with joint quantization and pruning of weights and activations. arXiv preprint arXiv:2110.08271. 2021 Oct 15.

[37] Lu Q, Jiang W, Xu X, Hu J, Shi Y. Quantization Through Search: A Novel Scheme to Quantize Convolutional Neural Networks in Finite Weight Space. InProceedings of the 28th Asia and South Pacific Design Automation Conference 2023 Jan 16 (pp. 378-383).

[38] Choi Y, El-Khamy M, Lee J. Towards the limit of network quantization. arXiv preprint arXiv:1612.01543. 2016 Dec 5.

[39] Pham H, Guan M, Zoph B, Le Q, Dean J. Efficient neural architecture search via parameters sharing. In International conference on machine learning 2018 Jul 3 (pp. 4095-4104). PMLR.

[40] Liu C, Zoph B, Neumann M, Shlens J, Hua W, Li LJ, Fei-Fei L, Yuille A, Huang J, Murphy K. Progressive neural architecture search. InProceedings of the European conference on computer vision (ECCV) 2018 (pp. 19-34).

[41] Real E, Aggarwal A, Huang Y, Le QV. Regularized evolution for image classifier architecture search. InProceedings of the aaai conference on artificial intelligence 2019 Jul 17 (Vol. 33, No. 01, pp. 4780-4789).

[42] Guo X, Wu Y, Miao J, Chen Y. LiteGaze: Neural architecture search for efficient gaze estimation. Plos one. 2023 May 1;18(5):e0284814.

[43] Zoph B, Vasudevan V, Shlens J, Le QV. Learning transferable architectures for scalable image recognition. InProceedings of the IEEE conference on computer vision and pattern recognition 2018 (pp. 8697-8710).

[44] Radosavovic I, Kosaraju RP, Girshick R, He K, Doll´ar P. Designing network design spaces. InProceedings of the IEEE/CVF conference on computer vision and pattern recognition 2020 (pp. 10428-10436).

[45] Ding X, Zhang X, Han J, Ding G. Diverse branch block: Building a convolution as an inception-like unit. InProceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2021 (pp. 10886-10895).

[46] Ding X, Guo Y, Ding G, Han J. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. InProceedings of the IEEE/CVF international conference on computer vision 2019 (pp. 1911-1920).

[47] Ding X, Zhang X, Ma N, Han J, Ding G, Sun J. Repvgg: Making vgg-style convnets great again. InProceedings of the IEEE/CVF conference on computer vision and pattern recognition 2021 (pp. 13733-13742).

[48] Krizhevsky A, Nair V, Hinton G. Cifar-10 (canadian institute for advanced research). URL http://www. cs. toronto. edu/kriz/cifar. html. 2010 Mar;5(4):1.

[49] Howard J. Imagewang. URL https://github. com/fastai/imagenette. 2019.

[50] He K, Zhang X, Ren S, Sun J. Identity mappings in deep residual networks. InComputer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14 2016 (pp. 630-645). Springer International Publishing.

[51] Gross S, Wilber M. Training and investigating residual nets. Facebook AI Research. 2016 May;6(3).

[52] He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. InProceedings of the IEEE international conference on computer vision 2015 (pp. 1026-1034).

[53] Mormille LH, Broni-Bediako C, Atsumi M. Regularizing self-attention on vision transformers with 2D spatial distance loss. Artificial Life and Robotics. 2022 Aug;27(3):586-93.

[54] Remerscheid NW, Ziller A, Rueckert D, Kaissis G. Smoothnets: Optimizing cnn architecture design for differentially private deep learning. arXiv preprint arXiv:2205.04095. 2022 May 9.