A Hybrid Approach to Image Classification on Densely Connected Convolutional Networks

Ta Nguyen Thanh
Institute for Artificial Intelligence
University of Engineering and Technology, Vietnam National University
E3 Building, 144 Xuan Thuy Street, Cau Giay District, Hanoi
23020437@vnu.edu.vn

Abstract

Image classification has been a central task in Machine Learning for many years, and numerous Convolutional Neural Networks have been constructed such as Visual Geometry Group (VGG Network), ResNet (Residual Network) and DenseNet. Following the paper by Huang et al. [1], we introduce a hybrid approach to this image classification problem by integrating several different methods, specifically, we incorporate Drop-Channel [3] regularization and Squeeze-and-Excitation (SE) blocks [4] into a DenseNet architecture built from scratch instead of being a pre-trained model. Finally, in order to efficiently scale this model efficiently into wider and deeper networks while utilizing GPU resources, we adapt the technique of using checkpoint to calculate gradients, also known as Memory Efficient DenseNet [2]. By applying multiple suitable practices to experiments on the two prevalent datasets CIFAR100 and Tiny ImageNet, we achieved promising results and improvements over the vanilla version of the network. This result enables deeper research for DenseNet with deeper layers and higher growth rate to bring about even better performance, not only in classification accuracy but also in memory efficiency.

1 Introduction

In the field of Machine Learning, Convolutional Neural Networks (CNNs) have emerged and become the primary approach when it comes to image or visual classification. Due to the ability to extract feature maps and learn from those maps, CNNs have proven to be highly effective in classifying various classes of images. CNNs are now widely applied in real-life problems and fields such as security, autonomous driving and especially medical imaging where they are wielded for the purpose of discovering patients' illnesses to administer medication for them. Due to their vast appearance in a great deal of fields in everyday life, and the needs of many people, CNNs are proving themselves to be one of the most versatile structures in Machine Learning.

After the advent of LeNet [5], a more complex network appeared with higher accuracy and easier training process, which is the AlexNet [6], although it has an Achilles heel when it comes to efficiency. Then there comes the ResNet [7], which comprises residual blocks, and has been innovated and improved in numerous ways in various distinctive papers. However, the residual mapping from this structure turns out to be expensive in computing, and very deep ResNets can suffer from training imbalance.

The DenseNet architecture [1] is suitable for hard problems in real life, since it can effectively learn complex hierarchical features while mitigating the vanishing gradient problem. Its unique connectivity pattern, where each layer receives feature maps from all preceding layers, fosters feature reuse and propagation, leading to remarkable parameter efficiency and strong performance.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Despite DenseNet's inherent strengths, there's always room for further improvement in model performance and resource utilization. This paper explores a hybrid approach to enhance the DenseNet architecture, without relying on pre-trained models. We integrate three key methods: Drop-Channel regularization [3] for improved generalization, Squeeze-and-Excitation (SE) [4] blocks to enhance feature recalibration, and Memory Efficient DenseNet [2] (checkpointing in PyTorch) to enable deeper and wider networks while managing GPU memory effectively. By combining these proven techniques within a DenseNet built from scratch, we aim to demonstrate a synergistic effect that pushes the boundaries of its classification accuracy and memory efficiency.

We evaluate our approach on the widely used CIFAR100 and Tiny ImageNet datasets, showcasing promising results that surpass the vanilla DenseNet. This research paves the way for future explorations into even deeper DenseNet architectures with higher growth rates, promising advancements not only in classification performance but also in computational practicality. Code are models are available at this GitHub repository.

2 Related work

The field of neural network architectures has seen significant progression since the 1980s, particularly with the advent of Convolutional Neural Networks (CNNs) in place of Multi-Layer Perceptrons (MLPs). LeNet [5] demonstrated the efficacy of CNNs by implementing more complex, yet trainable, structures for tasks like handwritten digit recognition. However, the demand for more sophisticated feature extraction and hierarchical composition from input images led to the development of deeper convolutional neural networks, most notably AlexNet [6]. While significantly deeper than its predecessor LeNet, AlexNet's full impact and widespread adoption by the academic community took time.

VGG [8], on the other hand, applied the idea of using repeated blocks in deep learning to make use of loops and subroutines. It demonstrated that increasing the total depth of the network, primarily by stacking very small 3×3 convolutional filters, consistently improved accuracy and performance. This architectural choice allowed deeper layers to learn increasingly complex and abstract features.

In other structures such as GoogLeNet [9] where the novel Inception Module was introduced, parallel branches with different filter sizes are utilized to capture multi-scale feature. In order to mitigate computational cost, 1 x 1 convolutional layers are integrated for dimensionality reduction prior to larger convolutions. At the network's end, the fully connected layer is replaced with a global average pooling to significantly reduce parameters and combat overfitting. Additionally, the intermediate layers are equipped with auxiliary classifiers to address vanishing gradients and provide regularization.

In ResNets [7], the innovation of skip connections (or residual connections) allowed direct pathways that bypass one or more layers, enabling the input to be added directly to the block's output. This design facilitates identity mapping, making it easier for deep networks to learn identity functions, thus preventing performance degradation and enabling training of extremely deep networks. Consequently, ResNets with these structures are less prone to vanishing or exploding gradients while allowing them to flow more easily through the network. Instead of learning a direct mapping, the network learns a "residual function", which is also known as the difference between the desired output and the input, and this makes it less challenging to optimize.

Building upon these advancements, DenseNet [1] introduced a novel approach that resulted in highly parameter-efficient, condensed models primarily through feature reuse. Its core idea, dense connectivity, dictates that each layer receives feature maps from all preceding layers in a feed-forward manner and passes its own feature maps to all subsequent layers within the same dense block. This dense connectivity ensures that every layer has direct access to the gradients from the loss function, effectively alleviating the vanishing gradient problem. Moreover, this extensive feature reuse and maximized information flow contribute significantly to achieving higher performance with fewer parameters. The number of new feature maps produced by each layer is controlled by a hyperparameter known as the growth rate.

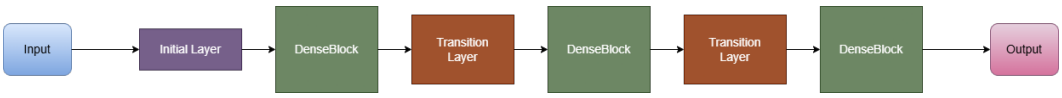


Figure 1: The structure of the DenseNet architecture

3 Method

3.1 Densely Connected Convolutional Network (DenseNet)

The DenseNet architecture connects all layers inside a *dense block*, from a layer, there are connections to all its previous layers, demonstrated comprehensively in Figure 2. Each layer implements a non-linear transformation $H_\ell(.)$ where ℓ indicates the layer. Let $\mathbf{x}_\ell$ be the output of the ℓ^{th} layer. Then $H_\ell(.)$ can be a composite function of operations such as Batch Normalization (BN) [10], rectified linear units (ReLU) [11], pooling, or Convolution [12]. The feature maps of all preceding layers are passed into layer ℓ^{th} by this equation:

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]) \tag{1}$$

where $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]$ indicates the concatenation of the outputs from all layers preceding the ℓ^{th} layer. For DenseNet, $H_\ell(.)$ is a composite function of three consecutive operations: batch normalization (BN), followed by a rectified linear unit (ReLU), and a 3 × 3 convolution (Conv). In our structure, we also implement *transition layers*, those down-sampling layers help change the size of feature maps. Transition layers consist of a convolutional layer and a pooling layer, and they lie between the *dense blocks* to connect them. With *dense blocks* and *transition layers*, the complete structure of DenseNet is revealed in Figure 1. Another definition is the growth rate of the network. If each function $H_\ell(.)$ produces k feature maps, then the ℓ^{th} layer has $k_0 + k \times (\ell - 1)$ feature maps, where k_0 is the number of channels in the input layer. The feature maps can be interpreted as the network's global state. Each layer contributes k new feature maps to this state. The growth rate controls the amount of new information each layer adds to the global state. Once updated, this global state becomes accessible throughout the entire network. Unlike traditional architectures, there's no need to duplicate this information across layers.

3.2 Bottleneck Layers and Compression

In this DenseNet architecture, we also implement a structure known as a Bottleneck Layer, which contains a 1 x 1 convolution before each 3 x 3 convolution to improve computational efficiency. The full sequence of operations is BN-ReLU-Conv(1 × 1)-BN-ReLU-Conv(3 × 3) [1]. Bottleneck Layers can also be amalgamated with a SE Block [4] (Figure 5), or the Drop Channel technique [3] (Figure 4). Each 1 x 1 convolution in this version will produce $4k$ feature maps. Because of this, we also applied another technique called the compression factor θ in *transition layers*. If a *dense block* contains m feature maps, then the *transition layer* will generate $\lfloor \theta m \rfloor$ output feature maps. In our experiments, we set the compression factor θ to 0.5, following the practice from [1].

3.3 Memory Efficient Implementation

To mitigate the high memory demands of DenseNet architectures, we adopt a memory-efficient strategy that reuses pre-allocated memory blocks for intermediate computations. This approach takes advantage of the fact that operations like feature concatenation and batch normalization are relatively inexpensive in terms of computation time but are responsible for a significant portion of the memory footprint. The tradeoff in this method is higher training time (around 20%), but memory consumption is reduced drastically.

In our implementation, we allocate two shared memory buffers: one for concatenation and the other for batch normalization [2]. During the forward pass, all intermediate results are written into these buffers, overwriting previous data as needed. Consequently, these feature maps are not retained permanently and are recomputed very quickly during the backward pass. Since we are implementing this on the PyTorch framework, the shared storage for gradients is an inherent component. Thanks

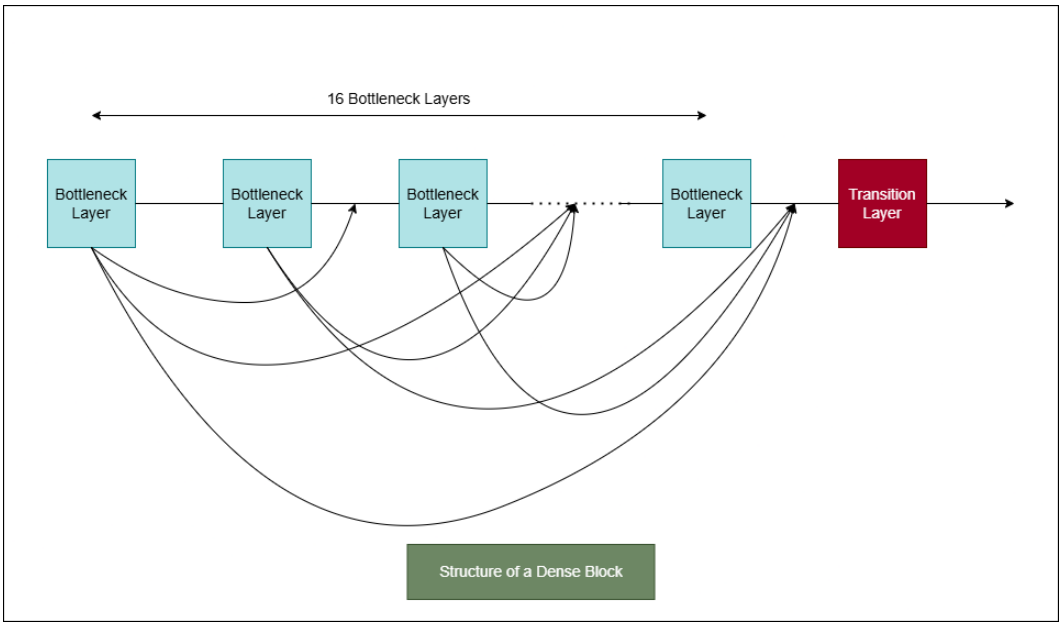


Figure 2: The structure of a Dense Block

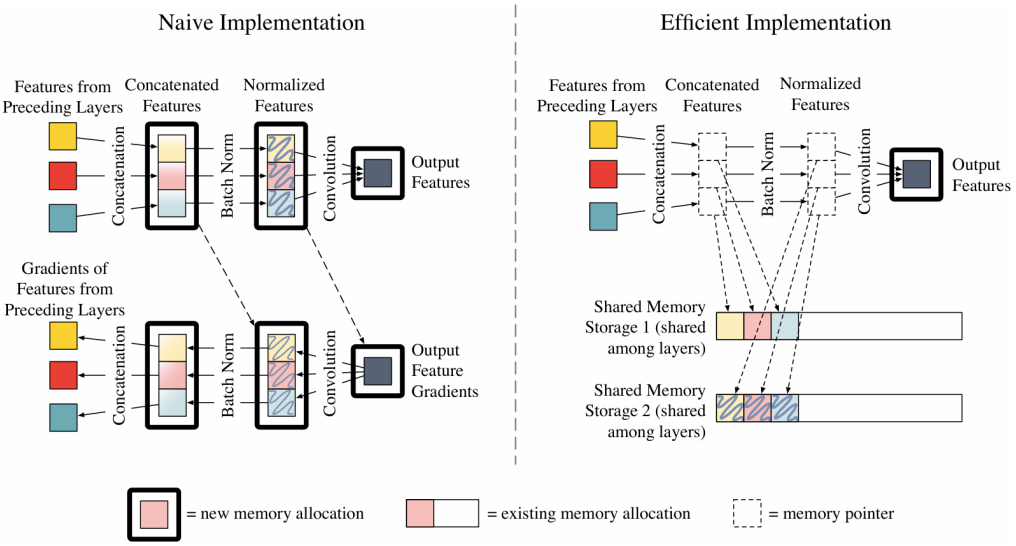


Figure 3: Naive DenseNet implementation and Memory Efficient DenseNet (adapted from [2])

to this, once the shared memory storage contains the correct data, we can perform regular back-propagation to compute gradients. This design allows us to train deep DenseNet models with memory usage that scales linearly with network depth, without compromising computational efficiency. Figure 3 describes detailedly about how this implementation actually works under the hood.

3.4 Drop Channel

According to [3], traditional dropout methods used in [1] are less effective for CNNs due to high spatial correlation within channels and conflicts with Batch Normalization [10]. The random deactivation in dropout disrupts BN's mean and variance stabilization, causing a variance shift that leads to unstable predictions during inference. The failure of dropout operations is attributed to their incorrect placement in the building blocks. The disharmony between the dropout operations and batch

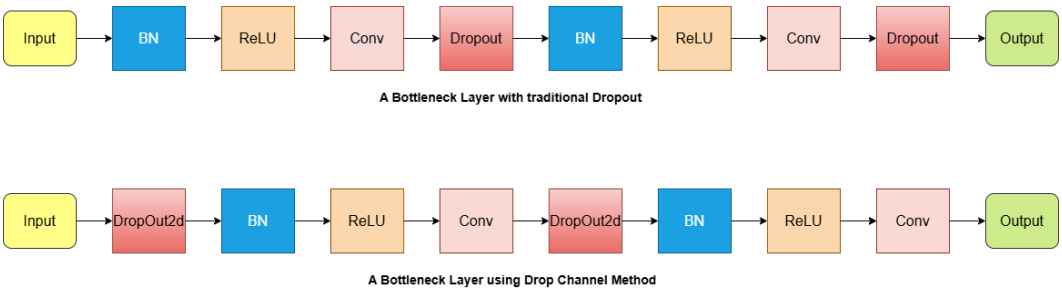


Figure 4: DenseNet using traditional Dropout and Drop Channel

normalization layers is also specified in the inconsistency of BN's variance estimation from training to inference when the BN layer coupled with the neuron-level dropout.

To alleviate the conflict with BN, dropout operations are placed before the convolutional layer rather than between convolution and BN. Furthermore, we adopt a dropout variant from the authors of [3], Drop Channel, which applies dropout at the convolutional level instead of the neuron level, effectively regularizing the network by randomly dropping entire convolutional filters during training. Drop Channel modifies the frequently observed Bottleneck Layer, as shown in Figure 4. This implementation can offer improved performance by aligning better with the structural characteristics of CNNs.

3.5 Squeeze and Excitation Block

Traditional CNNs process each channel of a feature map independently, potentially overlooking the relationships between channels. To address this, paper [4] introduces a new concept called the Squeeze and Excitation Block (SE Block). The SE Block solves this problem by enabling the network to perform dynamic channel-wise feature recalibration, allowing it to emphasize informative features and suppress less useful ones. Early layers learn class-agnostic feature importance, while deeper layers become class-specific, accumulating benefits throughout the network. The SE Block structure is made of:

- Squeeze: Global average pooling aggregates spatial information into a channel descriptor, producing a channel descriptor that captures global spatial information.
- Two fully connected (FC) layers with a reduction ratio r (default is set to 16) and non-linearities (ReLU, sigmoid) model channel-wise dependencies, producing scaling factors to recalibrate feature maps.
- Scale: Multiplies the original feature map by the generated weights, effectively recalibrating the feature responses.

SE Blocks can be seamlessly integrated into CNN structures such as Inception [9], ResNet [7], etc., leading to significant performance improvements across various computer vision tasks. By adding the SE Block [4] at the end of a Bottleneck Layer (displayed in Figure 5), we implement the technique in no time. Seeing that this technique can be useful, we decided to adapt it into our DenseNet architecture [1] to achieve better training results with ease.

3.6 Implementation Details

Based on [1], we adapted the structure of the DenseNet-BC with multiple different versions to combine distinctive methods. The highest and most abstract structure is illustrated in Figure 1. All the code is written in Python, using PyTorch as the main library. The depth is set to be 100, the compression factor $\theta = 0.5$, and the growth rate $k = 12$ [2]. The very first layer before the *dense blocks* is a convolution with twice the growth rate in output channels, which is applied to the input images. There are a total of three *dense blocks* and two *transition layers* between them. Each Bottleneck layer is implemented with the structure BN-ReLU-Conv(1×1)-BN-ReLU-Conv(3×3). If SE Blocks are integrated, then the structure becomes BN-ReLU-Conv(1×1)-BN-ReLU-Conv(3×3)-SE [4]. If Drop Channel is applied, then Dropout2d is used with a drop rate = 0.2 before the BN

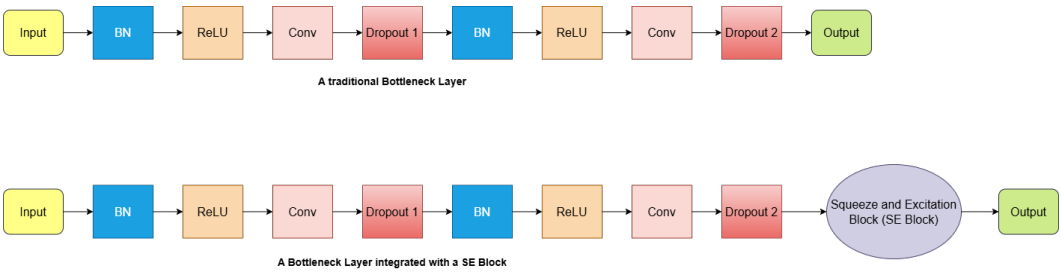


Figure 5: DenseNet with and without Squeeze and Excitation Block

layer [3]. Otherwise, we keep the vanilla structure and use traditional Dropout instead. Each *dense block* concatenates 16 Bottleneck layers via the function available in PyTorch library. If the model is implemented with the Memory Efficient approach, then we simply replace the concatenation method with checkpointing.

On the other hand, following the architecture from [1], each transition layer comprises a 1×1 convolutional layer followed by a 2×2 average pooling layer. The input to the transition layer is multiplied by the compression factor. At the end of the network, a global average pooling is performed, the network is flattened, and then a softmax classifier is attached with a suitable number of classes. We only have the resources to train a specific type of DenseNet-BC with a depth of 100 and $k = 12$ from scratch instead of using a pre-trained model, but we believe the results are consistent and can be extended to much larger and deeper network structures.

4 Experiments

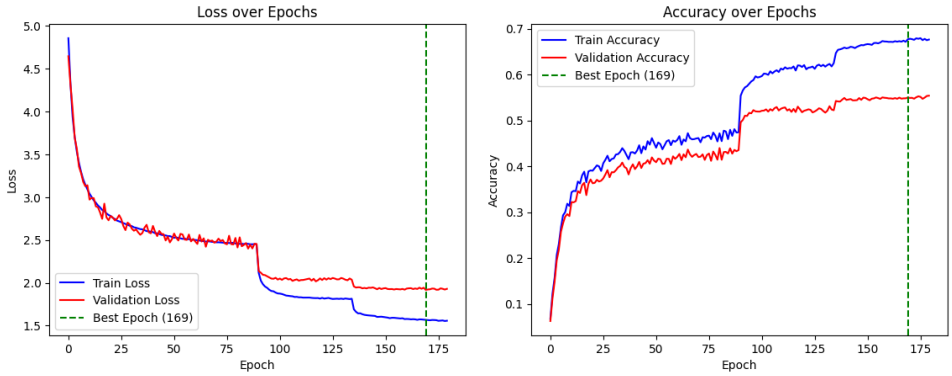
4.1 Datasets

CIFAR100. The CIFAR100 dataset consists of colored natural images with 32×32 pixels from 100 classes. The training and test sets contain 50,000 and 10,000 images, respectively, and we hold out 5,000 training images as a validation set. For data augmentation, we pad the images with 4 pixels and randomly crop them, and then randomly flip them horizontally. For preprocessing, we normalize the data using the channel means and standard deviations. Data augmentation is only applied to train set to avoid information leakage during training process, while data preprocessing is applied to validation set and test set.

Tiny ImageNet. The Tiny ImageNet dataset contains 100000 images of 200 classes (500 for each class) downsized to 64×64 colored images. Each class has 500 training images, 50 validation images and 50 test images. However, because this dataset was initially designed for competitions, its test set has no labels. Therefore, we adapt by splitting 10,000 images from training set as test set, and only train the model on the remaining 90,000 images. The validation set is kept intact. For data augmentation, we first resize the images to size 32 x 32 pixels, then pad the images with 4 pixels and randomly crop them, and randomly flip them horizontally afterwards. For preprocessing, we normalize the data using the channel means and standard deviations. Data augmentation is only applied to train set to avoid information leakage during training process, while data preprocessing is applied to validation set and test set.

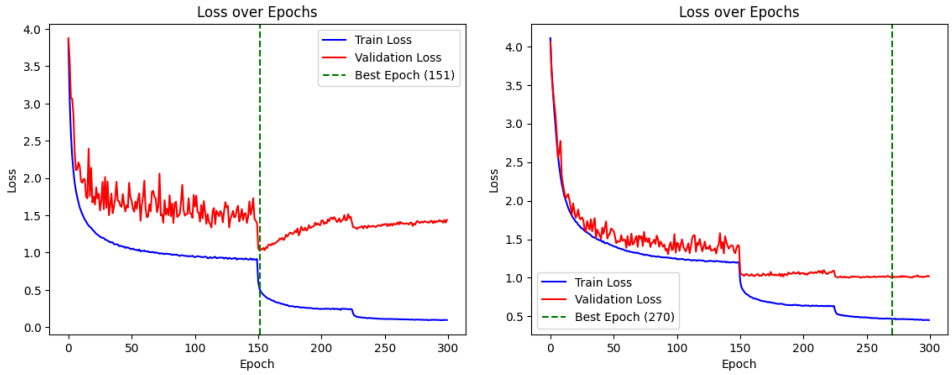
4.2 Training

The network is trained using Stochastic Gradient Descent (SGD). On CIFAR100 and Tiny ImageNet, we train the model using batch size 64 for 300 and 180 epochs, respectively. The initial learning rate is set to 0.1, and is divided by 10 at 50% and 75% of the total number of training epochs using MultiStepLR. We also use a weight decay of 10^{-4} and a Nesterov momentum of 0.9 without dampening. For each dataset, we train a total of 5 different combinations of the network: The Vanilla DenseNet, DenseNet with Drop Channel, DenseNet with SE Blocks, DenseNet with both Drop Channel and SE Blocks, and finally Memory Efficient DenseNet with Drop Channel and SE Blocks. Instead of early stopping, we evaluate the model with the least validation loss using the test set.



(a) Loss of the variant (b) Accuracy of the variant

Figure 6: Loss and Accuracy of best variant in training Tiny ImageNet dataset



(a) Loss of an overfitted Vanilla DenseNet model (b) Loss of a smoother DenseNet variant

Figure 7: Loss of Vanilla DenseNet and Best DenseNet variant in training CIFAR100 dataset

4.3 Classification Results

Let A = Drop Channel, B = SE Blocks, C = Memory Efficient, then we declare some notations:

- Vanilla: The original, unmodified DenseNet.
- A: DenseNet with Drop Channel.
- B: DenseNet with SE Blocks.
- A + B: DenseNet with both Drop Channel and SE Blocks.
- A + B + C: Memory Efficient DenseNet with Drop Channel and SE Blocks.

4.3.1 CIFAR100 dataset

Table 1 presents the results from training on the CIFAR100 dataset, performed using a single NVIDIA RTX 3060 GPU. For each model variant, we recorded the accuracy, peak memory usage, and training time.

The DenseNet variant incorporating Drop Channel and SE Blocks achieved the highest accuracy of 74.97%. This combined method boosted the accuracy by approximately 2.3% (compared to the Vanilla DenseNet). Conversely, the DenseNet variant employing the Memory Efficient technique achieved a slightly lower accuracy of 74.14%, but notably demonstrated approximately half the memory usage (compared to the Drop Channel + SE Blocks variant). This highlights the effectiveness of the memory-efficient approach in regulating consumption, albeit with a longer training time of 13 hours and 26 minutes. The Vanilla DenseNet model exhibited the shortest training time, requiring only 8 hours and 18 minutes. From Figure 6, the Vanilla DenseNet model is prone to overfitting,

Table 1: Comparison of performance metrics across model variants for CIFAR100 dataset.

	Vanilla	A	B	A + B	A + B + C
Accuracy (%)	72.68	74.34	73.48	**74.97**	74.14
Peak Memory (MB)	1222	1115	1229	1164	**505**
Training Time	**8h18m**	10h	8h36m	10h35m	**13h26m**

Table 2: Comparison of performance metrics across model variants for Tiny ImageNet dataset.

	Vanilla	A	B	A + B	A + B + C
Accuracy (%)	52.08	53.74	53.65	**54.39**	**54.50**
Peak Memory (MB)	1200	1114	1230	2144	**505**
Training Time	**5h43m**	5h51m	6h47m	9h53m	**13h35m**

while the DenseNet with Drop Channel and SE Blocks variant handles that problem pretty well, leading to much better result.

Analyzing individual methods, integrating only Drop Channel resulted in a 1.66% accuracy increase, while solely applying SE Blocks improved accuracy by 0.8%. These consistent outcomes further support the conclusion that the applied methods effectively enhance the model's performance, with their combination yielding superior results.

4.3.2 Tiny ImageNet dataset

Table 1 presents the results from training on the CIFAR100 dataset. Training was primarily conducted using a single NVIDIA L4 GPU. However, due to resource constraints, variants A + B and A + B + C of the model were trained on a single NVIDIA RTX 3060 GPU. While this change led to longer overall training times, the consistency of the figures allows for drawing valid conclusions about the method's effectiveness.

The Memory Efficient DenseNet with Drop Channel and SE Blocks achieved the highest accuracy of 54.5%, a 2.42% improvement over its Vanilla version. Crucially, it also consumed the least memory, at only 505 MB. This result suggests that implementing this technique enables the training of much deeper networks from scratch on mid-range GPUs, thereby improving GPU utilization at the cost of extended training times. In From Figure 7, we demonstrated the loss and accuracy of this method.

The DenseNet with Drop Channel and SE Blocks (without memory efficiency) also showed impressive accuracy at 54.39%, only 0.11% lower than the memory-efficient variant, despite its significantly higher memory consumption. In terms of training duration, the Vanilla DenseNet remained the fastest, completing training in just 5 hours and 43 minutes. These outcomes reinforce the efficacy of this implementation, indicating its potential for achieving promising results when applied to deeper network architectures.

5 Conclusion

In this report, we propose a novel approach to the traditional image classification problem by combining multiple established techniques with the DenseNet architecture. Rather than relying on a pre-trained model, we build our network from scratch and incorporate three key methods: Drop-Channel regularization, which enhances generalization; Squeeze-and-Excitation (SE) blocks, which improve feature recalibration by focusing on the most informative channels; and memory-efficient training via checkpointing, which enables the construction of deeper, more capable networks within GPU memory constraints.

Our experiments on the CIFAR-100 and Tiny ImageNet datasets yielded promising results, demonstrating consistent improvements over the baseline DenseNet. These findings confirm that integrating these proven techniques can lead to meaningful performance gains, not merely as a theoretical exercise but as a practical enhancement. Furthermore, this approach lays the groundwork for tackling more complex datasets, such as ImageNet ILSVRC-2012.

By leveraging this hybrid design, we are able to explore deeper DenseNet architectures with higher growth rates while effectively managing computational demands. This work is not only aimed at improving classification accuracy but also at enhancing the memory efficiency and practicality of CNNs in real-world applications. We are optimistic about the broader applicability of this approach and believe that combining well-established methods can further amplify the strengths of DenseNet.

References

[1] Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 4700–4708).

[2] Pleiss, G., Chen, D., Huang, G. & Weinberger, K. Q. (2017). Memory-Efficient Implementation of DenseNets. In *Advances in Neural Information Processing Systems (NeurIPS)* (pp. 3924–3933).

[3] Cai, S., Zhang, M. & Lu, S. (2019). Effective and Efficient Dropout for Deep Convolutional Neural Networks. *arXiv preprint arXiv:1904.03392*.

[4] Hu, J., Shen, L. & Sun, G. (2018). Squeeze-and-Excitation Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7132–7141).

[5] LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

[6] Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)* (pp. 1097–1105).

[7] He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778).

[8] Simonyan, K. & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*.

[9] Szegedy, C., Liu, W., Jia, Y., et al. (2015). Going Deeper with Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1–9).

[10] Ioffe, S. & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)* (pp. 448–456).

[11] Glorot, X., Bordes, A. & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)* (pp. 315–323).

[12] LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.