# CNNs for Image Classification

## A. Introduction

Image classification is a fundamental task in computer vision, with applications ranging from medical diagnosis to autonomous driving. Convolutional Neural Networks (CNNs) have revolutionized this field by enabling the automated recognition of objects, scenes, and patterns within images. This study presented the work on utilizing CNNs for image classification, summarizing approaches, findings, and contributions in this domain.

### A.1. Problem Statement

CNNs have emerged as the state-of-the-art technique for image classification tasks, thanks to their capacity to capture hierarchical features through convolutional layers and pooling operations. The primary aim of this project is to develop CNN models capable of accurately classifying images into their respective categories. To achieve this objective, the study makes use of CIFAR-100 which is widely recognized as standard benchmarks for training and evaluating image classification algorithms. By utilizing the datasets, we aim to assess the performance and effectiveness of the specified CNN networks in achieving accurate image classification.

### A.2. Comparison with Leading Competitors

The project builds upon the latest advancements in CNN architectures, training strategies, and data augmentation techniques. This study conducted an extensive review of leading competitors and their methods. Notably, it mainly compared three models including VGG, ResNet, and DenseNet, to assess its performance in terms of accuracy, computational efficiency, and generalization.

The findings demonstrate that CNN-based image classification model achieves competitive accuracy rates and outperforms some existing models in certain scenarios. The project fine-tuned the model's hyper-parameters and increasing the number of layers to make it adaptable to specific image classification tasks. The study also showcases the importance of data pre-processing, augmentation, and batch normalization techniques in enhancing the model's performance. Finally, the findings demonstrates that ResNet101 and DenseNet201 excel in image classification on the CIFAR-100 dataset, achieving impressive accuracy rates of 87.1% and 85.6%, respectively.

## B. Method Description

### B.1. The Basic Architecture of CNN

CNNs are designed to automatically learn and extract hierarchical features from images, making them highly effective for various computer vision tasks. A basic structure of CNN includes input layers, convolutional layers, pooling layers, activation functions, and fully connected(FC) layers, as showing Fig. 1

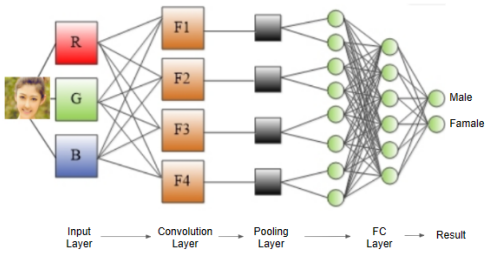



Figure 1. Basic Structure of CNNs

#### B.1.1 Convolutional Layers

CNNs consist of multiple convolutional layers that serve as feature extractors. In each convolutional layer, a set of learnable filters (also known as kernels) slides over the input image, performing element-wise multiplications and summing up the results to create feature maps. These filters capture low-level features like edges, textures, and patterns.

#### B.1.2 Pooling layers

Pooling layers follow convolutional layers and reduce the spatial dimensions of the feature maps. Max-pooling and average-pooling are commonly used operations that downsample the feature maps, retaining the most important information.

#### B.1.3 Activation Functions

Activation functions like Rectified Linear Unit (ReLU) introduce non-linearity into the model, enabling it to learn complex patterns. ReLU is commonly used due to its simplicity and effectiveness in mitigating the vanishing gradient problem.

#### B.1.4 Fully Connected Layers

After several convolutional and pooling layers, fully connected layers are used for classification. These layers take the high-level features extracted by previous layers and produce class probabilities through a softmax activation.

#### B.1.5 Batch Normalization (BN) Layers

BN is a normalization technique applied to intermediate feature maps within a neural network [4]. BN operates on mini-batches of data within each layer of a neural network.

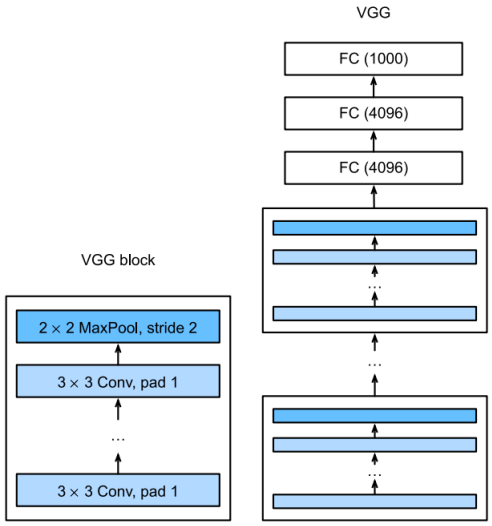



Figure 2. VGG Block and VGG Net

For each mini-batch, BN computes the mean and variance of the feature maps. It then normalizes the feature maps by subtracting the mean and dividing by the standard deviation. Additionally, BN introduces learnable scale and shift parameters that allow the model to adapt the normalized values if needed.

## B.2. VGG

Smola and Narayanamurthy(2014) proposed a "block" structure for building deep neural networks. In this architecture, one VGG block comprises a sequence of convolutional layers followed by a max pooling layer, which serves to downsample the spatial dimensions [5].

The block begins with one or more convolutional layers that use 3x3 kernels with padding set to 1. This padding ensures that the height and width of the feature maps remain the same, preserving spatial information.

A nonlinearity, often ReLU, is applied after each convolutional layer to introduce non-linearity into the network and enhance its capacity to learn complex features.

Following the convolutional layers, a max pooling layer with a 2x2 kernel and a stride of 2 is employed. This operation reduces the spatial resolution by a factor of 2, effectively halving the height and width of the feature maps.

This "block" structure, characterized by repeated VGG blocks in the network architecture, as Fig. 2, helps in creating deep and effective neural networks for tasks such as image classification. It ensures that important spatial details are preserved through the use of padding and allows for efficient downsampling through max pooling, making it a significant contribution to the field of deep learning.

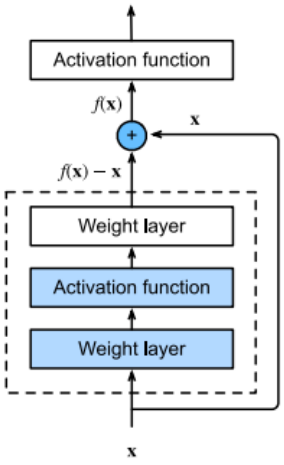



Figure 3. Residual Block

## B.3. ResNet

He, Zhang, Ren, and Sun (2016) introduced Residual Networks (ResNet), which is based on the concept of learning residuals [1]. The core idea of ResNet is that for a neural network layer or block, it should learn the 'residual' or incremental changes between the input and output, rather than completely re-modeling the target function.

To achieve residual learning, ResNet incorporates residual blocks, as Fig. 3, which include the following components: Identity Mapping [2], where the input is directly passed to the block's output; a series of convolutional layers for learning the residuals between input and output; and a non-linear activation function (typically ReLU) to introduce non-linearity before passing the output of the convolutional layers to the identity mapping output. Skip connections are used to achieve identity mapping by adding the input, x, to the output of the residual block.

Specifically, for each residual block, the input signal is passed through a series of convolutional layers and then added to the output signal. This means that the network can learn the increment from input to output, enabling gradients to propagate more easily to earlier layers and mitigating the vanishing gradient problem. Such a design allows for the stacking of more residual blocks, thereby creating a deeper network. Increasing network depth does not lead to a performance decline; on the contrary, it enhances the network's expressive capacity.

Towards the end of ResNet, a global average pooling layer is employed to reduce the spatial dimensions of the feature maps. This aids in reducing the number of parameters and makes the network invariant to the input image size.

ResNet also introduces a 'bottleneck structure' Fig. 4 that employs 1x1, 3x3, and 1x1 convolutional layers to

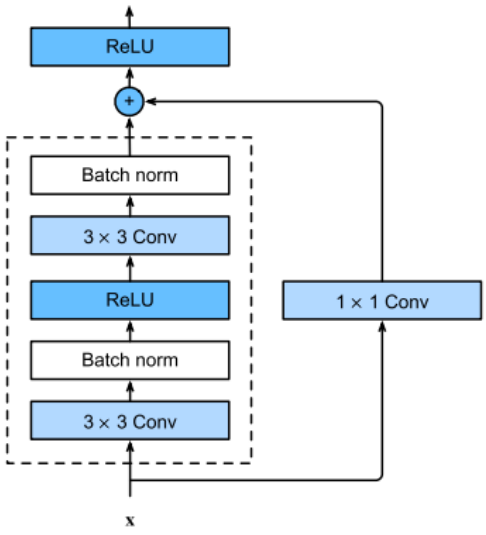



Figure 4. Bottleneck Block

reduce the number of channels while maintaining network depth, effectively reducing computational and memory costs.

### B.4. DenseNet

Huang et al(2016) proposed Densely Connected Convolutional Networks [3]. The core idea behind DenseNet is the introduction of dense connections within neural network layers, which fosters feature reuse and gradient flow. It addresses several key challenges in deep learning, including the vanishing gradient problem and the efficient use of parameters.

The fundamental building block of DenseNet is the "dense block." In a dense block, each layer receives feature maps from all preceding layers within the same block. This connectivity pattern creates densely connected layers, promoting the propagation of information and gradients throughout the network.

Mathematically, DenseNet performs a mapping from input x to its expansion(1).

$$\begin{aligned} \mathbf{x} \rightarrow & [\mathbf{x}, f_1(\mathbf{x}), \\ & f_2\left([\mathbf{x}, f_1(\mathbf{x})]\right), \\ & f_3\left([\mathbf{x}, f_1(\mathbf{x}), f_2\left([\mathbf{x}, f_1(\mathbf{x})]\right)]\right), \ldots] \end{aligned} \tag{1}$$

Finally, these expansions are incorporated into the multilayer perceptron to reduce the number of features. Fig. 5 shows that ResNet (on the left) and DenseNet (on the right) differ primarily in their approach to inter-layer connections: ResNet uses addition, while DenseNet uses concatenation.

DenseNet introduces a hyperparameter "growth rate." This parameter determines how many feature maps each layer in a dense block produces. A higher growth rate leads to more feature maps, increasing the network's capacity. Dense connectivity ensures that each layer has access to these feature maps for building on previously learned features.

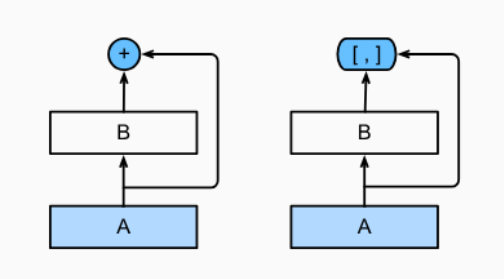



Figure 5. The difference between Residual Block and Dense Block

To control the number of parameters and computational cost, DenseNet includes transition layers between dense blocks. These transition layers incorporate batch normalization, a 1x1 convolution, and average pooling. They reduce the spatial dimensions while preserving important features and information.

## C. Methods Implementation

https://github.com/hahawang1986/deep-Learning.git

assignment2 Ning Ni a1869549.ipynb was the program of this assignment.

.pth files were the result of various models which can be reloaded by .ipynb file to show figures.

## D. Experiments and Analysis

### D.1. Experiments Equipment

This experiment utilizes the NVIDIA GeForce RTX 2080 Ti as the GPU processor, equipped with 4352 CUDA cores and featuring 11GB of GDDR6 video memory, making it well-suited for training complex neural network models.

### D.2. Adjustment of Hyperparameters

#### D.2.1 The Resolution of Image

The input size plays a crucial role in determining the model's perceptual capacity for processing images. Larger input sizes provide the model with the ability to capture finer image details and contextual information, leading to improved performance in classification and detection tasks. Conversely, smaller input sizes may result in information loss, making it more challenging for the model to recognize complex patterns and features within images.

As depicted in Fig. 6 and Fig. 7, using ResNet34 as an example, it is evident that when the image resolution is

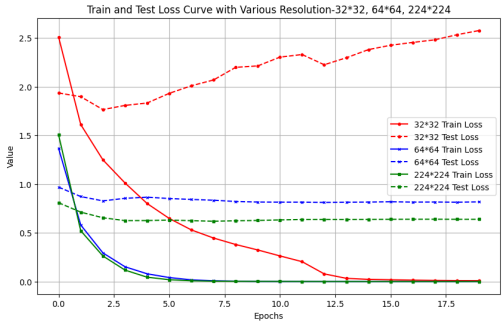



Figure 6. Loss Performance with Various Resolution

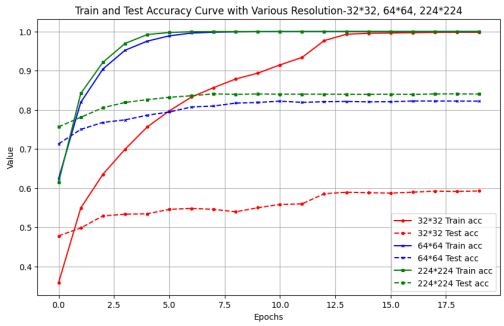



Figure 7. Accuracy Performance with Various Resolution

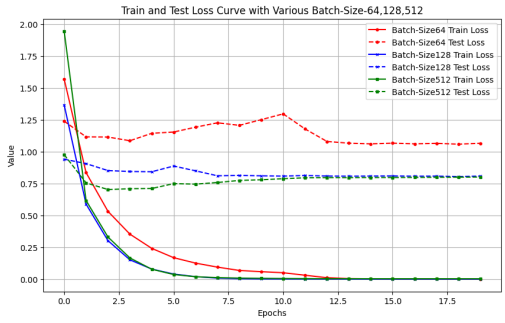



Figure 8. Loss Performance with Various Batch Size

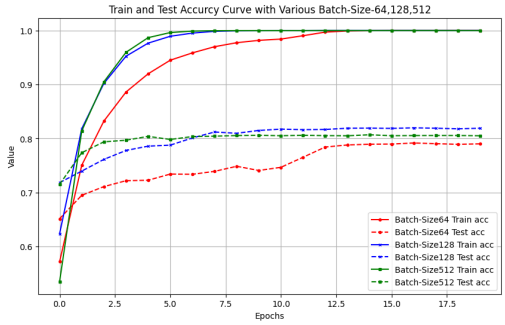



Figure 9. Accuracy Performance with Various Batch Size

higher, the training and testing loss converge more quickly, and the test dataset accuracy increases.

The input size determines the model's perceptual capacity for images. Larger input sizes allow the model to capture more image details and contextual information, which enhances performance in classification and detection tasks. Smaller input sizes can lead to information loss, making it challenging for the model to recognize complex patterns and features in images.

It's important to note that the choice of input size directly influences the computational and memory requirements of a model. Larger input images demand more computational resources and memory, potentially affecting both training and inference speed. Given the specific GPU used in this study, a resolution of 128x128 emerges as a reasonable parameter for this experiment, aligning with the computational capacity of the GPU

### D.2.2 Batch Size

A larger batch size typically accelerates convergence, requiring fewer update steps per epoch and allowing the model to quickly achieve higher training accuracy. The analysis takes ResNet34 as example. This behavior is evident in the training and test loss curves shown in Fig. 8. However, it's important to note that using a large batch size demands more GPU memory, which can be a limiting factor. In some instances, as illustrated in Fig. 9, a batch size of 512 may lead to worse generalization (test accuracy) compared to a batch size of 128, indicating a risk of overfitting.

On the other hand, a smaller batch size consumes less memory, potentially contributing to more stable training and superior generalization. Smaller batches also introduce a form of regularization, mitigating overfitting tendencies. It's worth noting that a smaller batch size results in slower convergence, requiring more updates to process the same data. This can be observed in the case of a batch size of 64, as demonstrated Fig. 8 and Fig. 9.

For this project, a batch size of 128 emerges as a reasonable choice, striking a balance between model performance and memory capacity.

### D.2.3 Optimizer

The SGD optimizer is used, along with learning rate scheduling. Each CNN architecture is trained for 20 epochs, with a learning rate of 0.01 for the first 12 or 13 epochs, followed by a reduced learning rate of 0.001 for the remaining epochs to promote further convergence.

It starts with a higher learning rate to facilitate swift initial progress, enabling the model to escape local minima and converge faster. The subsequent reduction in learning rate in later epochs fine-tunes the model, ensuring more precise convergence and improved generalization. This approach

contributes to both convergence and stability, preventing instability and oscillations in training. Moreover, it adapts the model to the specific characteristics of the training data and efficiently utilizes computational resources.

#### D.2.4 Initial Weight

Pretraining techniques have empowered these models to learn valuable feature representations from large-scale data. These features often encompass high-level abstract characteristics of images, improving their ability to capture image structures and patterns effectively. Additionally, due to the informative nature of the initial parameters, pretrained models typically converge faster compared to models starting from random initialization. This strategy is expected to notably elevate the performance of the CIFAR-100 image classification task, reduce training time, and provide enhanced generalization, enabling the model to better adapt to diverse categories and data characteristics.

This project seeks to leverage models from the PyTorch Model package, specifically VGG, ResNet, and DenseNet, in conjunction with their pretrained weights. Through the conducted experiments, it was observed that these pretrained models exhibit a 40% higher accuracy compared to their counterparts initialized with random weights when trained for the same number of epochs (20 in this project). This noteworthy improvement significantly enhances the training performance for CIFAR-100 image classification.

### D.3. VGG Models Classify Images in CIFAR100

This project utilized three VGG model including VGG11_BN, VGG16_BN, VGG19_BN. Their structures are shown as Fig. 10.

VGG11_BN is a variant of the VGG network with 11 weight layers. It includes 8 convolutional layers and 3 fully connected layers. Batch normalization is applied after each convolutional layer, which helps in training stability and faster convergence. VGG16_BN is an extended version of the VGG architecture with 16 weight layers. It features 13 convolutional layers and 3 fully connected layers. Like VGG11_BN, batch normalization is used throughout the network. VGG19_BN is a more complex model with 19 weight layers, which includes 16 convolutional layers and 3 fully connected layers. Batch normalization is integrated into the network to improve training efficiency.

The experimental results indicate that as the number of layers in the VGG architecture increases, both the test loss and test accuracy exhibit gradual improvement. However, it becomes apparent that when employing VGG19, there is no significant increase in image recognition accuracy, showing in Fig. 11 and Fig. 12. Therefore, for the CIFAR100 dataset, the VGG16 model has already demonstrated a high level of image recognition capability, achieving an impressive accuracy rate of 80.8%, Table 1.

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64 **LRN** | conv3-64 **conv3-64** | conv3-64 conv3-64 | conv3-64 conv3-64 | conv3-64 conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128 **conv3-128** | conv3-128 conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 |
| maxpool | | | | | |
| conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 **conv1-256** | conv3-256 conv3-256 **conv3-256** | conv3-256 conv3-256 conv3-256 **conv3-256** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

Figure 10. Different Structures of VGG [5]

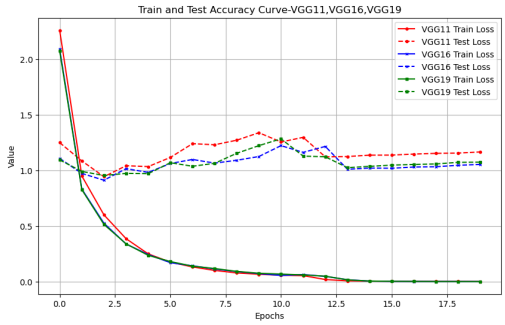



Figure 11. Train and Test Loss Curve - VGG11, VGG16, VGG19

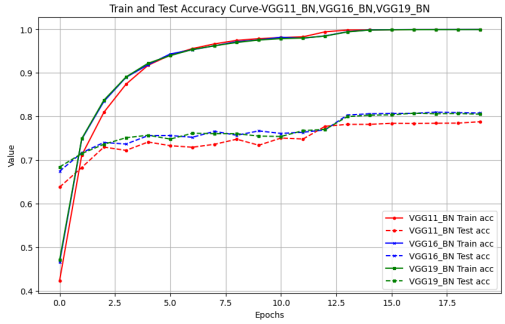


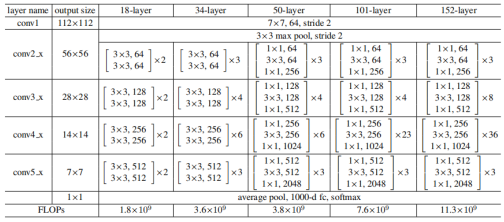

Figure 12. Train and Test Accuracy Curve - VGG11, VGG16, VGG19

### D.4. ResNet Models Classify Images in CIFAR100

ResNet34, ResNet50, and ResNet101 primarily differ in terms of network depth and the number of parameters, as illustrated in Fig. 13. ResNet34 consists of 34 convolutional layers and fully connected layers, primarily composed of

Table 1. Results of Different VGG Models

| model | test_loss | test_accuracy |
|---|---|---|
| VGG11_BN | 1.17 | 78.80% |
| VGG16_BN | 1.06 | 80.80% |
| VGG19_BN | 1.08 | 80.52% |



| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| | | 3×3 max pool, stride 2 | | | | |
| conv2_x | 56×56 | [3×3, 64; 3×3, 64] ×2 | [3×3, 64; 3×3, 64] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 |
| conv3_x | 28×28 | [3×3, 128; 3×3, 128] ×2 | [3×3, 128; 3×3, 128] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×8 |
| conv4_x | 14×14 | [3×3, 256; 3×3, 256] ×2 | [3×3, 256; 3×3, 256] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×23 | [1×1, 256; 3×3, 256; 1×1, 1024] ×36 |
| conv5_x | 7×7 | [3×3, 512; 3×3, 512] ×2 | [3×3, 512; 3×3, 512] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

Figure 13. Different Structures of ResNet [1]

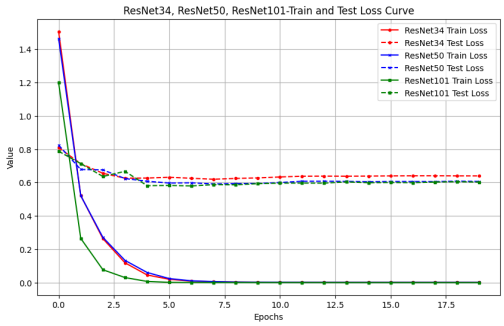



Figure 14. Train and Test Loss Curve - ResNet34, ResNet50, ResNet101

Table 2. Results of Different ResNet Models

| model | test_loss | test_accuracy |
|---|---|---|
| ResNet34 | 0.620 | 84.1% |
| ResNet50 | 0.607 | 85.2% |
| ResNet101 | 0.601 | 87.1% |

residual blocks. It has fewer parameters, making it suitable for situations with limited computational resources. On the other hand, ResNet50 comprises 50 convolutional layers and fully connected layers, introducing bottleneck blocks, and it is deeper. ResNet101 extends the depth even further with 101 convolutional layers and fully connected layers, containing more parameters, thereby requiring greater memory and computational resources.

With an increase in network depth, ResNet exhibits enhanced performance on both training and testing datasets. As depicted in Fig. 14 and Fig. 15, while all three ResNets achieve optimal training loss and accuracy, ResNet101 outshines the others with a test loss of 0.601 and a test accuracy of 87.1%, Table 2. This performance is notably superior to that of ResNet34 and ResNet50.

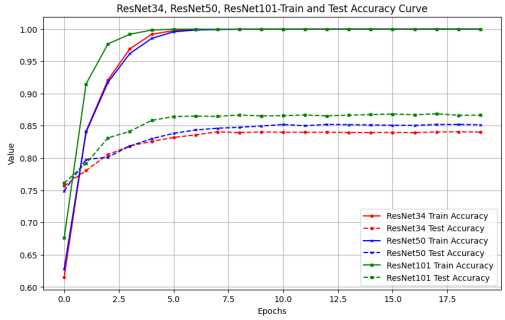



Figure 15. Train and Test Accuracy Curve - ResNet34, ResNet50, ResNet101

| Layers | Output Size | DenseNet-121 | DenseNet-169 | DenseNet-201 | DenseNet-264 |
|---|---|---|---|---|---|
| Convolution | 112 × 112 | 7 × 7 conv, stride 2 | | | |
| Pooling | 56 × 56 | 3 × 3 max pool, stride 2 | | | |
| Dense Block (1) | 56 × 56 | [1 × 1 conv; 3 × 3 conv] × 6 | [1 × 1 conv; 3 × 3 conv] × 6 | [1 × 1 conv; 3 × 3 conv] × 6 | [1 × 1 conv; 3 × 3 conv] × 6 |
| Transition Layer (1) | 56 × 56 | 1 × 1 conv | | | |
| | 28 × 28 | 2 × 2 average pool, stride 2 | | | |
| Dense Block (2) | 28 × 28 | [1 × 1 conv; 3 × 3 conv] × 12 | [1 × 1 conv; 3 × 3 conv] × 12 | [1 × 1 conv; 3 × 3 conv] × 12 | [1 × 1 conv; 3 × 3 conv] × 12 |
| Transition Layer (2) | 28 × 28 | 1 × 1 conv | | | |
| | 14 × 14 | 2 × 2 average pool, stride 2 | | | |
| Dense Block (3) | 14 × 14 | [1 × 1 conv; 3 × 3 conv] × 24 | [1 × 1 conv; 3 × 3 conv] × 32 | [1 × 1 conv; 3 × 3 conv] × 48 | [1 × 1 conv; 3 × 3 conv] × 64 |
| Transition Layer (3) | 14 × 14 | 1 × 1 conv | | | |
| | 7 × 7 | 2 × 2 average pool, stride 2 | | | |
| Dense Block (4) | 7 × 7 | [1 × 1 conv; 3 × 3 conv] × 16 | [1 × 1 conv; 3 × 3 conv] × 32 | [1 × 1 conv; 3 × 3 conv] × 32 | [1 × 1 conv; 3 × 3 conv] × 48 |
| Classification Layer | 1 × 1 | 7 × 7 global average pool | | | |
| | | 1000D fully-connected, softmax | | | |

Figure 16. Different Structures of DenseNet [3]

## D.5. DenseNet Models Classify Images in CIFAR100

DenseNet121, DenseNet169, and DenseNet201 are variants of the DenseNet architecture, and they primarily differ in network depth and model complexity, as depicted in Fig. 16.

DenseNet121, as the name suggests, consists of 121 layers, including densely connected convolutional layers and transition layers. It is a medium-sized model with relatively fewer parameters compared to its counterparts. This characteristic makes it a practical choice for tasks where computational resources are somewhat limited.

DenseNet169, on the other hand, offers increased network depth with 169 layers. It incorporates a greater number of densely connected layers and transition layers, making it more capable of capturing intricate features in data. This additional depth can lead to better performance but may require more memory and computing power.

DenseNet201 represents the deepest option among these models, with a total of 201 layers. It boasts an extensive network with densely connected convolutional layers and transition layers. Consequently, it contains the most parameters, offering high modeling capacity. However, this also demands substantial computational resources and memory.

With an increase in network depth, DenseNet demonstrates enhanced performance on both training and testing datasets. This is evident from the loss and accuracy curves shown in Figures Fig. 17 and Fig. 18.

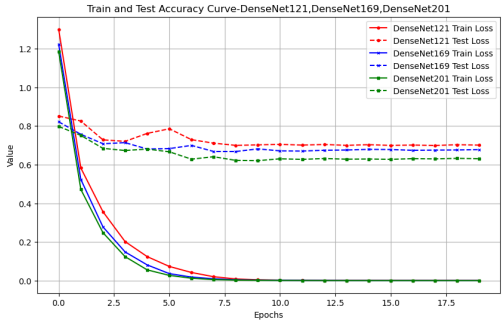



Figure 17. Train and Test Loss Curve - DenseNet121, DenseNet169, DenseNet201

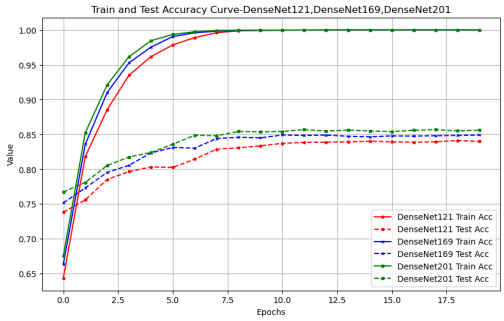



Figure 18. Train and Test Accuracy Curve - DenseNet121, DenseNet169, DenseNet201

Table 3. Results of Different DenseNet Models

| model | test_loss | test_accuracy |
|---|---|---|
| DenseNet121 | 0.701 | 83.99% |
| DenseNet169 | 0.678 | 84.91% |
| DenseNet201 | 0.631 | 85.61% |

DenseNet201 Outshines Others Among the three DenseNet models, DenseNet201 stands out with remarkable performance. It achieves a test loss of 0.631 and an impressive test accuracy of 85.61%, as illustrated in Table 3. These results are notably superior to those of DenseNet121 and DenseNet169.

### D.6. Performance Comparison of VGG, ResNet and DenseNet

In the analysis of the experimental results, as evident from Fig. 19 and Fig. 20, ResNet101 emerges as the frontrunner in terms of accuracy when compared to DenseNet201 and VGG16. Its remarkable performance underscores the profound impact of residual connections in training deep neural networks, making it a standout choice for image classification tasks.

Following closely behind ResNet101, DenseNet201 showcases its prowess in the realm of image classification.

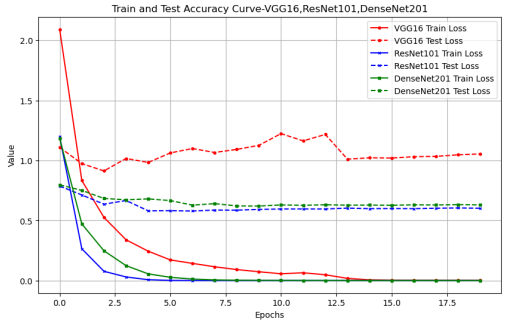



Figure 19. Train and Test Loss Curve -VGG16, ResNet101, DenseNet201

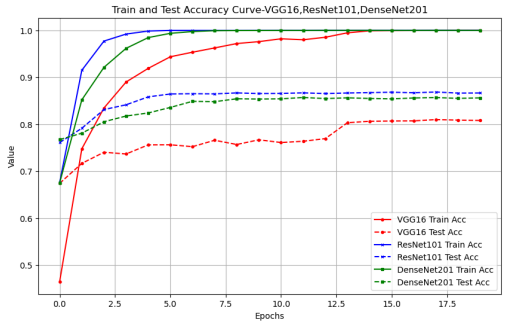



Figure 20. Train and Test Accuracy Curve -VGG16, ResNet101, DenseNet201

The dense connectivity within the network, where each layer is intricately linked to its preceding layers, fosters a robust feature reuse mechanism. This, in turn, contributes significantly to its impressive performance and its ability to learn from the data efficiently.

On the other hand, VGG16, although celebrated for its simplicity and elegance, falls short in terms of performance when pitted against its counterparts, ResNet101 and DenseNet201. The uniformity in its architecture, which may have been its strength in certain contexts, seems to limit its capacity to capture intricate features and patterns compared to the more sophisticated designs of ResNet101 and DenseNet201.

The triumph of ResNet101 can be primarily attributed to its depth and the ingenious use of residual connections. These connections, by mitigating the vanishing gradient problem, facilitate the training of exceptionally deep networks. Consequently, ResNet101 excels in capturing intricate features and patterns within the CIFAR-100 dataset, demonstrating the immense effectiveness of this architecture.

DenseNet201, with its dense connectivity, capitalizes on an elegant approach where each layer collaborates with all preceding layers. This dynamic encourages efficient feature reuse, a vital aspect of its commendable performance. The

proximity of DenseNet201's accuracy to that of ResNet101 reinforces the advantages of dense connectivity when training deep neural networks.

## E. Conclusion

This research has delved into the fascinating realm of image classification, a fundamental task in computer vision with applications spanning various fields. Leveraging Convolutional Neural Networks (CNNs), this research has explored different architectural variants and their performance in image classification, specifically on the CIFAR-100 dataset.

First, this study comprehensively assessed the impact of various hyperparameters, including image resolution, batch size, optimizer choice, and weight initialization. These choices play a pivotal role in training efficiency and model performance. It was observed that a higher image resolution and an appropriately chosen batch size can significantly improve both training and testing performance. Additionally, utilizing pretrained weights provides a substantial boost to model accuracy, allowing for more efficient training and enhanced generalization.

Second, the experiments examined three widely recognized CNN architectures: VGG, ResNet, and DenseNet. Each of these models presents unique characteristics that affect their performance in image classification. The experiments demonstrated that ResNet101 outperformed the other models, owing to its depth and the ingenious use of residual connections. The effectiveness of these connections in mitigating the vanishing gradient problem was evident, enabling ResNet101 to capture intricate features and patterns in the CIFAR-100 dataset. DenseNet201 closely followed ResNet101 in terms of accuracy, emphasizing the advantages of dense connectivity in training deep neural networks.

This research provides a comprehensive understanding of the factors influencing image classification performance and sheds light on the capabilities of different CNN architectures. It serves as a valuable resource for researchers and practitioners in the field of computer vision, offering insights into the design and optimization of CNN models for image classification tasks. Ultimately, the findings presented here contribute to the ongoing evolution of deep learning techniques for image classification and related applications.

## References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2, 6

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016. 2

[3] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 3, 6

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. 1

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2, 5