Dense Convolutional Network (DenseNet) on CIFAR-10 and the CIFAR-100

Yangyu Chen

Abstract

This report explores if Dense Convolutional Networks (DenseNet) improve performance on image classification tasks with CIFAR-10 and CIFAR-100. DenseNet achieves high accuracy and low computational expense through dense connectivity and the use of optimization techniques, surpassing Vision Transformers (ViT) on smaller datasets. These results can inform model selection for image classification tasks under limited computational resources. DenseNet is applied to the CIFAR-10 dataset using deep learning and neural networks.

Keywords: deep learning; **DenseNet**; CIFAR-10; neural networks; Vision transformer.

Introduction

GitHub link:

The computer vision curriculum is a major field of deep learning, and Convolutional Neural Networks (CNN) have been instrumental in achieving state-of-the-art performance for a range of computer vision tasks, including image classification, object detection, image segmentation, etc. One of these architectures, Dense Convolutional Networks (DenseNet) stand out due to their unique design, which promotes feature reuse and alleviates the vanishing gradient problem through dense connectivity. This study examines the performance of DenseNet on the CIFAR-10 and CIFAR-100 datasets, with the following three main aims:

1. In this study, the goal is to modify DenseNet in order to be optimized for small datasets by redesigning the architecture, loss function, and training process.
2. A little experimental study to see the differences and compare with Vision Transformers (ViT) to quickly analyze what best suits and where to use.

3. To evaluate the computational efficiency and scalability of DenseNet for realistic context performance.

The report shows the strengths of DenseNet design elements, including bottleneck and transition layers, and proves its application to resource-poor settings. In addition, this work serves as a reference for future exploration of hybrid architectures and transformer-based networks for low spatial scale image classification.

Related Work

CNNs have transformed the field of computer vision (CV). In CV tasks like image classification, object detection, and semantic segmentation, CNNs achieved the state of the art before the emergence of ViT and other transformer-based methods. Among these architectures, Dense Convolutional Networks (DenseNets) [1] have garnered much interest for their design of encouraging feature reuse as well as enhancing gradient propagation. It chronicles the application of DenseNet on the CIFAR-10 and CIFAR-100 datasets and elucidates the architecture's performance alongside the methods used in the training and evaluation processes.

Fully-Connected Neural Networks (CNNS)

Configured for grid-like data (images) processing via convolution, CNNs are a special kind of neural network. They are made up of convolutional layers that apply learnable filters across the input space to identify features and spatial dependencies in data. Traditional CNNs face vanishing gradient issues with network depth, leading to difficulty in effectively training a very deep network. The invention of residual blocks (ResNet) addresses the problem, making it possible to train very deep neural networks. DenseNet borrows this idea.

Methodology

DenseNet Architecture

DenseNets extend to the point where every layer is connected directly to every other layer in a feed-forward manner. This design allows features learned at each layer to be accessible by all layers, which encourages feature reuse and prevents the vanishing gradient problem. Also, the bottleneck layers in DenseNets reduce the dimensionality of the feature map, so they are computationally efficient.

DenseNet Design Principles

Key Components

Bottleneck Layers

This will be introduced in bottleneck layers. The 1×1 convolution reduces the input channels, and the 3×3 convolution captures features. This is a process through which feature extraction is done.

Transition Layers

In the transition layer, the feature maps are downsampled, and the number of channels is reduced. We use a batch normalization operation, a 1×1 convolution, and an average pooling operation.

Vision Transformer

Vision Transformer [4](ViT) is a deep learning model based on the Transformer architecture, primarily used for image recognition and other computer vision tasks. Unlike traditional Convolutional Neural Networks (CNNs), ViT treats images as serialized inputs and leverages self-attention mechanisms to process relationships between pixels in the image. ViT first divides the input image into fixed-size patches, such as 16x16 pixels, and then flattens each patch into a one-dimensional vector. These vectors are mapped into a high-dimensional space through a linear projection layer, forming the embedded representation of the image patches. Since the

Transformer model itself does not have the ability to capture sequence order, ViT adds positional embeddings to each image patch to maintain spatial structure information. The core of ViT is multiple Transformer encoder layers, each of which includes multi-head self-attention and a feed-forward neural network. These encoder layers process the embedded vectors of the image patches, calculate global dependencies between image patches, and gradually extract global features. ViT introduces a special classification token (CLS Token) for the final classification task. The CLS Token is input into the Transformer along with the embedded vectors of other image patches and represents the global features of the entire image in the final output. Finally, the CLS Token is sent to a fully connected layer for classification. After processing, the CLS Token output from ViT is passed through a fully connected layer to generate the final classification results. ViT's advantage lies in its ability to capture the global context of the entire image, which is very helpful for fine-grained segmentation tasks. It can effectively distinguish between visually similar but semantically different objects, improving segmentation accuracy. ViT also performs well in image generation and reconstruction tasks, where its self-attention mechanism can model globally and generate more realistic image content. Overall, ViT achieves feature extraction and classification of images by dividing images into small patches, treating them as serialized tokens, and inputting them into the Transformer encoder, a process similar to the Transformer model used for processing text sequences in natural language processing.

Experiments

DATA Preprocessing

Before training the model, to improve the network's performance. the images were normalized. This step normalizes the pixel values to have selected mean and unit variance. In deep learning, appropriate preprocessing of image data is crucial as it helps the model learn and generalize better. In this code, preprocessing steps include converting images to PIL images, applying random horizontal flipping, random rotation, converting to Tensors, and normalization. For testing data, the preprocessing steps are slightly simpler, including conversion to PIL images, conversion to Tensors, and normalization.Converting to PIL images is the first step in preprocessing because PyTorch's ToPILImage() transformation can convert image data from NumPy array format to PIL image format. PIL (Python Imaging Library) is an image processing library that provides a wealth of image manipulation features, facilitating subsequent processing.Random horizontal flipping and random rotation are data augmentation techniques that increase the diversity of the data by randomly flipping and rotating images. The benefit of this is to prevent the model from becoming overly dependent on specific directions or positions in the original dataset, thereby improving the model's generalization ability. For example, random horizontal flipping can ensure that the model is not sensitive to the left-right direction of the image, while random rotation can ensure that the model has some robustness to different orientations of the image.Converting to Tensors involves transforming PIL images into PyTorch's Tensor format because PyTorch models can only process data in Tensor format. This conversion step is necessary as it allows the image data to be read and processed by the model.Normalization scales the image data to a specific range (usually between 0 and 1) by subtracting the mean and dividing by the standard deviation. In this code, the mean and standard deviation for each color channel are set to 0.5, which means that each pixel value of the image will be scaled to between -1 and 1. Normalization helps to speed up the convergence of the model during the training process because it ensures that different input features have the same scale, allowing the gradient descent algorithm to work more effectively.

Model Initialization

Optimization Strategy

The Adam[2] optimization algorithm was used. The Adam optimization algorithm is an efficient optimization method used in deep learning, combining the advantages of Momentum and RMSprop algorithms. It adaptively adjusts the learning rate for each parameter by computing the first moment (mean) and second moment (uncentered variance) of the gradients. This approach is not only capable of dealing with sparse gradient issues but also suitable for large-scale data and parameter volume problems, as well as non-stationary objectives and very noisy or sparse gradient issues.The key advantage of the Adam algorithm lies in its adaptability, as it independently adjusts the learning rate for each parameter, making the algorithm more flexible when facing parameters with different gradient magnitudes. Additionally, Adam introduces a bias correction mechanism to address initialization bias issues, ensuring faster convergence in the early stages of training. Due to its low memory requirements and robustness to parameters, Adam has become one of the preferred optimizers in deep learning, especially suitable for complex non-convex optimization problems.

Training Process

The training phase involved feeding batches of images through the network, computing the loss based on the predicted output and the true label, and then adjusting the network weights via backpropagation. The training process was repeated for several epochs until the network reached a satisfactory level of accuracy on the training data.

Evaluation Metrics

To assess the performance of the trained DenseNet model, the accuracy metric was used. Accuracy measures the proportion of correctly classified images out of the total number of images tested. The model was evaluated on a test set which was not seen during the training phase to provide an unbiased estimate of the model's generalization capabilities.

Datasets

The CIFAR-10 dataset is widely used. It consists of 60,000 32×32 color images in 10 classes, with 6,000 images per class. And the Humpback Whale Identification dataset contains thousands of images of humpback whale flukes. Individual whales have been identified by researchers and given an Id.

The CIFAR-100 dataset consists of 60,000 32×32 color images, divided into 100 classes. Each class in CIFAR-100 contains 600 images, making for a total of 60,000 images per class, just like in CIFAR-10. The major difference between CIFAR-10 and CIFAR-100 is the number of classes; CIFAR-100 has 10 times as many classes, but with fewer images per class. The images in CIFAR-100 are grouped into 20 superclasses, with each superclass containing 5 classes that share some visual or semantic relationship

Results

After training the DenseNet model on the CIFAR-10 dataset, the network demonstrated high accuracy on both the training and test sets. The model achieved a test accuracy of 89% after 3 epochs, indicating that the dense connectivity pattern facilitated effective learning and generalization. DenseNet has also demonstrated promising results on the CIFAR-100 dataset. The loss and accuracy plots of DenseNet are shown below.

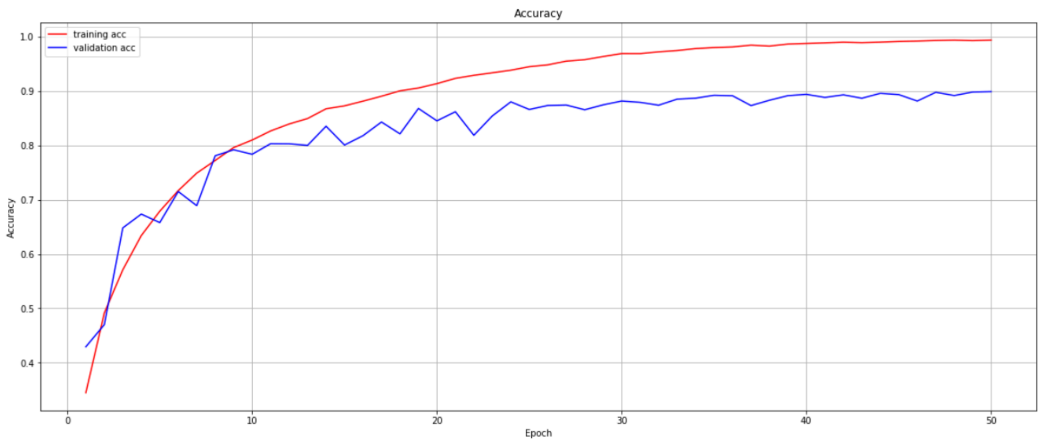

Figure 1: the accuracy of densenet

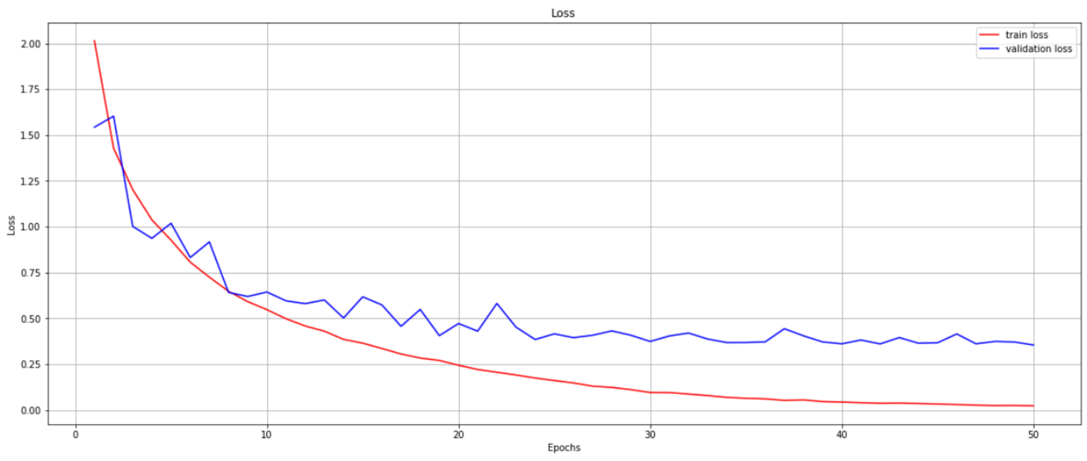

Figure 2: loss figure

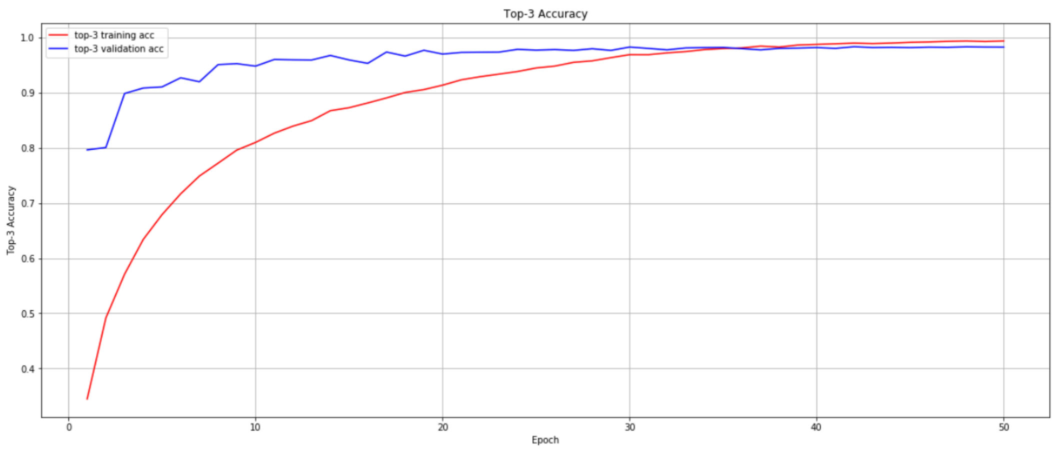

Figure 3: Top 3 accuracy

From my experiments, I found that adding dropout, activation functions, convolutional or fully connected layers, and adjusting optimizers have minimal impact on the final results. The most crucial aspect is data processing, including techniques like data augmentation and flipping. Figure 4 below presents the results of Vision Transformer on the CIFAR-10 dataset.

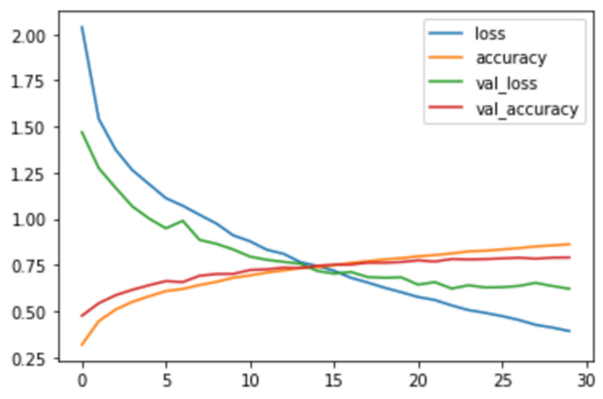

Figure 4: vision transformer

On the CIFAR-10 dataset, the Vision Transformer (ViT) shows some interesting results when compared with neural networks. According to search results, the performance of ViT on CIFAR-10 differs from that of Convolutional Neural Networks (CNNs). Traditional CNNs, due to their local receptive fields and weight sharing characteristics, typically perform well on small dataset tasks.

On the other hand, ViT processes images by dividing them into fixed-size patches and inputting these patches as serialized tokens into the Transformer model, utilizing self-attention mechanisms to handle the images. This approach excels at handling global information but may not be as efficient as CNNs on small datasets. A study proposed the addition of deep convolutional modules as shortcut connections in the ViT model to ensure that the model can capture both local and global information, significantly improving ViT's performance on small datasets like CIFAR-10.

The ViT outweighs the neural networks in terms of capturing global information on CIFAR-10, but it needs more methods (data augmentation, structural) to outperform the neural networks to capture global information on small datasets. These findings highlight the need to select the right model architecture for specific data and tasks.

Following my results, changing the parameters of the Vision Transformer (ViT) has a straight effect on the performance of the model. By running multiple scenarios through parameters such as Image segmenting method (Patch Size), Number of Transformer encoder layers, Number of heads in a multi-head attention mechanism, a hidden layer dimension (Hidden Size), and learning rate (Learning Rate), we can have a certain impact on the accuracy and efficiency of the given model. Patch Size: consider that with a larger Patch Size, we capture more global information, yet we lose some details, and the opposite with smaller Patch Size. Adding more encoder layers potentially improves the expressive power of the model at the cost of increased time complexity. Although increasing heads does help the model capture more details, it incurs higher computational costs. We can increase the hidden layer dimension to improve the model's expressive power, but it also raises the computing and memory overhead. The learning rate impacts both convergence speed and stability of models, just like in neural networks.

Conclusion

Also, the CIFAR-10 classification effect and CIFAR-100 classification effect prove the efficiency of this architecture on complex image classification tasks. Such a pattern of connecting can enable building a deeper model to adjust the number of parameters that have access to help the function as a bottleneck when denominated as a transition between input layers and levels of abstraction. The following report highlights the significance of architectural designs in deep learning models. These results demonstrate how the design of the DenseNet architecture allows for greater reuse of extracted features, leading to more effective use of learned patterns. This is due to the fact that dense connections between layers allowed for a flow of information as well as stable gradients even in very deep networks. Additionally, through the utilization of bottleneck layers and transition layers, we were able to maintain the richness of the learned features while alleviating some of the computational strain. Then, respectively, we go through models of the other Transformer[3] model or large language model.

References

[1] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4700-4708.

[2] Kingma D P. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[3] Vaswani A. Attention is all you need[J]. Advances in Neural Information Processing Systems, 2017.