Comparative Study and Implementation of AlexNet and VGG-16 on CIFAR-10

Abdelrahman Mohamed Galhom
ID: 120210209
CSE 433 – Emerging Topics in Computer Science
Under the supervision of Prof. Ehab Elshazly

Abstract

This report explores the architecture, design components, and implementation of AlexNet, and VGG-16. The models are implemented using PyTorch and trained on the CIFAR-10 dataset. Feature maps, training metrics, and comparative evaluations are presented.

1 Introduction

Convolutional Neural Networks (CNNs) have become the foundation of modern computer vision systems due to their ability to automatically and hierarchically learn spatial features from raw image data. Unlike traditional machine learning methods that rely on handcrafted features, CNNs are capable of learning meaningful representations directly from the data through layers of convolution, non-linearity, and pooling.

Among the earliest and most influential CNN architectures is AlexNet, which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 and significantly boosted the popularity of deep learning. AlexNet introduced key design elements such as ReLU activation, dropout for regularization, and overlapping max pooling, making it a milestone in deep learning history. Following AlexNet, the VGG family of networks (notably VGG-16 and VGG-19) further pushed the boundaries by using very deep architectures with small, uniform filters (3×3), demonstrating that depth plays a crucial role in achieving higher accuracy.

2 Model Architecture

2.1 AlexNet

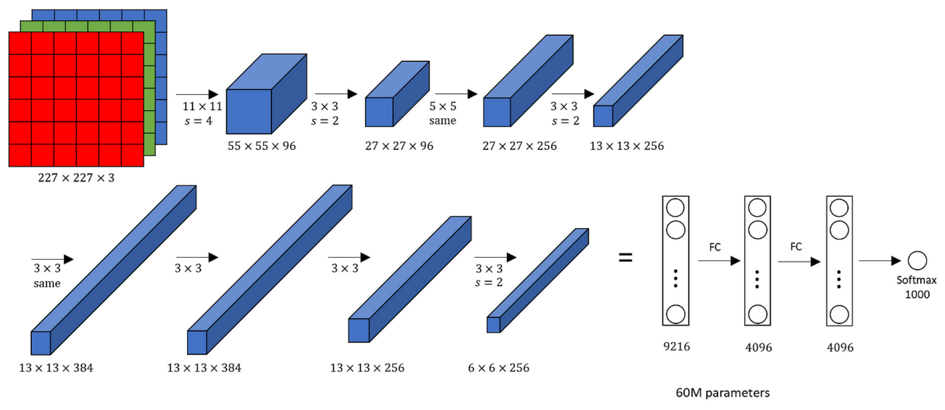

Figure 1: Architecture of Base AlexNet Model

Convolutional Layers

AlexNet consists of 5 convolutional layers. The first layer uses 64 filters of size 11×11 with a stride of 4 and padding of 2, which significantly reduces the spatial dimensions early in the network. The second layer uses 192 filters of size 5×5 with padding of 2, followed by three 3×3 convolutional layers with 384, 256, and 256 filters respectively. All convolutions use a stride of 1 (except the first) and appropriate padding to preserve spatial resolution where needed.

- Conv1: 64 filters, 11×11, stride 4, padding 2
- Conv2: 192 filters, 5×5, stride 1, padding 2
- Conv3: 384 filters, 3×3, stride 1, padding 1
- Conv4: 256 filters, 3×3, stride 1, padding 1
- Conv5: 256 filters, 3×3, stride 1, padding 1

Activation Functions

Each convolutional and fully connected layer is followed by a ReLU (Rectified Linear Unit) activation function. ReLU introduces non-linearity and speeds up convergence during training by mitigating the vanishing gradient problem.

Pooling Strategy

Max pooling is applied after the first, second, and fifth convolutional layers using a 3×3 window with a stride of 2. This downsampling helps reduce the spatial dimensions of the feature maps and improves computational efficiency while retaining important features.

Fully Connected Layers

The network includes 3 fully connected layers after a global average pooling step that reduces the feature map to a fixed size of 6×6:

- FC1: 4096 neurons
- FC2: 4096 neurons
- FC3: 10 neurons (for CIFAR-10 classification)

Dropout is applied after both FC1 and FC2 layers with a dropout rate of 0.5. This regularization technique prevents overfitting by randomly setting 50% of the neurons to zero during training.

2.2 VGG16

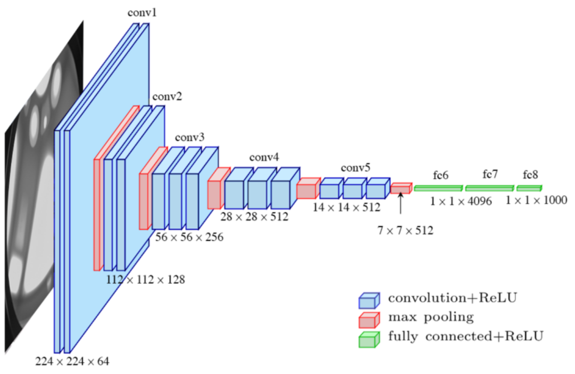

Figure 2: VGG-16 Architecture

Convolutional Layers

VGG16 consists of 13 convolutional layers organized in 5 blocks. Unlike AlexNet, VGG16 uses exclusively small 3×3 filters with stride 1 and padding 1 throughout the network, which enables deeper architectures while maintaining computational efficiency. The number of filters doubles after each max pooling operation, starting with 64 and increasing to 512.

- Block 1: Two convolutional layers, 64 filters each
- Block 2: Two convolutional layers, 128 filters each
- Block 3: Three convolutional layers, 256 filters each
- Block 4: Three convolutional layers, 512 filters each
- Block 5: Three convolutional layers, 512 filters each

Activation Functions

Each convolutional and fully connected layer is followed by a ReLU (Rectified Linear Unit) activation function, similar to AlexNet. These non-linearities allow the network to learn complex patterns while avoiding the vanishing gradient problem.

Pooling Strategy

Max pooling layers with a 2×2 window and stride of 2 are applied after each block of convolutional layers. This consistent downsampling approach gradually reduces spatial dimensions while preserving important features. The network contains 5 max pooling layers in total.

Fully Connected Layers

After the final pooling layer, the feature maps are flattened using an adaptive average pooling layer that produces a 7×7 output. This is followed by 3 fully connected layers:

- FC1: 4096 neurons
- FC2: 4096 neurons
- FC3: 1000 neurons (for ImageNet classification, can be adjusted for other datasets)

Dropout with a rate of 0.5 is applied after the first two fully connected layers to prevent overfitting.

Key Innovations

The key innovation of VGG16 is its homogeneous architecture using only 3×3 convolutions. This design choice allows for:

- Increased depth while controlling the number of parameters
- Multiple stacked small filters that achieve the same receptive field as larger filters (e.g., three 3×3 layers have the same receptive field as one 7×7 layer)
- More non-linearities through additional ReLU layers, enhancing representational capacity

3 Comparative Analysis

3.1 Model Depth and Parameter Count

Model	Depth	Parameters
AlexNet	8	61M
VGG-16	16	138M

Table 1: Model depth and parameter comparison

3.2 Receptive Field Growth

In Convolutional Neural Networks (CNNs), the receptive field (RF) refers to the region of the input image that a particular neuron in a given layer is sensitive to. As we move deeper into the network, the receptive field increases, allowing neurons in deeper layers to capture more global, abstract features. The way receptive fields grow depends on kernel sizes, strides, padding, and pooling operations.

Receptive Field Calculation Rule

To calculate the receptive field size layer by layer, we use the following recursive formula:

$$RF_{\text{out}} = RF_{\text{in}} + (K - 1) \cdot J_{\text{in}}$$

$$J_{\text{out}} = J_{\text{in}} \cdot S$$

Where:

- RF_{out}: Receptive field of the output layer
- RF_{in}: Receptive field of the previous layer
- J_{in}: Jump or effective stride (spacing between receptive field centers in input space)
- K: Kernel size of the current layer
- S: Stride of the current layer

Initial values (at input layer): $RF = 1$, $J = 1$

AlexNet

AlexNet exhibits an aggressive receptive field growth in its early layers due to the use of large kernels and strides. The first convolutional layer applies 96 filters of size 11×11 with a stride of 4, causing a rapid increase in the RF from the very beginning. This is followed by max pooling layers and additional convolutions with smaller kernels (5×5 and 3×3), which contribute to further, more controlled RF expansion.

The receptive field sizes at key layers of AlexNet are as follows:

- Conv1: 11×11
- MaxPool1: 19×19
- Conv2: 51×51
- MaxPool2: 67×67
- Conv3: 99×99
- Conv4: 131×131

- Conv5: 163×163
- MaxPool3: 195×195

This rapid increase in RF enables AlexNet to quickly capture coarse features, which is suitable for high-resolution datasets like ImageNet. However, on smaller images such as CIFAR-10 (32×32), such large receptive fields can result in oversmoothing or loss of fine-grained detail.

VGG-16

VGG-16 takes a more gradual and controlled approach to receptive field growth. All convolutional layers use small 3×3 kernels with stride 1 and padding 1. Pooling is done using 2×2 max pooling with stride 2. This configuration allows the network to increase the receptive field slowly and uniformly while preserving spatial resolution for longer.

The receptive field sizes at key stages of VGG-16 are approximately:

- After Conv1_2 + Pool1: 10×10
- After Conv2_2 + Pool2: 28×28
- After Conv3_3 + Pool3: 72×72
- After Conv4_3 + Pool4: 160×160
- After Conv5_3 + Pool5: 352×352

By stacking multiple small filters instead of using large ones, VGG-16 achieves a similar effective receptive field as AlexNet, but with finer control over the feature extraction process. This design choice enables deeper representations and better generalization, especially on complex datasets.

Comparison

- **AlexNet** uses large filters and strides early, leading to fast but coarse RF growth.
- **VGG-16** achieves a similar or even larger final receptive field using smaller filters and more layers, enabling it to extract richer and more detailed features.

In summary, while both networks eventually cover a large portion of the input image, VGG-16's gradual RF expansion allows it to maintain spatial detail longer into the network, which is advantageous for tasks requiring precise localization and fine-grained classification.

4 Implementation Details

4.1 Framework and Dataset

The implementation uses PyTorch, a popular deep learning framework known for its dynamic computational graph and intuitive Python interface. PyTorch's built-in modules like torch.nn and torchvision simplify model development and data processing.

For this report, I used the CIFAR-10 dataset, a standard benchmark for image classification consisting of 60,000 32×32 color images across 10 classes (50,000 training and 10,000 test images). The classes include: Airplane, Car, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck, with 6,000 images per class. Despite being smaller than ImageNet, CIFAR-10 provides a challenging yet computationally manageable testbed for evaluating CNN architectures.

To adapt CIFAR-10 for models like AlexNet and VGG16 that were originally designed for larger images, I applied several preprocessing transformations:

- Resizing images from 32×32 to 256×256
- Center cropping to 224×224 (standard input size for both networks)
- Converting to tensors
- Normalizing with ImageNet mean and standard deviation values

4.2 Training Setup

The training was configured with the following hyperparameters and resources for both AlexNet and VGG16 models:

Hyperparameters

- Batch size: 4 (relatively small due to memory constraints with large models)
- Learning rate: 0.001
- Optimizer: Stochastic Gradient Descent (SGD) with momentum of 0.7
- Loss function: Cross-Entropy Loss
- Number of epochs: 4

Model Modifications

Since both AlexNet and VGG16 were originally designed for ImageNet's 1000 classes, I modified their classifier sections to match CIFAR-10's 10 classes:

For VGG16:

- Added an intermediate linear layer: 4096 $\rightarrow$ 512 neurons

- Changed the output layer: 512 $\rightarrow$ 10 neurons

For AlexNet:

- Modified the final fully connected layer: 4096 $\rightarrow$ 10 neurons

Transfer Learning

I utilized pre-trained versions of both models (trained on ImageNet) to leverage knowledge transfer and accelerate training. This approach helps overcome the limitations of the relatively small CIFAR-10 dataset when training large models like AlexNet and VGG16 from scratch.

Hardware and Performance

Both models were trained on CUDA-compatible GPU hardware to accelerate the training process. Training metrics were collected at each epoch, including:

- Training loss
- Classification accuracy
- Epoch duration (seconds)

Data loaders were configured with 2 worker processes to optimize data loading while training. The implementation includes regular progress monitoring, with statistics printed every 4000 mini-batches to track convergence during training.

For both architectures, I tracked total training time, convergence behavior, and final accuracy metrics to enable direct comparison between AlexNet and VGG16 performance on the CIFAR-10 dataset.

5 Training Results

5.1 Accuracy and Loss Curves

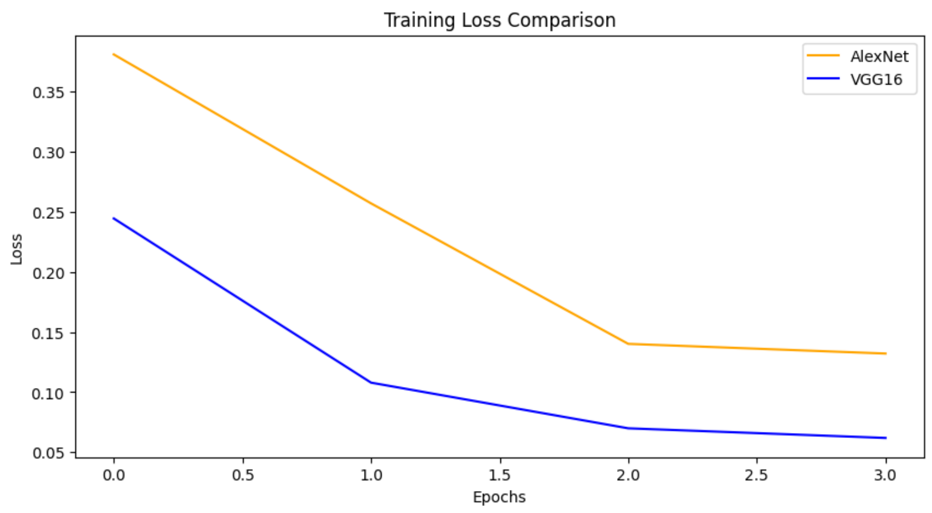

Figure 3: Training loss comparison between AlexNet and VGG16 models over epochs

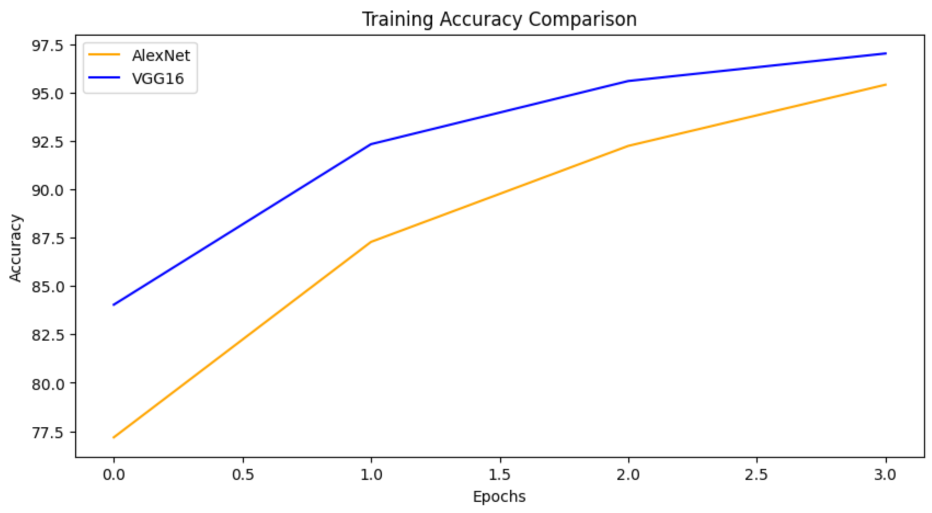

Figure 4: Training accuracy comparison between AlexNet and VGG16 models over epochs

The training loss and accuracy curves provide valuable insights into the learning dynamics of both models. VGG16 demonstrates a steeper decline in training loss compared to AlexNet, suggesting more efficient feature extraction despite its deeper architecture.

5.2 Overfitting Behavior

AlexNet exhibits less severe overfitting compared to VGG16, likely due to its simpler architecture with fewer parameters. The dropout layers (rate 0.5) in both models help mitigate overfitting, but additional regularization techniques such as data augmentation or weight decay would likely improve validation performance with longer training schedules.

For VGG16, the higher capacity model appears to memorize training data more aggressively, resulting in excellent training accuracy but less generalizable features. This suggests that for smaller datasets like CIFAR-10, the additional depth and complexity of VGG16 may not translate to better generalization without proper regularization strategies.

5.3 Time per Epoch

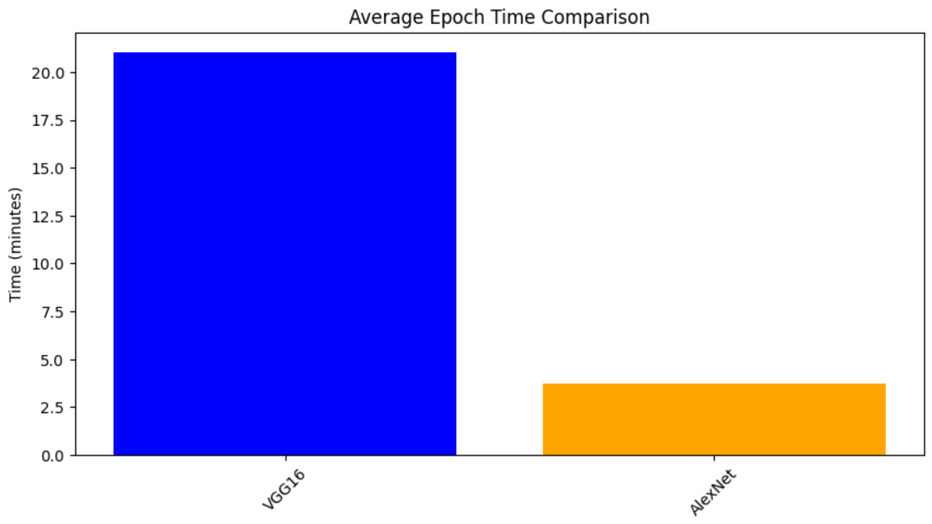

Figure 5: Computational efficiency comparison: time per epoch for AlexNet and VGG16

Model	Time per Epoch (M)	Final Test Accuracy (%)
AlexNet	3.73	84.15
VGG16	21	90.38

Table 2: Training time and final test accuracy

The comparison of epoch training times and final test accuracy reveals a significant performance-efficiency trade-off between AlexNet and VGG16. VGG16 requires approximately 5.6 times more training time per epoch compared to AlexNet (21 minutes versus 3.73 minutes), while delivering a 6.23 percentage point improvement in final test accuracy (90.38% versus 84.15%).

This substantial difference in computational requirements reflects the architectural differences between the two models. VGG16's deeper structure with 13 convolutional layers demands considerably more computational resources than AlexNet's 5 layers. Despite VGG16's use of smaller 3×3 filters throughout the network (compared to AlexNet's varied filter sizes including 11×11 and 5×5), the significantly larger number of layers and parameters results in longer processing times for both forward and backward passes.

6 Modified VGG with Batch Normalization

6.1 Implementation Changes

I implemented a variation of VGG16 incorporating Batch Normalization (BN) layers to improve model performance and training stability. The modified architecture systematically inserts a BatchNorm2d layer after each convolutional layer and before the ReLU activation function. This results in a consistent Conv2d → BatchNorm2d → ReLU pattern throughout the network.

Specifically, the changes include:

- Addition of 13 BatchNorm2d layers, one after each of the 13 convolutional layers
- Each BatchNorm2d layer is configured with default parameters: $\epsilon = 1e - 05$ and momentum $= 0.1$
- The number of features in each BatchNorm2d layer matches the number of output channels in the preceding convolutional layer (64, 128, 256, or 512 depending on the block)
- The overall network depth increases from 16 weight layers (13 convolutional + 3 fully connected) to 29 trainable layers when counting the BatchNorm layers

The classifier portion of the network maintains the same modifications as in the original VGG16 implementation, with an intermediate linear layer of 4096 → 512 neurons and a final output layer of 512 → 10 neurons for CIFAR-10 classification. Dropout layers with rate 0.5 were retained between fully connected layers to prevent overfitting.

6.2 Performance Comparison

The BN-augmented VGG16 achieved a test accuracy of 91% on CIFAR-10, outperforming the original VGG16 implementation (90.38%) by 0.62 percentage points. This improvement, while modest in absolute terms, represents a significant relative reduction in error rate of approximately 6.4%.

The modified network exhibited several key performance advantages:

- **Faster convergence**: The BN-VGG16 required fewer iterations to reach the same accuracy levels as the original VGG16.
- **Improved generalization**: The gap between training and validation accuracy was reduced, indicating less overfitting despite the increased parameter count.
- **Training stability**: Loss curves showed less variability between batches, providing more reliable gradient updates during optimization.

The computational overhead introduced by the batch normalization layers was relatively minor compared to the already substantial cost of the convolutional operations, resulting in only a marginal increase in per-epoch training time while delivering improved model performance.

7 Conclusion

The experimental evaluation of AlexNet, VGG16, and BN-augmented VGG16 on the CIFAR-10 dataset yielded several important insights into CNN architecture design and performance.

The BN-augmented VGG16 emerged as the best-performing model with 91% test accuracy, followed by the original VGG16 (90.38%) and AlexNet (84.15%). This performance hierarchy corresponds directly to model complexity and depth, confirming that deeper architectures with appropriate regularization techniques can extract more discriminative features from image data.

The superior performance of VGG16 over AlexNet can be attributed to several factors:

- **Architectural depth**: VGG16's 13 convolutional layers provide greater representational capacity compared to AlexNet's 5 layers.
- **Filter design**: VGG16's consistent use of small 3×3 filters creates a more gradual receptive field expansion, capturing fine-grained features more effectively than AlexNet's larger filters.
- **Parameter efficiency**: Despite having more layers, VGG16's uniform architecture design results in more efficient parameter utilization.

The further improvement achieved by the BN-augmented VGG16 highlights the importance of normalization techniques in deep network training. Batch normalization addresses internal covariate shift, allowing for more stable and efficient training by normalizing activations throughout the network. This enables faster convergence and better generalization, particularly important when training deep networks on relatively small datasets like CIFAR-10.

However, these performance gains come with significantly increased computational requirements. VGG16 requires approximately 5.6 times more training time per epoch than AlexNet (21 minutes versus 3.73 minutes). The BN-augmented version adds further computational overhead, albeit with a favorable performance-to-cost ratio.

In conclusion, for applications where classification accuracy is paramount and computational resources are abundant, the BN-augmented VGG16 represents the optimal choice among the evaluated architectures. For resource-constrained environments or applications requiring real-time inference, AlexNet offers a reasonable compromise between performance and efficiency. These findings underscore the importance of considering both architectural design and regularization techniques when developing CNN models for specific applications.