Comparative Study and Implementation of AlexNet and VGG Architectures on CIFAR-10 Dataset

Abstract

This study presents a comparative analysis of AlexNet and VGG architectures implemented on the CIFAR-10 dataset. The research explores the architectural design of both networks, their performance characteristics, and modifications to improve performance. Through implementation in PyTorch, we compare these models based on accuracy, training dynamics, parameter efficiency, and generalization capabilities. Our findings reveal that VGG-16 with Batch Normalization significantly outperforms other architectures, achieving 91.94% test accuracy after 40 epochs, while the standard VGG-16 without Batch Normalization fails to learn at all. We also demonstrate that a reduced-depth version of VGG (VGG-8) achieves comparable performance to AlexNet with fewer parameters, illustrating the importance of architectural design choices in convolutional neural networks.

Contents

1 Introduction

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision, with architectures like AlexNet and VGG representing significant milestones in the development of deep learning. AlexNet, introduced by Krizhevsky et al. in 2012, was the first CNN to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), demonstrating the power of deep learning for image classification. VGG, developed by the Visual Geometry Group at Oxford in 2014, pushed the boundaries further with deeper architectures and systematic design principles.

This study aims to:

1. Examine the architectural design of AlexNet and VGG networks
2. Implement and train these models on the CIFAR-10 dataset
3. Compare their performance in terms of accuracy, training dynamics, and parameter efficiency
4. Investigate the impact of modifications such as batch normalization and reduced network depth

The CIFAR-10 dataset, consisting of 60,000 32×32 color images across 10 classes, provides an ideal benchmark for comparing these architectures due to its manageable size while still presenting a meaningful classification challenge.

2 Architectural Design and Components

2.1 AlexNet

AlexNet represented a breakthrough in CNN architecture when it was introduced in 2012. The original AlexNet was designed for 224×224 input images from ImageNet, but for this study, we adapt it to the smaller 32×32 CIFAR-10 images.

2.1.1 Convolutional Layers

Our AlexNet implementation for CIFAR-10 consists of five convolutional layers with the following configuration:

- Layer 1: 64 filters of size 3×3, stride 1, padding 1
- Layer 2: 192 filters of size 3×3, stride 1, padding 1
- Layer 3: 384 filters of size 3×3, stride 1, padding 1
- Layer 4: 256 filters of size 3×3, stride 1, padding 1
- Layer 5: 256 filters of size 3×3, stride 1, padding 1

Note that this is a simplified version compared to the original AlexNet, which used larger filter sizes (11×11 in the first layer) and strides, as those dimensions are not suitable for the smaller CIFAR-10 images.

2.1.2 Activation Functions

AlexNet uses Rectified Linear Unit (ReLU) activation functions after each convolutional and fully connected layer. ReLU is defined as $f(x) = \max(0, x)$, which helps alleviate the vanishing gradient problem present in earlier activation functions like sigmoid or tanh, allowing for faster training of deep networks.

2.1.3 Pooling Strategy

MaxPooling is applied after the 1st, 2nd, and 5th convolutional layers:

- MaxPool after Conv1: 2×2 kernel, stride 2
- MaxPool after Conv2: 2×2 kernel, stride 2
- MaxPool after Conv5: 2×2 kernel, stride 2

This progressively reduces spatial dimensions while preserving important features.

2.1.4 Fully Connected Layers

The AlexNet classifier consists of three fully connected (FC) layers:

- FC1: $256 \times 4 \times 4 \rightarrow 4096$ neurons
- FC2: $4096 \rightarrow 1024$ neurons
- FC3: $1024 \rightarrow 10$ neurons (output layer for CIFAR-10's 10 classes)

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

2.2 VGG-16

VGG architectures, introduced in 2014, are characterized by their use of small 3×3 convolutional filters throughout the network and increased depth compared to previous architectures.

2.2.1 Convolutional Layers

VGG-16 consists of 13 convolutional layers organized into five blocks:

- Block 1: Two convolutional layers with 64 filters of size 3×3, stride 1, padding 1
- Block 2: Two convolutional layers with 128 filters of size 3×3, stride 1, padding 1
- Block 3: Three convolutional layers with 256 filters of size 3×3, stride 1, padding 1
- Block 4: Three convolutional layers with 512 filters of size 3×3, stride 1, padding 1
- Block 5: Three convolutional layers with 512 filters of size 3×3, stride 1, padding 1

The consistent use of 3×3 filters with stride 1 and padding 1 throughout the network is a hallmark of the VGG design philosophy.

2.2.2 Activation Functions

Like AlexNet, VGG-16 uses ReLU activation functions after each convolutional and fully connected layer.

2.2.3 Pooling Strategy

MaxPooling with 2×2 kernels and stride 2 is applied after each block, resulting in a more regular pattern of pooling compared to AlexNet.

2.2.4 Fully Connected Layers

The VGG-16 classifier consists of three fully connected layers:

- FC1: 512 → 4096 neurons
- FC2: 4096 → 4096 neurons
- FC3: 4096 → 10 neurons (output layer for CIFAR-10's 10 classes)

In our implementation for CIFAR-10, we maintained the original fully connected layer structure but adapted the input size to match the feature maps produced by the convolutional layers.

2.3 VGG-16 with Batch Normalization

For this version, we modified the standard VGG-16 architecture by adding Batch Normalization layers after each convolutional layer but before the ReLU activation function. The rest of the architecture remains identical to the standard VGG-16.

2.4 VGG-8 (Reduced Depth)

VGG-8 is a reduced-depth version of VGG with 8 convolutional layers organized into four blocks:

- Block 1: Two convolutional layers with 64 filters of size 3×3, stride 1, padding 1
- Block 2: Two convolutional layers with 128 filters of size 3×3, stride 1, padding 1
- Block 3: Two convolutional layers with 256 filters of size 3×3, stride 1, padding 1
- Block 4: Two convolutional layers with 512 filters of size 3×3, stride 1, padding 1

The fully connected layers are simplified to:

- FC1: $512 \times 2 \times 2 \rightarrow 4096$ neurons
- FC2: 4096 → 10 neurons (output layer)

Model	Depth (Conv + FC Layers)	Parameter Count
AlexNet	5 + 3 = 8 layers	23.2M parameters
VGG-16	13 + 3 = 16 layers	33.6M parameters
VGG-16 with BN	13 + 3 = 16 layers	33.6M parameters
VGG-8	8 + 2 = 10 layers	13.1M parameters

Table 1: Comparison of model depth and parameter count

3 Comparative Analysis

3.1 Model Depth and Parameter Count

The parameter counts reveal that VGG-16 has approximately 45% more parameters than AlexNet, primarily due to its greater depth. VGG-16 with Batch Normalization has slightly more parameters than the standard VGG-16 due to the additional learnable parameters in the batch normalization layers. VGG-8 has significantly fewer parameters than both AlexNet and VGG-16, demonstrating the impact of reducing network depth.

3.2 Feature Extraction Strategies

AlexNet employs a more heterogeneous approach to feature extraction, with varying filter configurations across layers. The network starts with relatively few filters (64) and rapidly increases to 384 before reducing to 256 in the deeper layers. This design allows AlexNet to capture a diverse range of features at different scales.

VGG, on the other hand, follows a more homogeneous and systematic approach. The number of filters starts small (64) in the initial layers and doubles after each pooling layer ($64 \rightarrow 128 \rightarrow 256 \rightarrow 512 \rightarrow 512$). This systematic doubling of filters compensates for the reduction in spatial dimensions after pooling, maintaining the network's representational capacity.

3.3 Filter Design Philosophy

The filter design philosophy represents one of the most significant differences between AlexNet and VGG:

AlexNet was designed with a more ad-hoc approach, using different filter sizes across layers. In the original architecture, it used large 11×11 filters in the first layer with a stride of 4, rapidly downsampling the input. While our CIFAR-10 implementation uses 3×3 filters due to the smaller input size, the original design philosophy of AlexNet favored larger filters and aggressive downsampling early in the network.

VGG introduced a more principled approach by using only 3×3 convolutional filters with stride 1 throughout the network. The key insight of VGG was that stacking multiple 3×3 convolutional layers could achieve the same effective receptive field as larger filters while using fewer parameters and incorporating more non-linearities (activation functions). For example, two stacked 3×3 convolutions have the same effective receptive field as a single 5×5 convolution but with fewer parameters and an additional non-linearity.

3.4 Regularization and Generalization Techniques

Both networks employ several regularization techniques to prevent overfitting:

AlexNet uses:

- Dropout (0.5) in fully connected layers
- Data augmentation (random crops, horizontal flips)
- Weight decay (L2 regularization)

VGG uses:

- Similar regularization techniques as AlexNet
- More depth, which can act as an implicit regularizer

VGG with Batch Normalization adds:

- Batch normalization after each convolutional layer, which has a regularizing effect by normalizing layer inputs
- This also allows for higher learning rates and reduces the need for careful initialization

4 Receptive Field Analysis

The receptive field of a network refers to the region in the input space that influences a particular feature in the output. Understanding how the receptive field grows through a network provides insights into how the network processes spatial information.

4.1 Receptive Field Calculation

The receptive field size (r) at layer l can be calculated recursively:

- Initial receptive field (r_0) = 1
- r_l = r_l-1 + (k_l - 1) * s_l-1

Where k_l is the kernel size of layer l, and s_l-1 is the product of all previous strides.

4.2 AlexNet Receptive Field Growth

For our adapted AlexNet with 3×3 filters:

- Layer 1: 3×3 receptive field
- After Pool 1 (2×2, stride 2): 6×6 receptive field
- Layer 2: 10×10 receptive field
- After Pool 2 (2×2, stride 2): 14×14 receptive field
- Layer 3: 18×18 receptive field

- Layer 4: 22×22 receptive field
- Layer 5: 26×26 receptive field
- After Pool 5 (2×2, stride 2): 30×30 receptive field

By the final convolutional layer, AlexNet has a receptive field covering almost the entire 32×32 CIFAR-10 image.

4.3 VGG-16 Receptive Field Growth

For VGG-16 with all 3×3 filters:

- Block 1 (two 3×3 layers): 5×5 receptive field
- After Pool 1: 6×6 receptive field
- Block 2 (two 3×3 layers): 10×10 receptive field
- After Pool 2: 12×12 receptive field
- Block 3 (three 3×3 layers): 20×20 receptive field
- After Pool 3: 24×24 receptive field
- Block 4 (three 3×3 layers): 36×36 receptive field
- After Pool 4: 44×44 receptive field
- Block 5 (three 3×3 layers): 60×60 receptive field
- After Pool 5: 76×76 receptive field

VGG-16's receptive field grows more gradually but reaches a larger size than AlexNet, allowing it to capture wider contextual information. The receptive field exceeds the input image size of 32×32, which means that in the deeper layers, each neuron is influenced by the entire input image.

4.4 Detailed Receptive Field Calculations

4.4.1 AlexNet Detailed Calculations

For our AlexNet implementation modified for CIFAR-10 (32×32 images):

By the final layer, the receptive field is 46×46, which is larger than the input image size (32×32). This means that each neuron in the final layer can theoretically "see" the entire input image and potentially more (though in practice the receptive field is limited by the image boundaries).

4.4.2 VGG-16 Detailed Calculations

For the VGG-16 implementation adapted for CIFAR-10:

By the time we reach the final layer of VGG-16, the theoretical receptive field is 236×236, which is much larger than the 32×32 input image. This means that each neuron in the deepest layer of VGG-16 has the potential to be influenced by the entire input image (and more, if the image were larger).

Layer	Operation	Kernel	Stride	Cum. Stride	Receptive Field	Calculation
Input	-	-	-	1	1×1	r_0 = 1
Conv1	3×3 conv	3	1	1	3×3	r_1 = 1 + (3-1)*1 = 3
Pool1	2×2 maxpool	2	2	2	6×6	r_2 = 3 + (2-1)*1 = 4*1.5 ≈ 6
Conv2	3×3 conv	3	1	2	10×10	r_3 = 6 + (3-1)*2 = 10
Pool2	2×2 maxpool	2	2	4	14×14	r_4 = 10 + (2-1)*2*2 = 14
Conv3	3×3 conv	3	1	4	22×22	r_5 = 14 + (3-1)*4 = 22
Conv4	3×3 conv	3	1	4	30×30	r_6 = 22 + (3-1)*4 = 30
Conv5	3×3 conv	3	1	4	38×38	r_7 = 30 + (3-1)*4 = 38
Pool5	2×2 maxpool	2	2	8	46×46	r_8 = 38 + (2-1)*8 = 46

Table 2: Detailed receptive field calculations for AlexNet

Layer	Operation	Kernel	Stride	Cum. Stride	Receptive Field	Calculation
Input	-	-	-	1	1×1	r_0 = 1
Conv1_1	3×3 conv	3	1	1	3×3	r_1 = 1 + (3-1)*1 = 3
Conv1_2	3×3 conv	3	1	1	5×5	r_2 = 3 + (3-1)*1 = 5
Pool1	2×2 maxpool	2	2	2	6×6	r_3 = 5 + (2-1)*1 = 6
Conv2_1	3×3 conv	3	1	2	10×10	r_4 = 6 + (3-1)*2 = 10
Conv2_2	3×3 conv	3	1	2	14×14	r_5 = 10 + (3-1)*2 = 14
Pool2	2×2 maxpool	2	2	4	16×16	r_6 = 14 + (2-1)*2 = 16
Conv3_1	3×3 conv	3	1	4	24×24	r_7 = 16 + (3-1)*4 = 24
Conv3_2	3×3 conv	3	1	4	32×32	r_8 = 24 + (3-1)*4 = 32
Conv3_3	3×3 conv	3	1	4	40×40	r_9 = 32 + (3-1)*4 = 40
Pool3	2×2 maxpool	2	2	8	44×44	r_10 = 40 + (2-1)*4 = 44

Table 3: Partial receptive field calculations for VGG-16 (first 3 blocks)

4.5 Comparison of Receptive Field Growth

VGG's approach of stacking multiple small filters leads to a more gradual increase in the receptive field compared to AlexNet's original design with larger filters. This gradual growth allows VGG to learn more complex hierarchical patterns and may contribute to its better performance on various vision tasks.

5 Implementation and Training

5.1 Implementation Details

All models were implemented in PyTorch with the following configurations:

- Optimizer: SGD with momentum 0.9 and weight decay 5e-4
- Learning rate: 0.01 with cosine annealing schedule
- Loss function: Cross-entropy loss
- Batch size: 128

Layer	Operation	Kernel	Stride	Cum. Stride	Receptive Field	Calculation
Conv4_1	3×3 conv	3	1	8	60×60	r_11 = 44 + (3-1)*8 = 60
Conv4_2	3×3 conv	3	1	8	76×76	r_12 = 60 + (3-1)*8 = 76
Conv4_3	3×3 conv	3	1	8	92×92	r_13 = 76 + (3-1)*8 = 92
Pool4	2×2 maxpool	2	2	16	108×108	r_14 = 92 + (2-1)*16 = 108
Conv5_1	3×3 conv	3	1	16	140×140	r_15 = 108 + (3-1)*16 = 140
Conv5_2	3×3 conv	3	1	16	172×172	r_16 = 140 + (3-1)*16 = 172
Conv5_3	3×3 conv	3	1	16	204×204	r_17 = 172 + (3-1)*16 = 204
Pool5	2×2 maxpool	2	2	32	236×236	r_18 = 204 + (2-1)*32 = 236

Table 4: Partial receptive field calculations for VGG-16 (blocks 4-5)

- Data augmentation: Random crop with padding 4, random horizontal flip
- Normalization: Mean (0.4914, 0.4822, 0.4465), Std (0.2470, 0.2435, 0.2616)

All models were trained for 40 epochs, except for an additional shorter 5-epoch experiment to compare early learning behavior.

5.2 Training Results

5.2.1 Test Accuracy

Model	5 Epochs	40 Epochs
AlexNet	56.84%	87.81%
VGG-16	10.00%	10.00%
VGG-16 with BN	83.86%	91.94%
VGG-8	27.75%	88.32%

Table 5: Test accuracy comparison

The most striking observation is that the standard VGG-16 fails to learn at all, remaining at 10% accuracy (equivalent to random guessing on CIFAR-10's 10 classes) even after 40 epochs. In contrast, VGG-16 with Batch Normalization achieves the highest accuracy of 91.94% after 40 epochs, significantly outperforming all other models.

5.2.2 Training Time

Model	Avg. Time per Epoch (5 Epochs)	Avg. Time per Epoch (40 Epochs)
AlexNet	25.48s	36.09s
VGG-16	30.89s	47.08s
VGG-16 with BN	31.55s	50.20s
VGG-8	25.13s	37.97s

Table 6: Training time comparison

VGG-16 with Batch Normalization has the highest computational cost per epoch, which is expected given its additional batch normalization layers. Interestingly, VGG-8 has a similar training time to AlexNet despite having fewer parameters, suggesting that the depth of the network also influences computational efficiency.

6 Result Analysis and Discussion

6.1 Learning Dynamics

6.1.1 AlexNet

- Shows steady improvement throughout training
- Reaches 56.84% accuracy in 5 epochs and 87.81% in 40 epochs
- Displays good learning capacity without suffering from vanishing gradients

6.1.2 VGG-16

- Completely fails to learn, stuck at 10% accuracy (random chance)
- The loss remains constant at around 2.3 throughout training
- This failure is consistent with known issues in training very deep networks without proper normalization or initialization

6.1.3 VGG-16 with Batch Normalization

- Learns remarkably fast, reaching 83.86% accuracy in just 5 epochs
- Achieves the highest final accuracy of 91.94% after 40 epochs
- Demonstrates the dramatic impact of batch normalization on training deep networks

6.1.4 VGG-8

- Shows slower initial learning (27.75% after 5 epochs) compared to AlexNet
- Eventually catches up to AlexNet's performance (88.32% vs. 87.81% after 40 epochs)
- Illustrates that reduced depth can still achieve strong performance with proper training

6.2 The Impact of Batch Normalization

The dramatic difference between VGG-16 (10.00% accuracy) and VGG-16 with Batch Normalization (91.94% accuracy) provides a clear demonstration of the importance of batch normalization in training deep networks. Batch normalization addresses the internal covariate shift problem by normalizing the inputs to each layer, which helps in several ways:

1. **Stabilizes learning**: By normalizing layer inputs, batch normalization reduces the tendency of gradients to explode or vanish.

2. **Accelerates training**: VGG-16 with BN reaches higher accuracy in 5 epochs than VGG-16 without BN reaches in 40 epochs.

3. **Improves generalization**: The final accuracy of VGG-16 with BN is significantly higher than AlexNet or VGG-8.

4. **Acts as regularization**: Batch normalization has a regularizing effect that helps prevent overfitting.

6.3 The Effect of Network Depth

Comparing VGG-16 BN (91.94% accuracy) with VGG-8 (88.32% accuracy) demonstrates that additional depth does provide benefits in terms of final accuracy. However, the comparison between VGG-8 and AlexNet (87.81% accuracy) is particularly interesting. Despite having significantly fewer parameters (13.1M vs. 23.2M), VGG-8 achieves slightly better accuracy, suggesting that the VGG architecture's consistent use of 3×3 filters is more parameter-efficient than AlexNet's more heterogeneous design.

6.4 Parameter Efficiency

Model	Parameters	Test Accuracy	Parameters per % Accuracy
AlexNet	23.2M	87.81%	264K
VGG-16	33.6M	10.00%	3,360K
VGG-16 with BN	33.6M	91.94%	366K
VGG-8	13.1M	88.32%	148K

Table 7: Parameter efficiency comparison

This analysis reveals that VGG-8 is the most parameter-efficient model, requiring only 148K parameters per percentage point of accuracy. VGG-16 with Batch Normalization, while achieving the highest accuracy, is less parameter-efficient than VGG-8 and AlexNet. The standard VGG-16 is incredibly inefficient due to its failure to learn.

7 Visualization of Learned Features

7.1 Filter Visualization

Visualization of the first-layer filters reveals distinct patterns learned by each network:

- **AlexNet**: First-layer filters show a mix of edge detectors and color blobs, reflecting the network's ability to capture both shape and color information.

- **VGG-16**: Due to its failure to learn, the filters remain largely unstructured, with random noise-like patterns.

- **VGG-16 with BN**: Displays well-defined edge detectors, color blobs, and more structured patterns than AlexNet, indicating better feature learning.
- **VGG-8**: Shows similar patterns to VGG-16 with BN but with slightly less structure, consistent with its lower but still competitive performance.

7.2 Feature Map Visualization

Feature maps from the first convolutional layer show how each network processes input images:

- **AlexNet**: Feature maps highlight edges, textures, and color regions effectively.
- **VGG-16**: Feature maps lack meaningful structure, consistent with the network's failure to learn.
- **VGG-16 with BN**: Feature maps show strong activation for relevant features, with clear differentiation between background and foreground.
- **VGG-8**: Feature maps are similar to VGG-16 with BN but less refined, consistent with its lower depth.

8 Conclusion

This comparative study of AlexNet and VGG architectures on the CIFAR-10 dataset yields several important insights:

1. **The critical role of batch normalization**: VGG-16 with Batch Normalization dramatically outperforms standard VGG-16, achieving the highest test accuracy of 91.94%. This underscores the importance of normalization techniques in training deep networks.
2. **Architecture design matters**: Despite having fewer parameters, VGG-8 outperforms AlexNet, suggesting that the consistent use of small 3×3 filters in VGG architectures is more parameter-efficient than AlexNet's more heterogeneous design.
3. **Depth vs. efficiency trade-off**: While VGG-16 with BN achieves the highest accuracy, VGG-8 provides the best balance between accuracy and parameter efficiency, requiring only 148K parameters per percentage point of accuracy.
4. **Early performance indicators**: The 5-epoch experiments demonstrate that early performance can be a strong indicator of final performance, with VGG-16 with BN showing superior learning dynamics from the beginning.

These findings highlight the importance of architectural design choices in convolutional neural networks and demonstrate that clever design principles (like VGG's consistent use of small filters) and normalization techniques (like batch normalization) can significantly improve performance and efficiency.

9 References

1. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

2. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

3. Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

4. Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

5. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

6. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

7. Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

8. Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.

9. Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization? In Advances in Neural Information Processing Systems (pp. 2483-2493).