CNN Algorithms Architecture Report

AlexNet

AlexNet, proposed in 2012 by Google engineers, is considered a breakthrough and game-changing design in the field of computer vision. It achieved remarkable success by winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, significantly outperforming all other competitors. This success marked a pivotal moment, demonstrating the power of deep neural networks and large-scale learning, leading to the development of subsequent influential architectures.

Historical Context: Before AlexNet, datasets for computer vision were often small and specialized. However, the increasing availability of cheaper compute resources spurred the creation of larger datasets like LabelMe and the massive ImageNet, which contains around 15 million labeled images organized in a hierarchical structure. Training models on such large datasets necessitated models with higher capacity, for which Convolutional Neural Networks (CNNs) are particularly well-suited due to their built-in network structure that resembles the human visual system. The ILSVRC, an annual competition, provided a benchmark for evaluating these models, and AlexNet's performance in 2012 showcased a significant leap in accuracy.

Architectural Overview: AlexNet introduced an eight-layer deep network. This architecture comprised:

- Five convolutional layers (CNN layers). These layers are responsible for extracting hierarchical features from the input image by convolving learned filters (kernels) across it. The first convolutional layer used a large 11x11 filter with a stride of 4, applying 96 kernels. Subsequent convolutional layers employed different filter sizes and numbers of kernels (e.g., a 5x5 filter with 256 kernels in the second layer).
- Three fully connected layers (DNN or deep neural network layers). These layers take the high-level features learned by the convolutional layers and perform classification. The first two fully connected layers each had 4096 neurons.
- The final layer was a 1000-way softmax layer, which outputted a probability distribution over the 1000 object categories of the ImageNet dataset used for training.

The architecture also incorporated max pooling layers after some of the convolutional layers. These pooling layers reduced the spatial dimensions of the feature maps, making the network more invariant to small translations in the input. AlexNet employed 3x3 max pooling with a stride of 2, which resulted in overlapping pooling where the pooling regions overlapped.

A key aspect of AlexNet's implementation was its training on two GPUs. This was due to the memory limitations of single GPUs at the time and allowed for a larger network size. The network was strategically split across the two GPUs, with inter-GPU connections in the third convolutional layer and the fully connected layers, while connections in layers two, four, and five were confined to a single GPU. This division also led to the GPUs learning potentially different representations, such as one being more attuned to color information.

The input to AlexNet was typically an image of size 227x227x3 (although the research paper mentions 224x224, the difference is attributed to data augmentation) representing the height, width, and RGB channels. These images were often pre-processed by reducing their resolution to around 256x256 pixels.

Key Architectural Innovations and Their Contributions:

- ReLU (Rectified Linear Unit) Activation Function: AlexNet was a pioneer in using the ReLU activation function throughout the network, replacing the more traditional sigmoid and tanh functions. ReLU, defined as max(0, x), has a crucial advantage: it enables faster learning compared to saturating activation functions like sigmoid and tanh. ReLU helps to alleviate the vanishing gradient problem, which can hinder the training of deep networks. Studies showed that ReLU-based networks could train several times faster (e.g., six times faster) than those using tanh.

- Dropout: To combat the issue of overfitting, especially with a large number of parameters (62 million parameters in AlexNet) and a relatively limited amount of labeled data, AlexNet introduced the dropout technique. Dropout randomly deactivates neurons with a probability of 0.5 in the fully connected layers during training. This prevents neurons from becoming too co-dependent on each other, forcing the network to learn more robust and generalizable features. By effectively training an ensemble of thinned networks, dropout significantly reduces overfitting and improves the model's performance on unseen data.

- Local Response Normalization (LRN): AlexNet employed local response normalization (LRN) in the first and second convolutional layers. This technique, inspired by biological mechanisms in the retina, performs a kind of brightness normalization across adjacent feature maps. While LRN was used in AlexNet, it has largely been replaced by batch normalization (BN) in more recent architectures.

- Overlapping Pooling: Unlike the non-overlapping pooling typically used in earlier CNNs, AlexNet utilized overlapping max pooling (3x3 kernel with a stride of 2). The authors found that this strategy reduced the top-1 and top-5 error rates compared to non-overlapping pooling and also helped to reduce overfitting slightly.

- Data Augmentation: To further reduce overfitting and improve the model's robustness, AlexNet employed various data augmentation techniques. These included translation and horizontal reflection of the training images. Additionally, they altered the intensity of the RGB channels in the training images using Principal Component Analysis (PCA) on the ImageNet dataset, adding multiples of the principal components multiplied by random variables to simulate changes in illumination and color. This ensured that the network learned that object identity is invariant to these changes. Training was performed on millions of augmented images.

Training Process: AlexNet was trained using batch stochastic gradient descent (SGD) with momentum. The large dataset and the depth of the network required significant computational resources and time for training, which was facilitated by the use of two powerful GPUs.

Performance and Impact: AlexNet's performance in the ILSVRC 2012 was groundbreaking. It achieved a top-5 error rate of 18.2%, which was a significant improvement (around 8% lower) compared to the second-best entry. This dramatic improvement demonstrated the potential of deep learning for image recognition and sparked a revolution in the field.

Under the Hood Insights: Visualizing the learned filters in the first convolutional layer revealed that AlexNet learned to detect edges, corners, and color blobs. As one moves deeper into the network, the learned features become more complex and abstract, with later layers responding strongly to higher-level concepts like faces even though the dataset didn't explicitly label them as such. Furthermore, the feature vectors in the second to last layer formed an embedding space where images of similar concepts were located close to each other in the high-dimensional space. This indicated that AlexNet was learning meaningful semantic representations of images.

In conclusion, AlexNet's success was attributed to its deeper architecture, the strategic use of ReLU activation, the effective regularization provided by dropout and data augmentation, the efficiency gained from overlapping pooling and multi-GPU training, and the sheer scale of the ImageNet dataset and available compute power. It marked a significant shift in the AI community towards deep learning and paved the way for the rapid advancements seen in computer vision in the subsequent years.

VGGNet

VGGNet, introduced by the Visual Geometry Group (VGG) from Oxford, is a significant convolutional neural network (CNN) architecture known for its depth, ranging from 11 to 19 layers. It addressed the importance of network depth in achieving better performance in image recognition tasks. VGGNet was inspired by the AlexNet architecture, which demonstrated a breakthrough performance on the ImageNet challenge in 2012.

Motivation and Design Principles:

The primary motivation behind VGGNet was to investigate the effect of network depth on its performance in large-scale image recognition settings. Instead of varying other parameters significantly, the architects of VGGNet focused on steadily increasing the depth of the network by adding more convolutional layers. A key design choice in VGGNet was the use of very small convolutional filters of size 3x3 in all convolutional layers. This was in contrast to AlexNet, which used varying filter sizes like 11x11, 5x5, and 3x3. The 3x3 filter size was chosen as it is the smallest size that can still capture the notion of left, right, up, down, and center.

A stack of two 3x3 convolutional layers without any pooling in between has an effective receptive field of 5x5, and three such layers have a 7x7 effective receptive field. Using multiple stacked small filters has several advantages: it makes the decision function more discriminative due to the introduction of more non-linear rectified linear units (ReLUs), and it also reduces the number of parameters to be learned compared to using a single larger filter.

Architecture Details:

The VGGNet family consists of several configurations with varying depths, primarily VGG11, VGG13, VGG16, and VGG19. All these configurations follow a similar structure:

- Input Layer: The input to the network is typically a 224x224 RGB image.

- Convolutional Layers: A sequence of convolutional layers with 3x3 filters is used throughout the network. The stride is fixed to 1 pixel, and padding is used to preserve the spatial resolution after convolution (same padding). The number of filters starts small and increases as the network deepens. For example, it might start with 64 filters and increase to 128, 256, and finally 512.
- Max Pooling Layers: After some convolutional layers, max pooling layers with a size of 2x2 and a stride of 2 are inserted. These layers reduce the spatial dimensions of the feature maps by half, providing some translation invariance.
- Fully Connected Layers: The convolutional part of the network is followed by three fully connected (FC) layers. The first two FC layers typically have 4096 neurons each, and the final FC layer has 1000 neurons for classification on the ImageNet dataset (corresponding to the 1000 classes). A softmax activation function is used in the last fully connected layer to produce the class probabilities.
- ReLU Activation: All convolutional and fully connected layers (except the output layer) use the Rectified Linear Unit (ReLU) non-linearity. ReLU helps in mitigating the vanishing gradient problem, allowing for the training of deeper networks.

Specific Configurations (VGG16 and VGG19):

- VGG16 consists of 13 convolutional layers and 3 fully connected layers, totaling 16 weight layers. The arrangement of layers includes blocks of convolutional layers followed by max pooling. The number of filters in the convolutional layers typically increases as the depth increases: for instance, starting with 64 filters in the initial blocks, then 128, 256 (used in two blocks), and finally 512 (used in three blocks) before the fully connected layers.
- VGG19 is a deeper version with 16 convolutional layers and 3 fully connected layers, totaling 19 weight layers. It largely follows the same principles as VGG16 but with more convolutional layers in some of the blocks.

Key Characteristics:

- Uniform 3x3 Convolutional Filters: The consistent use of small 3x3 filters is a defining characteristic of VGGNet.
- 2x2 Max Pooling: Max pooling layers with a kernel size of 2x2 and a stride of 2 are used for spatial downsampling.
- Increased Depth: VGGNet demonstrated the benefit of significantly deeper networks for improved performance.
- ReLU Non-linearities: The use of ReLU activation functions throughout the network helps in training deeper models effectively.
- Fully Connected Layers at the End: Three fully connected layers are used for the final classification.

Advantages over AlexNet:

- Improved Accuracy: VGGNet achieved better accuracy on the ImageNet dataset compared to AlexNet, primarily due to its increased depth and the use of smaller, more uniform convolutional filters.
- More Discriminative Features: The stack of multiple 3x3 convolutional layers allows for learning more complex and discriminative features compared to the larger filters used in earlier layers of AlexNet.

- Simpler Architecture: Despite being deeper, VGGNet has a more homogeneous architecture compared to AlexNet, making it potentially easier to understand and implement. AlexNet had more variations in filter sizes and included local response normalization, which was later found to be not very effective in practice.

Training and Performance:

VGGNets were trained on the large-scale ImageNet dataset with 1000 classes. The training procedure involved stochastic gradient descent with momentum and L2 weight decay for regularization. Data augmentation techniques, such as random cropping and horizontal flipping, were also employed to increase the training data size and improve generalization. The deeper VGG16 and VGG19 networks generally outperformed the shallower versions (VGG11 and VGG13), confirming the benefit of increased depth.

Implementation Details:

VGGNet architectures are readily available in deep learning libraries like Keras and PyTorch. These libraries provide pre-trained weights on the ImageNet dataset, which can be used for transfer learning on other image classification tasks. When using pre-trained VGG models for transfer learning, it's common to remove the top (classification) layer and replace it with a new fully connected layer suitable for the specific task's number of classes. The convolutional layers can be kept frozen (non-trainable) to leverage the features learned on ImageNet, or they can be fine-tuned on the new dataset.

In Keras, you can easily load VGG16 or VGG19 with pre-trained weights using `tf.keras.applications.VGG16` or `tf.keras.applications.VGG19`, specifying `weights='imagenet'` and `include_top=False` to exclude the original classification layer. Similarly, in PyTorch, the `torchvision.models` module provides implementations of VGG architectures with options for pre-trained weights.

Variations (VGG16 vs. VGG19):

The main difference between VGG16 and VGG19 is the number of convolutional layers. VGG19 has a few more convolutional layers compared to VGG16, particularly in the blocks with a larger number of filters. While VGG19 is slightly more accurate due to its increased depth, it also has more parameters and requires more computational resources for training and inference. In many practical scenarios, VGG16 is often preferred due to its slightly lower computational cost while still providing excellent performance.

VGGNet was a pivotal architecture in the history of deep learning for computer vision. It convincingly demonstrated the importance of network depth for achieving state-of-the-art performance in image recognition. Its simple and consistent architecture, characterized by the use of small 3x3 convolutional filters, has made it a widely adopted and influential model. Even with the emergence of more recent and efficient architectures, VGGNet remains a valuable concept for understanding the principles of deep CNNs and serves as a strong baseline for various computer vision tasks, especially when using transfer learning.

Residual Networks (ResNets)

Residual Networks (ResNets) are a crucial architecture in modern deep learning, particularly for convolutional neural networks (CNNs), and were introduced in 2015. They gained significant attention by winning the ImageNet challenge in 2015 and achieving performance surpassing human capabilities in image classification.

The Core Idea: Learning Residuals Instead of directly learning a complex mapping from an input X to a desired output Y, ResNets are based on the idea of learning the residual or the difference between X and Y. The network tries to model a function F(X) such that the desired output Y can be represented as Y = X + F(X). The intuition behind this is that in many tasks, the output Y already contains a lot of information from the input X, and it's easier for the network to learn the small changes or "residual" F(X) needed to get from X to Y. A super-resolution example effectively illustrates this, where the high-resolution output Y contains most of the information present in the low-resolution input X, and the network only needs to learn the fine details F(X) to achieve super-resolution. This approach also aligns with the idea in applied mathematics and computational science where efforts are often focused on modeling the unknown difference between known quantities.

The Problem ResNets Solve: Training Deep Networks Prior to ResNets, training very deep neural networks (with many layers) was challenging due to issues like the vanishing gradient problem and the difficulty in the network remembering the initial input over many layers. As networks became deeper, performance would often degrade, and it was hard for gradients to effectively propagate back through the network to update the weights, especially in the earlier layers. The input signal could also get lost or "forgotten" as it passed through numerous layers.

The Solution: Skip Connections (or Jump Connections) The key innovation in ResNets is the introduction of skip connections (also called jump connections). These connections allow the input of a block of layers to be directly added to the output of that block. This creates a shortcut, allowing data and gradients to bypass some layers.

Here's how it works within a residual block:

- The input X enters the block.
- It goes through a series of layers, typically including convolutional layers, normalization, and activation functions (like ReLU). Let's say the function learned by these layers is F(X).
- The original input X is then added to the output of these layers, resulting in X + F(X).
- This sum might then go through a final activation function (like ReLU).

Benefits of Skip Connections:

- Addresses Vanishing Gradients: Skip connections provide a more direct path for gradients to flow backward through the network, mitigating the vanishing gradient problem that occurs in very deep networks. Even if the gradients through the main processing layers become small, the gradient can still flow effectively through the identity connection (the original input).

- Enables Training of Much Deeper Networks: By alleviating the gradient issue and the problem of forgetting the input, ResNets made it possible to train networks with hundreds of layers (e.g., 152 layers in some ResNet architectures), significantly deeper than what was achievable before. Deeper networks have a greater capacity to learn complex features.
- Simplifies Learning for Residual Blocks: Each residual block's task becomes learning the residual F(X), which is often a smaller and easier function to learn than the entire mapping from input to output. If a block of layers is not needed, it can essentially learn an identity function ($F(X) \approx 0$), allowing the input to pass through unchanged.
- Improved Backpropagation: The skip connections facilitate better backpropagation of loss errors through the deep network.
- Modularity: Residual blocks have a consistent structure, making it easier to build and experiment with networks of varying depths by simply adding or removing blocks.

ResNet Architecture and Variations: A typical ResNet architecture consists of stacking many residual blocks. While the basic idea of a residual block is consistent, the specific layers within the block (number of convolutional layers, filter sizes, activation functions, normalization) can vary.

ResNets can be adapted for different tasks:

- Image-to-Image Tasks (e.g., Super-resolution): Residual blocks with convolutional layers using a stride of 1 and padding of 1 can maintain the input and output dimensions (height, width, and number of channels). In the super-resolution example, the low-resolution input can be conceptually upscaled to the desired high-resolution size (e.g., by repeating pixel values) so that the dimensions match.
- Classification Tasks: To reduce the spatial dimensions (height and width) and increase the number of feature channels, convolutional layers within the residual blocks can use a stride greater than 1 (e.g., stride of 2). This downsampling process helps to extract more abstract features for classification.

Connections to Other Concepts:

- Numerical Integration: The structure of a residual block, where the output is the input plus a small change (the residual), bears a resemblance to the Euler method for numerical integration of differential equations. If we consider the layers of a ResNet as discrete time steps, the residual can be seen as modeling the change in the state of a system over a small time interval.
- Neural Ordinary Differential Equations (Neural ODEs): The connection to numerical integration has further inspired the development of Neural ODEs. Instead of having discrete residual blocks, Neural ODEs model the continuous flow of data through a neural network defined by an ordinary differential equation. ResNets can be seen as a discrete approximation (using the Euler method) of such a continuous flow.

Considerations When Building ResNets:

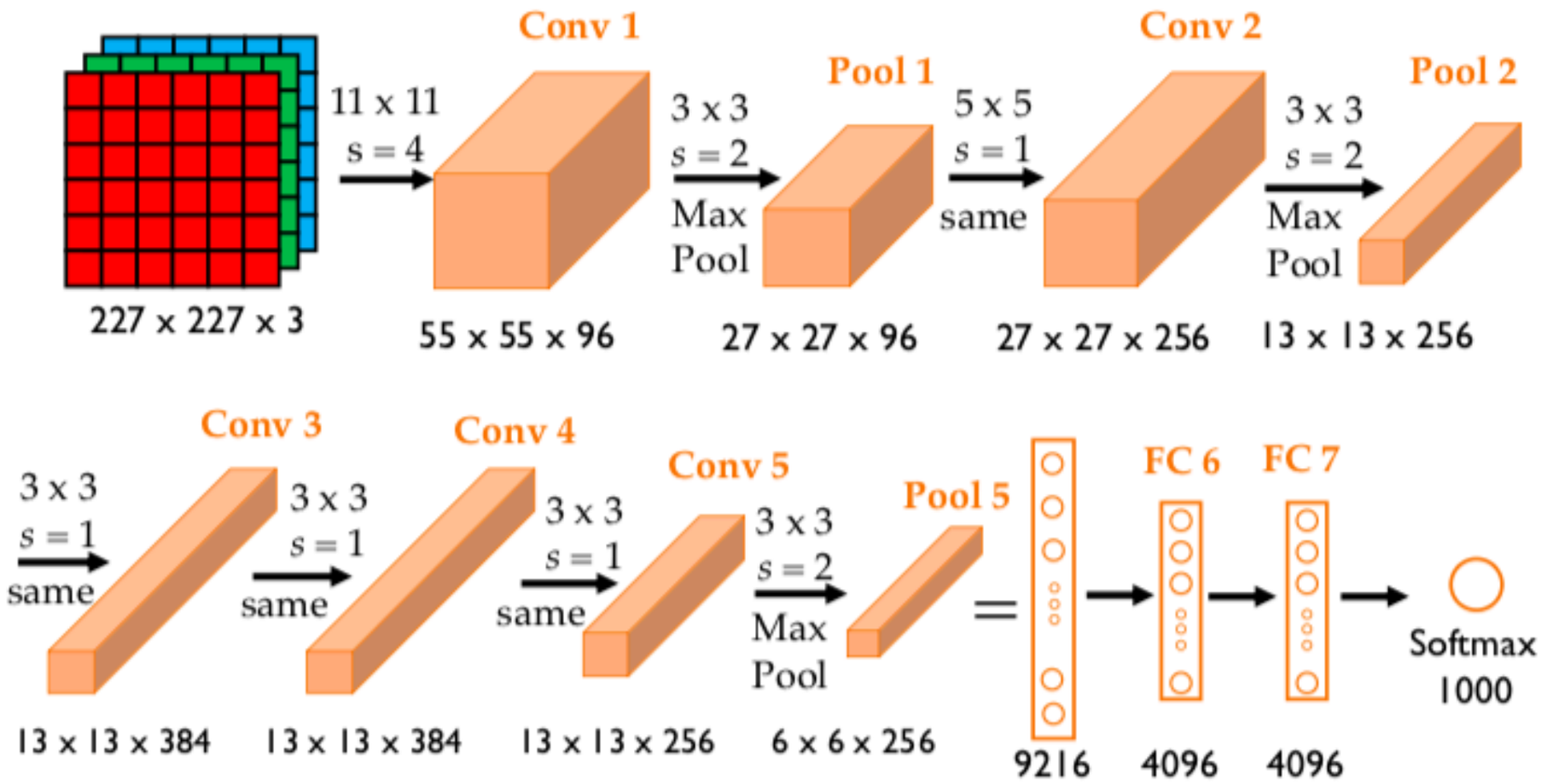
- Shape Matching: When adding the input to the output of a residual block (using addition or concatenation), their dimensions must be compatible. For convolutional layers, this means the height, width, and number of channels should often match. One-by-one convolutional layers can be used to adjust the number of channels to ensure compatibility.

- Addition vs. Concatenation: While both addition and concatenation can be used to combine the input and the block's output in skip connections, addition is generally preferred in deep ResNets to avoid an explosion in the size of the activation tensors and the number of parameters that can occur with repeated concatenation. Concatenation might be used sparingly.

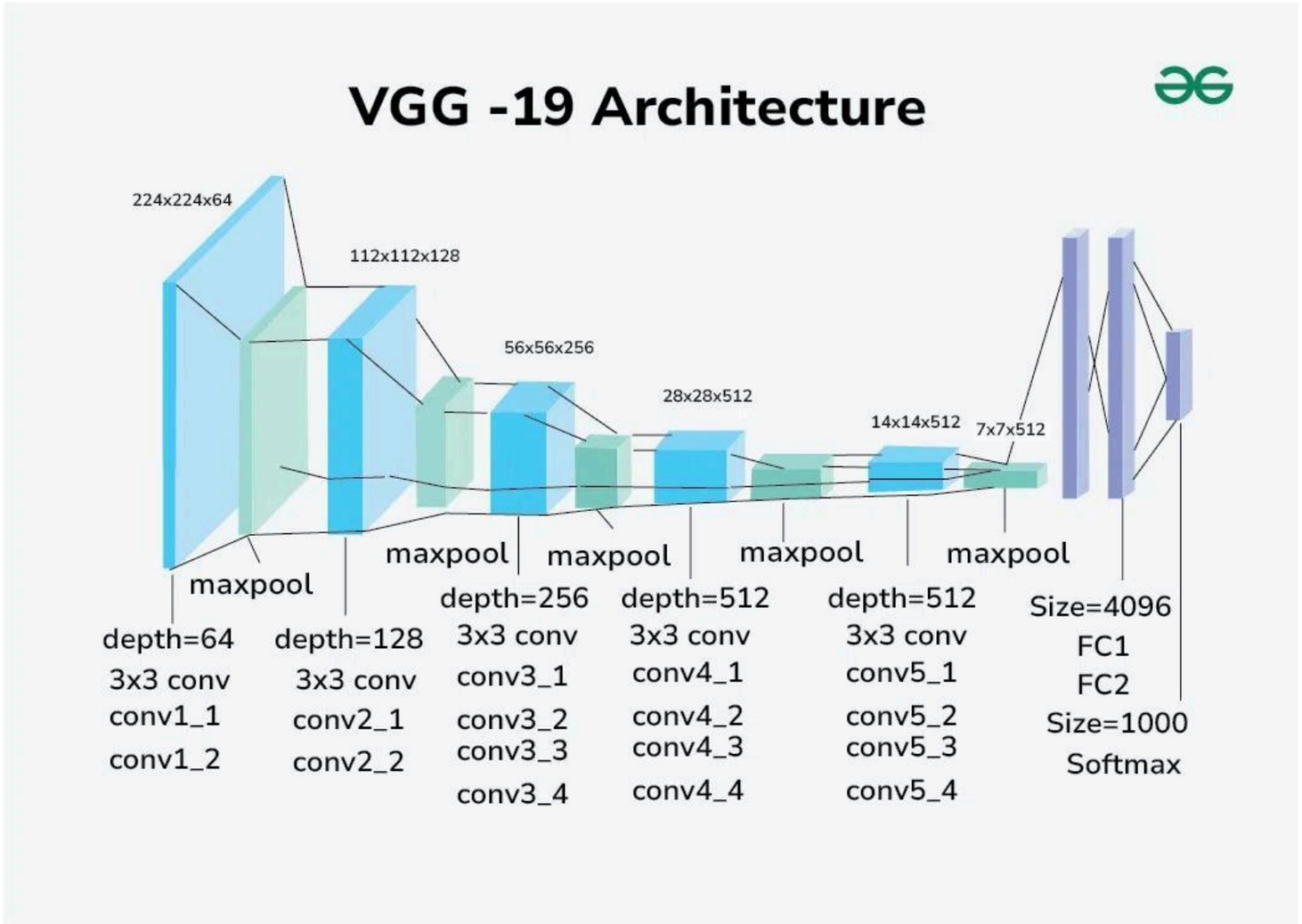
In summary, ResNets revolutionized the field of deep learning by introducing the concept of residual learning and skip connections. This innovation effectively addressed the challenges of training very deep networks, leading to significant performance improvements in various computer vision tasks and inspiring further research into the connections between neural networks and dynamical systems.

	CNN Architectures		
Comparison	**AlexNet (2012)**	**VGGNet (2014)**	**ResNet (2015)**
Input Layer	227x227x3 (RGB image).	224x224x3.	1. **Initial Layers**: • **Conv1**: 7x7 kernel, 64 filters, stride 2 → 112x112x64. • **Max Pool**: 3x3 kernel, stride 2 → 56x56x64. 2. **Residual Blocks**: • **Basic Block** (ResNet-34): 2×[Conv3x3, ReLU, BatchNorm]. • **Bottleneck Block** (ResNet-50+): 1×1 (reduce) → 3x3 → 1x1 (expand). • **Skip Connection**: Identity mapping added to block output. 3. **Global Average Pooling**: Replaces FC layers for parameter reduction. 4. **Output Layer**: 1000 neurons (softmax).
Convolutional Layers	• **Conv1**: 96 filters, 11x11 kernel, stride 4, ReLU → Output: 55x55x96. • **Max Pool1**: 3x3 kernel, stride 2 → Output: 27x27x96. • **Conv2**: 256 filters, 5x5 kernel, padding 2, ReLU → Output: 27x27x256. • **Max Pool2**: 3x3 kernel, stride 2 → Output: 13x13x256. • **Conv3**: 384 filters, 3x3 kernel, padding 1, ReLU → Output: 13x13x384. • **Conv4**: 384 filters, 3x3 kernel, padding 1, ReLU → Output: 13x13x384. • **Conv5**: 256 filters, 3x3 kernel, padding 1, ReLU → Output: 13x13x256. • **Max Pool3**: 3x3 kernel, stride 2 → Output: 6x6x256.	• **Block1**: 2× [Conv3×3, 64 filters] → Max Pool2×2 → 112×112×64. • **Block2**: 2× [Conv3×3, 128 filters] → Max Pool2×2 → 56×56×128. • **Block3**: 3× [Conv3×3, 256 filters] → Max Pool2×2 → 28×28×256. • **Block4**: 3× [Conv3×3, 512 filters] → Max Pool2×2 → 14×14×512. • **Block5**: 3× [Conv3×3, 512 filters] → Max Pool2×2 → 7×7×512.	

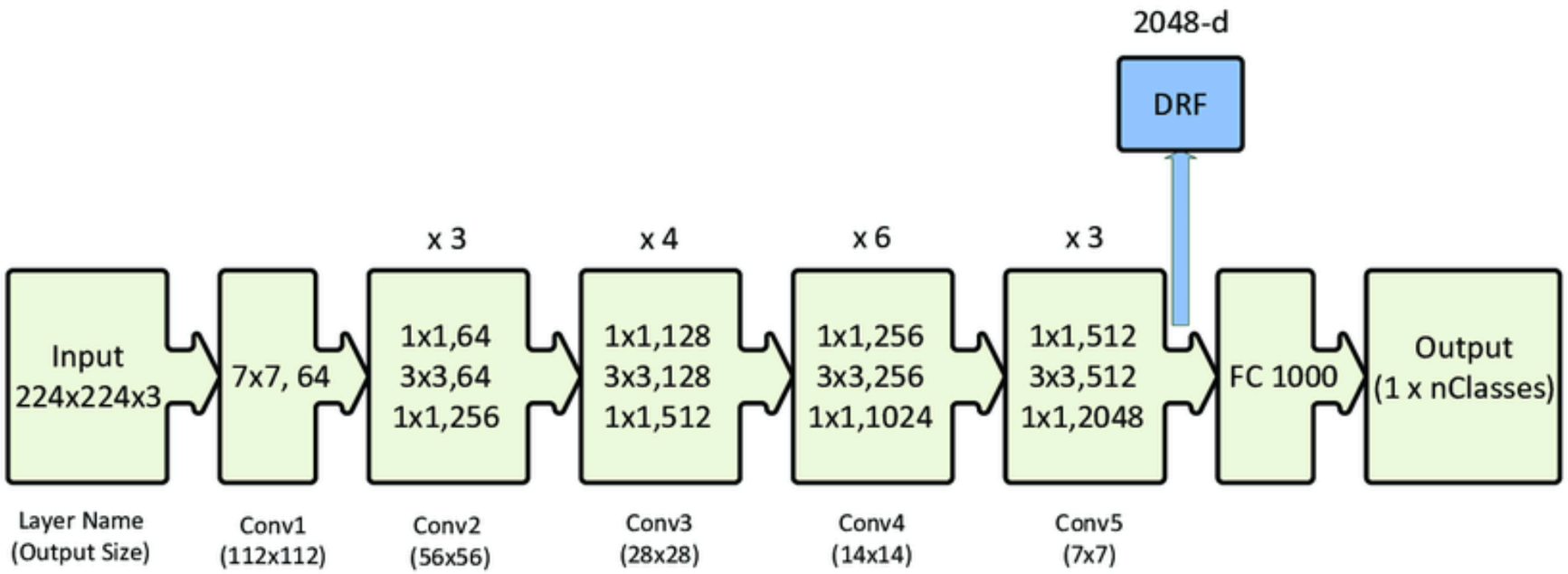
Fully Connected (FC) Layers	• **FC1**: 4096 neurons, ReLU, Dropout (0.5). • **FC2**: 4096 neurons, ReLU, Dropout (0.5). • **FC3**: 1000 neurons (softmax for ImageNet classes).	• **FC1-2**: 4096 neurons, ReLU, Dropout (0.5). • **FC3**: 1000 neurons (softmax).	
Characteristics	• ReLU Activation: Faster convergence compared to tanh/sigmoid. • Overlapping Pooling: Reduces overfitting. • Dropout: Regularization in FC layers. • Multi-GPU Training: Split across 2 GPUs for efficiency.	• **Uniform 3×3 Kernels**: Increased depth with small receptive fields. • **Depth**: 16/19 layers (VGG16/VGG19) for better feature extraction. • **Simplicity**: Repetitive structure eases implementation.	• **Skip Connections**: Solve vanishing gradients; enable 100+ layers. • **Batch Normalization**: Stabilizes training. • **Efficiency**: Fewer parameters than VGG (e.g., ResNet-50: 25M vs. VGG16: 138M).
Drawbacks	• High Memory Usage: Large kernel sizes (11x11, 5x5) in early layers. • Limited Depth: Only 8 learned layers; outperformed by deeper networks. • Computational Cost: Inefficient by modern standards.	• **Parameter Explosion**: 138M parameters (mostly in FC layers). • **Computationally Heavy**: Inefficient for real-time applications. • **Vanishing Gradients**: Hard to train deeper variants (e.g., VGG19).	• **Complexity**: Careful design needed for skipping connections. • **Redundancy**: Some blocks may learn trivial residuals.
Pictorial Explanation	Input → Conv11x11 → Pool → Conv5x5 → Pool → Conv3x3 → Conv3x3 → Conv3x3 → Pool → [FC → Dropout] ×2 → Output	Input → [Conv3x3×2 → Pool] ×2 → [Conv3x3×3 → Pool] ×3 → [FC×2 → Dropout] → Output	Input → Conv7x7 → Pool → [Residual Blocks×N] → Avg Pool → Output Residual Block: Input → Conv → ReLU → Conv → +Input → ReLU

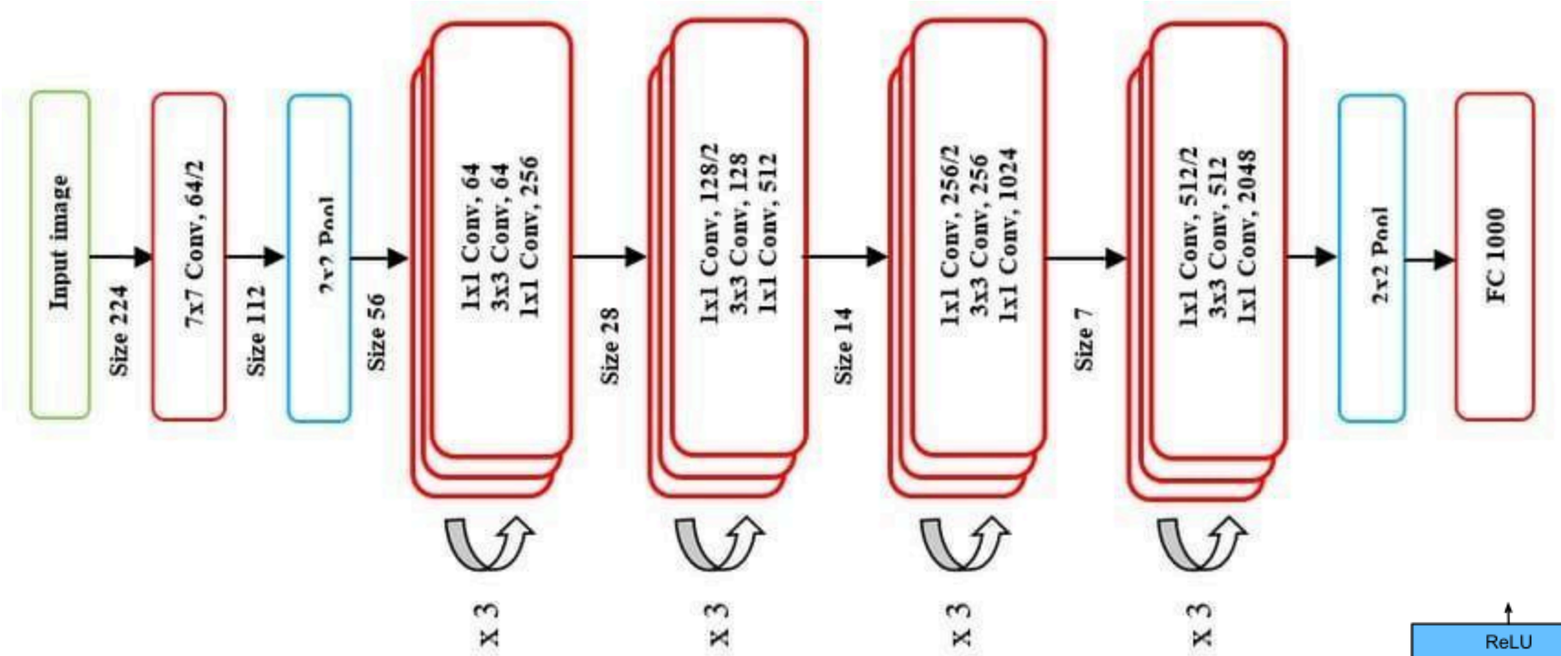


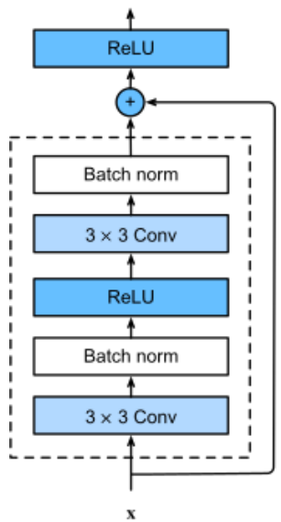
AlexNet Architecture

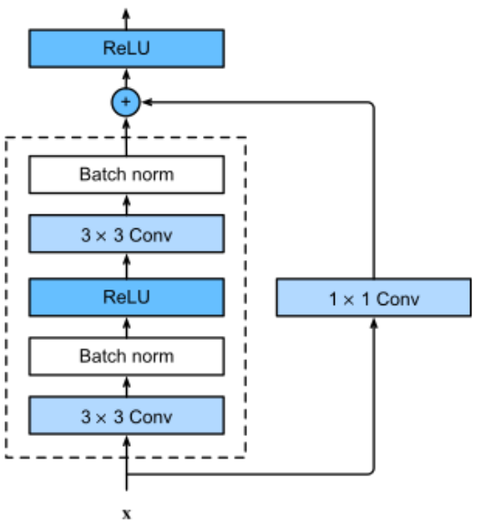


VGGNet Architecture


2048-d
DRF
x 3
x 4
x 6
x 3
Input
224x224x3
7x7, 64
1x1,64
3x3,64
1x1,256
1x1,128
3x3,128
1x1,512
1x1,256
3x3,256
1x1,1024
1x1,512
3x3,512
1x1,2048
FC 1000
Output
(1 x nClasses)
Layer Name
(Output Size)
Conv1
(112x112)
Conv2
(56x56)
Conv3
(28x28)
Conv4
(14x14)
Conv5
(7x7)







ResNet Architecture