UNDERSTANDING CNNS

Understanding AlexNet: A Detailed Walkthrough

In this article, we explore AlexNet, a CNN developed by Alex Krizhevsky and others in 2012.

Azel Daniel Sep 24, 2020 · 9 min read ★

Figure 1: In the background — AlexNet General Architecture (image by author)

Table of Contents

. . .

1. An Overview of AlexNet

AlexNet was a deep neural network that was developed by Alex Krizhevsky and others in 2012. It was designed to classify images for the ImageNet LSVRC-2010 competition, where it achieved state of the art results [1]. It also worked with multiple GPUs.

. . .

2. Convolutional Neural Network Basics

Convolutional Neural Networks were initially designed to mimic the biological processes of human vision. The networks utilised multiple layers that were ordered in different ways to form distinct network architectures. The three types of layers that traditional neural networks included were:

1. Convolution Layers

2. Subsampling Layers

3. Fully Connected Layers

Convolution Layers

Two-dimensional convolution layers used trainable *kernels* or filters to perform *convolution* operations, sometimes including an optional trainable *bias* for each kernel. These convolution operations involved *moving* the kernels over the input in steps called *strides*. Generally, the larger the stride was, the more spaces the kernels skipped between each convolution. This led to less overall convolutions and more miniature output size. For each placement of a given kernel, a multiplication operation was performed between the input *section* and the kernel, with the bias summed to the result. This produced a *feature map* containing the convolved result. The feature maps were typically passed through an activation function to provide input for the subsequent layer. The size of the feature map was calculated as [(input_size − kernel_size + 2 × padding) / stride] + 1.

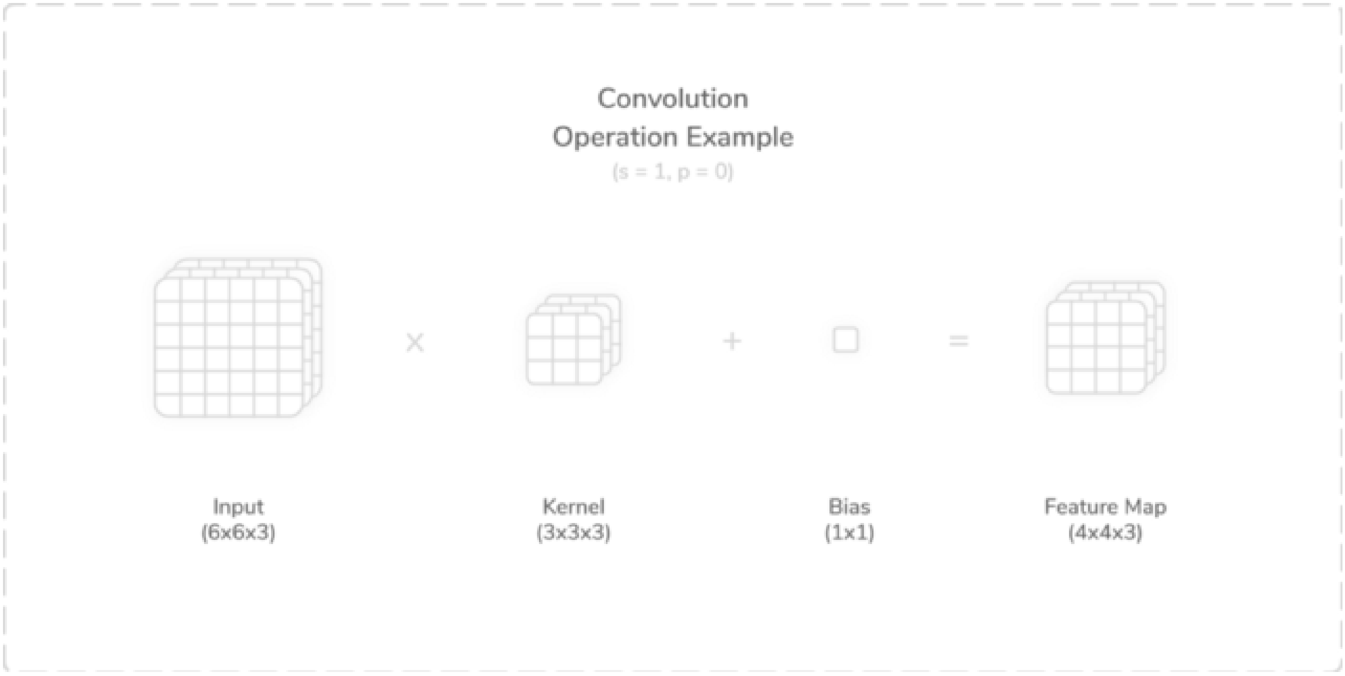
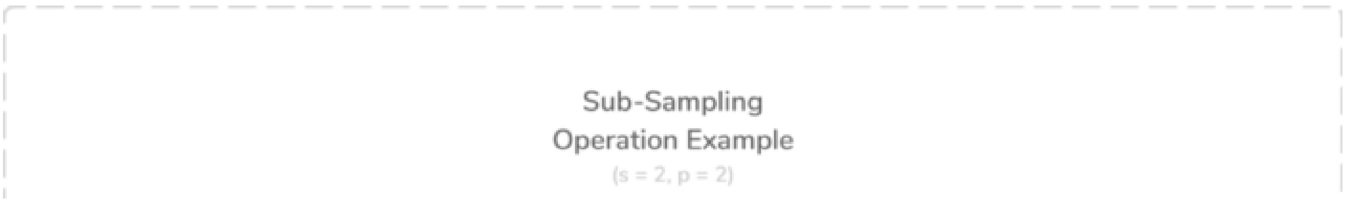


Figure 2: Convolution Operation Example (image by author)

Figure 2 shows a convolution operation that involved a 3-channel (6×6) image, a 3-channel (3×3) kernel and a (1×1) bias. The operation was implemented with a stride of one and zero padding. This meant that the kernel moved over each (3×3) section of the image in an overlapping left-to-right motion, shifting one place between each operation. This resulted in a 3-channel (4×4) convolved feature map. Generally, the number of kernel channels was always identical to the number of input channels.

Subsampling Layers

Two-dimensional subsampling layers used non-trainable kernels or *windows* to down-sample input features. This typically reduced the size of the features significantly and helped to remove a network's position reliance. There were two conventional types of subsampling: Average Pooling and Max Pooling. Both methods computed either the average or maximum of the values present in each kernel to be included in the resulting feature map. The size of the feature map for subsampling layers was calculated the same as convolution layers. Some implementations of these layers included some trainable parameters to aid with overall model learning.



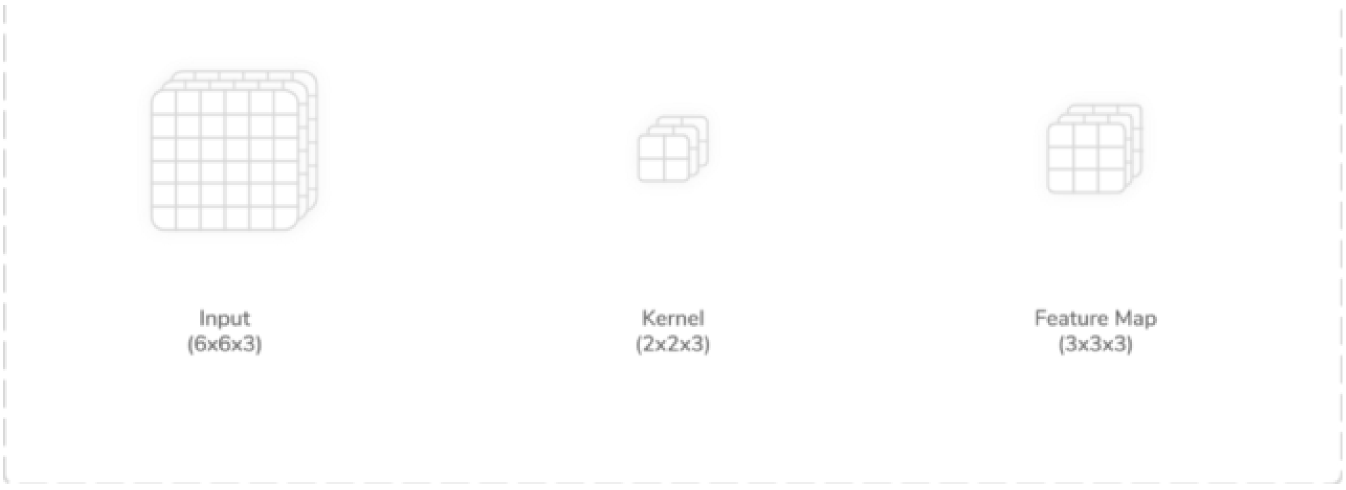


Figure 3: Subsampling Operation Example (image byauthor)

Figure 3 shows a subsampling operation that involved a 3-channel (6×6) image and a 3-channel (2×2) kernel. The operation was implemented with a stride of two and zero padding. This meant that the kernel moved over each (2×2) input section in a non-overlapping motion, shifting two places between each operation. This resulted in a 3-channel (3×3) down-sampled feature map.

Fully Connected Layers

Fully connected layers were ununique to CNNs. However, they typically were included as the last few layers of most CNNs, appearing after several convolution and subsampling operations were performed. Fully connected layers were independent neural networks that possessed one or more hidden layers. Their operations involved multiplying their inputs by trainable *weight* vectors, with a trainable *bias* sometimes summed to those results. The output of these layers was traditionally sent through activation functions, similarly to convolution layers.

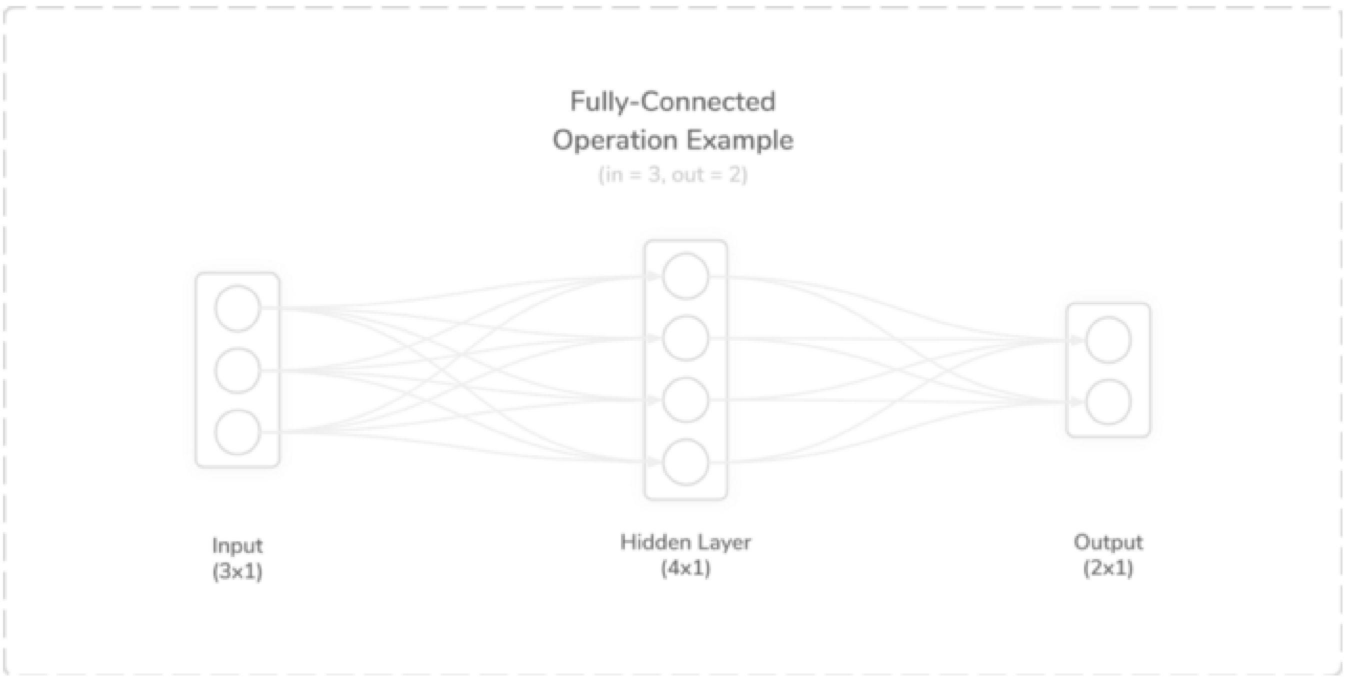


Figure 4: Fully -Connected Operation Example (image y author)

Figure 4 shows a fully connected operation that involved a (3×1) input, a (4×1) hidden layer and a (2×1) output.

. . .

3. A Walkthrough of AlexNet's Architecture

AlexNet was classified as a deep convolutional network and it was built to accommodate coloured images of size (224x224x3). It boasted a total of over 62 million trainable parameters.

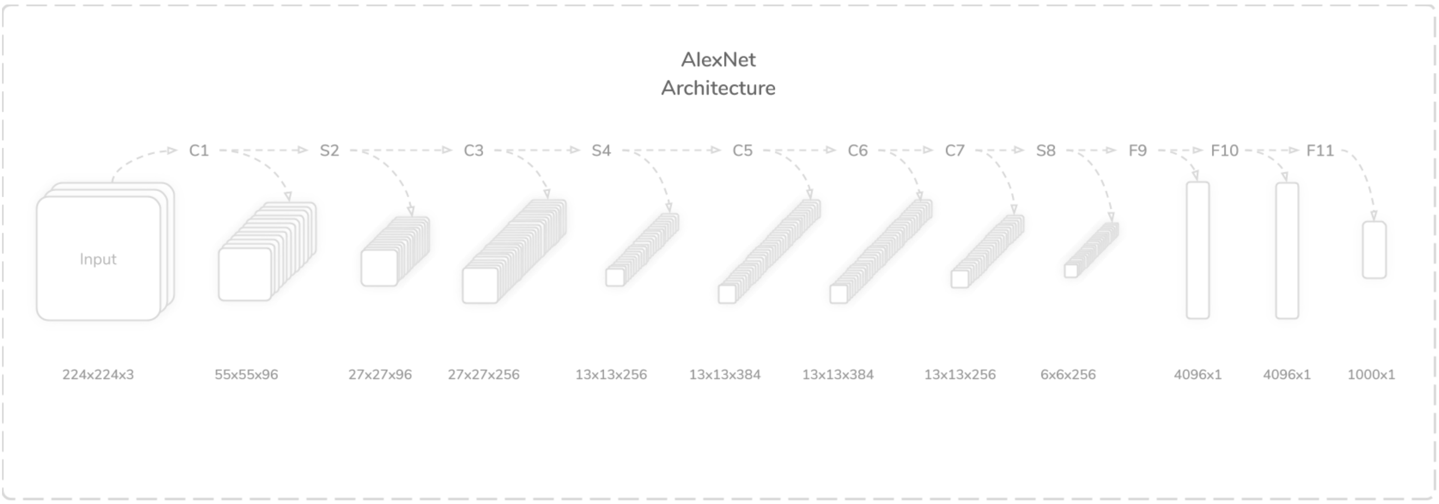


Figure 5: AlexNet Architecture Diagram (image by author)

The 11 layers of AlexNet were:

1. Layer C1: Convolution Layer (96, 11×11)
2. Layer S2: Max Pooling Layer (3×3)
3. Layer C3: Convolution Layer (256, 5×5)
4. Layer S4: Max Pooling Layer (3×3)
5. Layer C5: Convolution Layer (384, 3×3)
6. Layer C6: Convolution Layer (384, 3×3)
7. Layer C7: Convolution Layer (256, 3×3)

8. Layer S8: Max Pooling Layer (3×3)
9. Layer F9: Fully-Connected Layer (4096)
10. Layer F10: Fully-Connected Layer (4096)
11. Layer F11: Fully-Connected Layer (1000)

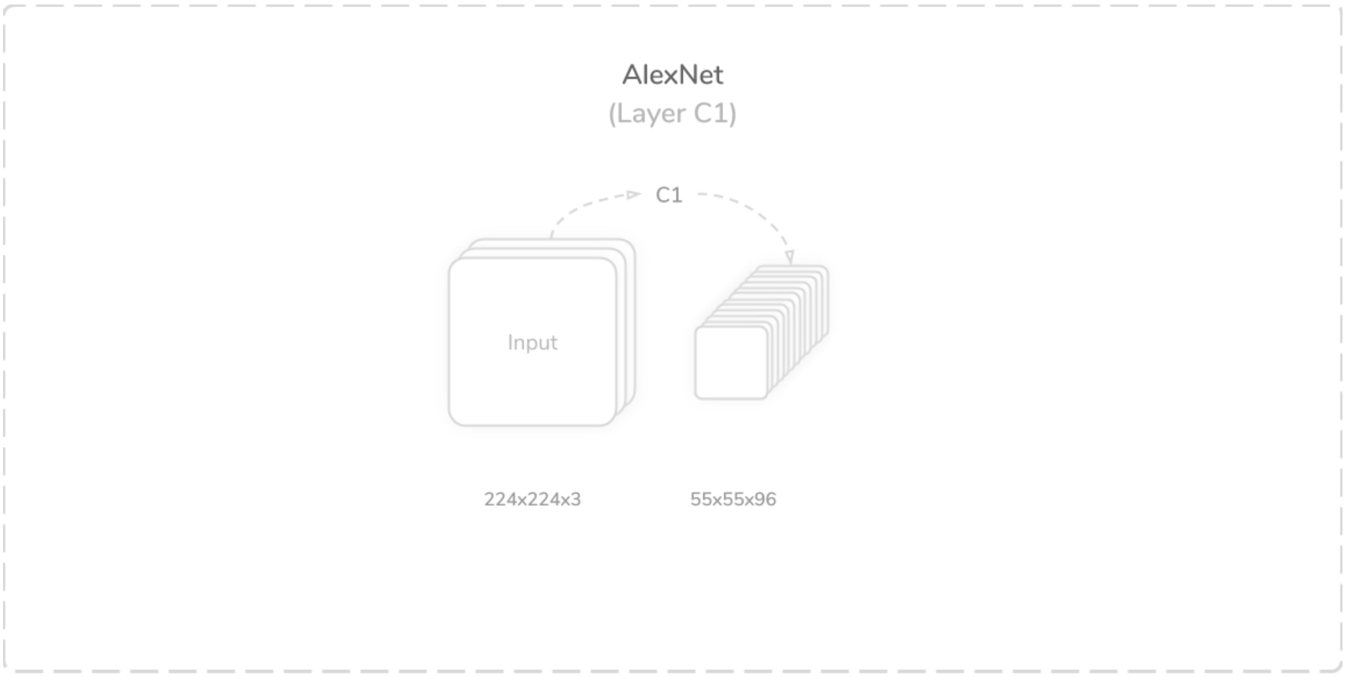


Figure 6: AlexNet C1 — First Convolution Layer Diagram (image by author)

C1: First Convolution Layer

The first layer of AlexNet was a convolutional layer that accepted a (224×224×3) image tensor as its input. It performed a convolution operation using 96 (11×11) kernels with a stride of four and a padding of two. This produced a (55×55×96) output tensor that was then passed through a ReLu activation function then on to the next layer. The layer contained 34,944 trainable parameters.

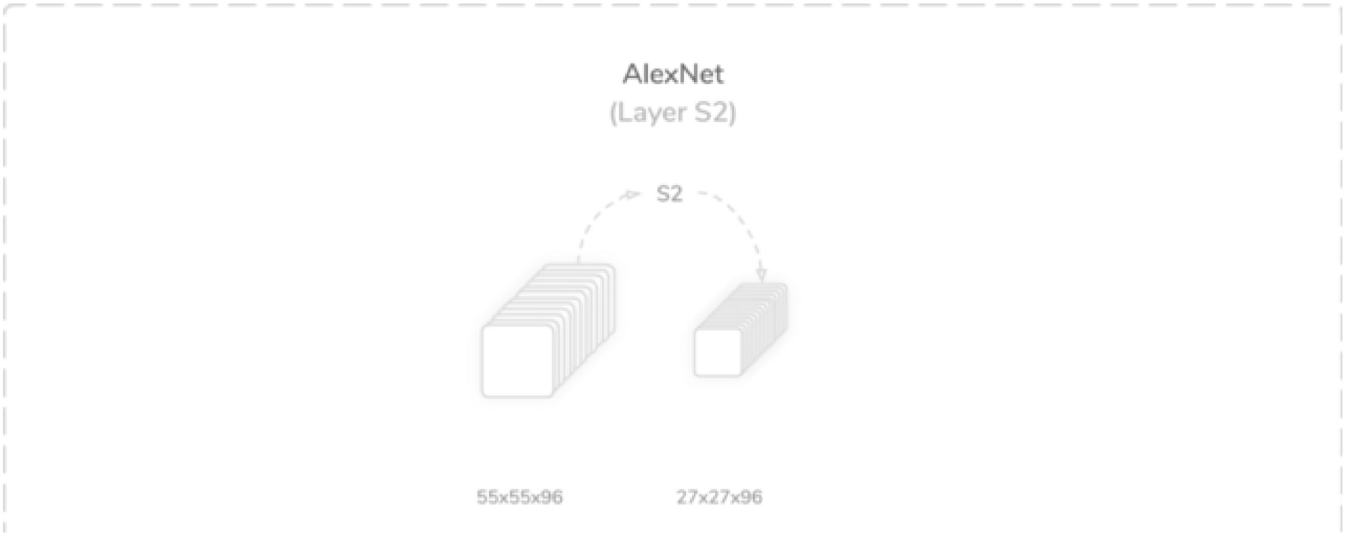


Figure 7: AlexNet S2 — First Max Pooling Layer Diagram (image by author)

S2: First Max Pooling Layer

The second layer of AlexNet was a max-pooling layer that accepted output from the layer C1, a (55×55×96) tensor, as its input. It performed a zero-padded sub-sampling operation using a (3×3) kernel with a stride of two. This produced a (27×27×96) output tensor that was passed on to the next layer.

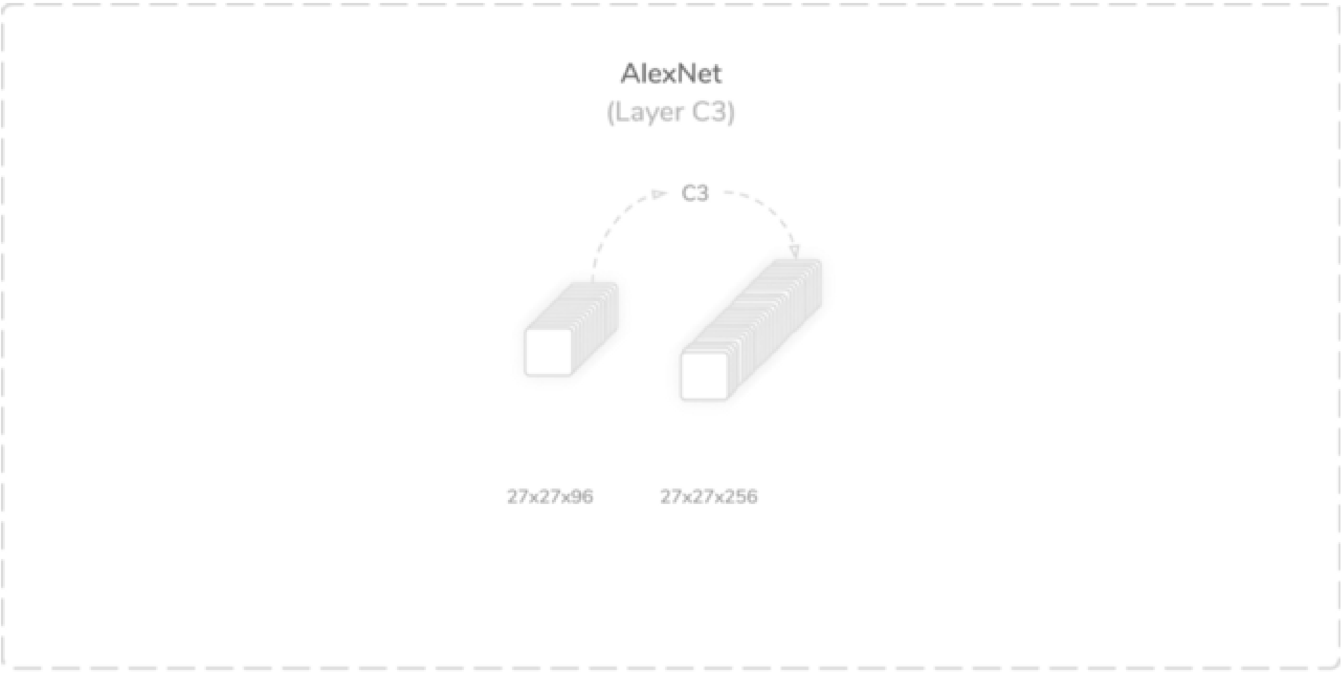
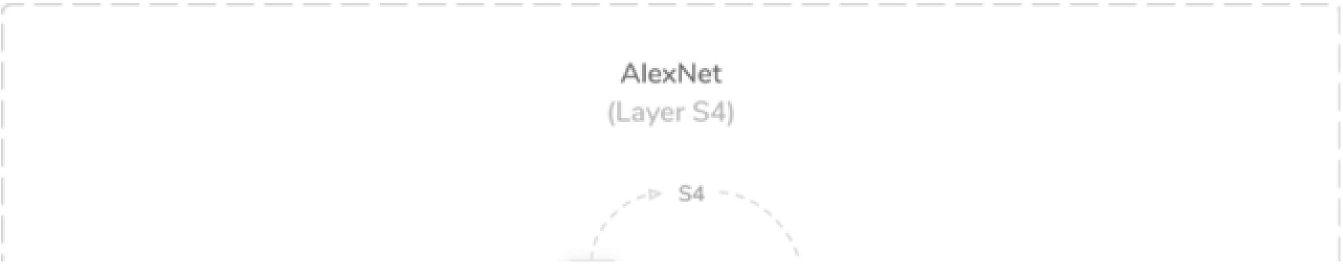


Figure 8: AlexNet C3 — Second Convolution Layer Diagram (image by author)

C3: Second Convolution Layer

The third layer of AlexNet was another convolutional layer that accepted output from the layer S2, a (27×27×96) tensor, as its input. It performed a convolution operation using 256 (5×5) kernels with a stride of one, and a padding of two. This operation produced a (27×27×256) output tensor that was then passed through a ReLu activation function, and then on to the next layer. The layer contained 614,656 trainable parameters, which summed to a total of 649,600 trainable parameters so far.



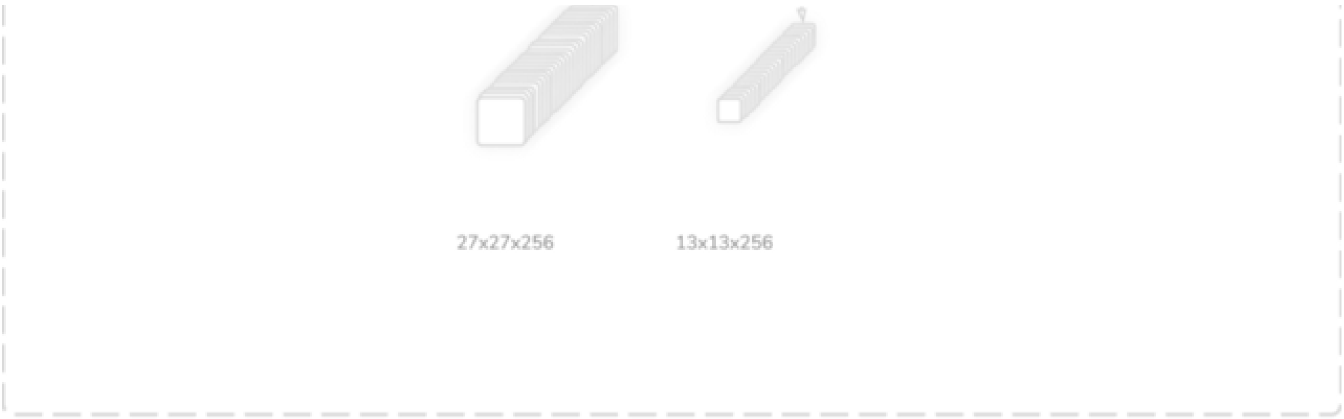


Figure 9: AlexNet S4 — Second Max Pooling Layer Diagram (image by author)

S4: Second Max Pooling Layer

The fourth layer of AlexNet was a max-pooling layer that accepted output from the layer C3, a (27×27×256) tensor, as its input. It performed a zero-padded sub-sampling operation using a (3×3) kernel with a stride of two, similar to layer S2. This produced a (13×13×256) output tensor that was then passed through a ReLU activation function, and then on to the next layer.

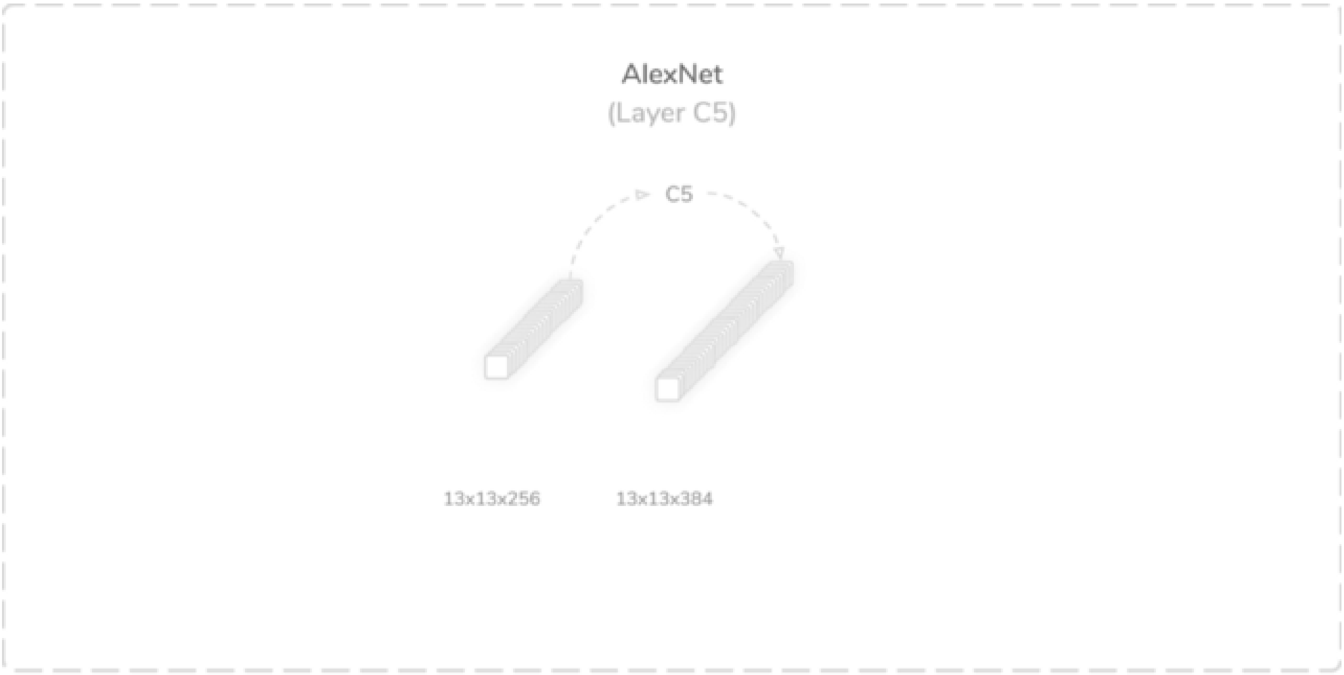


Figure 10: AlexNet C5 — Third Convolution Layer Diagram (image by author)

C5: Third Convolution Layer

The fifth layer of AlexNet was another convolutional layer that accepted output from the layer C5, a (13×13×256) tensor, as its input. It performed a convolution operation using 384 (3×3) kernels with a stride and padding of one. This produced a (13×13×384) output tensor that was then passed through a ReLu activation function, and then on to the next layer. The layer contained 885,120 trainable parameters, which summed to a total of 1,534,720 trainable parameters so far.

Figure 11: AlexNet C6 — Fourth Convolution Layer Diagram (image by author)

C6: Fourth Convolution Layer

The sixth layer of AlexNet was another convolutional layer that accepted output from the layer C5, a (13×13×384) tensor, as its input. It performed the same convolution operation as layer C5, which lead to the same output size. The output was also passed through a ReLu activation function. The layer contained 1,327,488 trainable parameters, which summed to a total of 2,862,208 trainable parameters so far.

Figure 12: AlexNet C7 — Fifth Convolution Layer Diagram (image by author)

C7: Fifth Convolution Layer

The seventh layer of AlexNet was another convolutional layer that accepted a (13×13×384) tensor from the layer C6 as its input. It performed a convolution operation using 256 (3×3) kernels with a stride and padding of 1. This produced a (13×13×256) output tensor. The output was also passed through a ReLu activation function. The layer contained 884,992 trainable parameters, which summed to a total of 3,747,200 trainable parameters so far.

Figure 13: AlexNet S8 — Third Max Pooling Layer Diagram (image by author)

S8: Third Max Pooling Layer

The eighth layer of AlexNet was a max pooling layer that accepted a (13×13×256) tensor from the layer C7 as its input. It performed a zero-padded sub-sampling operation using a (3×3) window region with a stride of two, similar to layer S2 and S4. This produced a (6×6×256) output tensor that was then passed through a ReLU activation function, and then on to the next layer.

Figure 14: AlexNet F9 — First Fully Connected Layer Diagram (image by author)

F9: First Fully Connected Layer

The ninth layer of AlexNet was a fully connected layer that accepted a flattened (6×6×256) tensor from the layer S8 as its input. It performed a weighted sum operation with an added bias term. This produced a (4096×1) output tensor that was then passed through a ReLu activation function, and on to the next layer. The layer contained 37,752,832 trainable parameters, which summed to a total of 41,500,032 trainable parameters so far.

Figure 15: AlexNet F10 — Second Fully Connected Layer Diagram (image by author)

F10: Second Fully Connected Layer

The tenth layer of AlexNet was another fully connected layer that accepted a (4096×1) tensor from the layer F9 as its input. It performed the same operation as layer F9 and produced the same (4096×1) output tensor that was then passed through a ReLu activation function, and then on to the next layer. The layer contained 16,781,312 trainable parameters, which summed to a total of 58,281,344 trainable parameters so far.

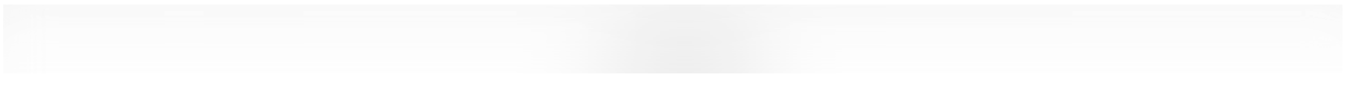


Figure 16: AlexNet F11 — Third Fully Connected Layer Diagram (image by author)

F11: Third Fully Connected Layer

The eleventh and final layer of the network was also a fully connected layer that accepted a (4096×1) tensor from the layer F10 as its input. It performed the same operation as layer F9 and F10 and produced a (1000×1) output tensor that was then passed through a softmax activation function. The layer contained 4,097,000 trainable parameters, which summed to a total of 62,378,344 trainable parameters overall. The output of the softmax activation function contained the predictions of the network.

. . .

4. Analysis of AlexNet

AlexNet was trained on the ImageNet LSVRC-2010 dataset using a batch size of 128, a momentum of 0.9, and a weight decay of 0.0005[1]. The model used a zero-mean Gaussian distribution initialiser with a standard deviation of 0.01[1]. Dropout was also used in the first two fully-connected layers to reduce overfitting. While AlexNet was undoubtably a breakthrough algorithm for its time, for it to work as intended, it requires the use of at least two GPUs. Not many individuals may have access to multiple GPUs.

. . .

5. Summary of AlexNet

To summarise, the following are key takeaways about the AlexNet network:

1. Operated with 3-channel images that were (224×224×3) in size.
2. Used max pooling and ReLU activations when subsampling.
3. Used ReLu activations for convolutions.
4. Used (3×3) kernels for max pooling.
5. Used either (11×11), (5×5), or (3×3) kernels for convolutions.
6. Classified images into one of 1000 classes.

. . .

Next Steps

Now that you have this knowledge about AlexNet, you can use prebuilt AlexNet models, or even build one yourself with libraries such as PyTorch or Keras. You can train AlexNet on the ImageNet database or one of the many other large scale image classification datasets.

References

[1] A. Krizhevsky, S. Ilya, and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks (2012), In Advances in Neural Information Processing Systems 25 (NIPS 2012)

Data Science Machine Learning Artificial Intelligence Convolutional Network