Hand Crafted CNN

by

Jean Vigroux

A Master's Project Report submitted to the
faculty of the Graduate School of
the University at Buffalo, The State University of New York
in partial fulfillment of the requirements for the
degree of

Department of Computer Science and Engineering

Abstract

This project implements a convolutional neural network (CNN) from scratch in C++ and CUDA, with the goal of learning more about GPU parallelism and neural network concepts. The network is a five-layer architecture trained on Fashion-MNIST. Three versions were created: a sequential C++ baseline, Single GPU CUDA Implementation, and a multi-GPU Implementation. Performance was measured using strong-scaling tests. On a single GPU, inference reaches a peak speedup of ~59x over a single-thread. Training, on the other hand, saw the speedup peak at 4.7x. On multiple GPUs, the peak training speedup was 6.23x with a relatively strong efficiency throughout. This illustrates well the cost of communication as well as Amdahl's Law. Compared to popular frameworks like PyTorch and Tensor, this project's implementation was substantially slower. However, this project was an excellent learning process, and while the results might not be competitive, it was deeply satisfying to see parallelism concepts in real life code.

Table of Contents

List of Tables

List of Figures

Introduction

Convolutional neural networks (CNNs) have been widely used for different tasks across a variety of fields. Some of these tasks include image classification (Sharma et al., 2018), object detection (Wang et al., 2026), game playing (Clark & Storkey, 2015), and even medical computer vision tasks (Yamashita et al., 2018). However, while CNNs have been proven to be effective in multiple domains, increased computational demands and intensive training requirements are limitations of these systems that persist today. Parallelization techniques have been explored to accelerate training and optimize runtime performance (Jia et al., 2018). This project involves parallelizing a CNN to evaluate these issues in the modern day.

What is a CNN?

The defining principle of convolutional neural networks (CNNs) is applying a filter across a matrix. The three defining layers of a CNN are the convolutional layer, the pooling layer, and the fully-connected layer. The convolutional layer performs the most intensive computations and contains input data, a filter, and a feature map (Kaur, 2025). The filter is applied to an area of the input data and a dot product is calculated between the overlapping input elements and the filter. The filter then shifts by a stride, and the operation is repeated until every element of the input matrix is covered. The resulting output data computed from the dot products from the input data and filter is called a feature map (IBM, 2021).

The pooling layer is a downsampling operation that reduces the size of the feature map. The two main types of pooling are average pooling and max pooling. Average pooling is when each pooling operation averages the values within the

receptive field. Max pooling, which was utilized in this project, is when each pooling operation selects the pixel with the maximum value within the receptive field (Gholamalinezhad & Khosravi, 2020).

The final layer is the fully-connected layer, in which every input node is connected to every output node. Essentially, a direct connection is formed between nodes in the previous layer and nodes in the output layer. This distribution of information produces the final prediction (Aramendia, 2024).

The Evolution of CNNs

Before the invention of CNNs, neural networks were trained for image classification by using feedforward neural networks. Images would be flattened into a list of pixels and the data would flow in one direction from the input layer to the output layer. The issue with this methodology was that spatial information was discarded by flattening the image (Biswas, 2024).

The earliest precursor to CNNs was the Neocognitron, developed by Kunihiko Fukushima in 1979. This was a multi-layered, self-organized neural network composed of alternating layers that was trained using a form of unsupervised learning. The cascading hierarchy of the alternating layers allowed for pattern recognition that was unaffected by shift in position or shape distortion of input patterns. Furthermore, Fukushima described the Neocognitron as “learning without a teacher,” meaning that it would rely on a set of repeatedly presented stimulus patterns to progress (Fukushima, 1980). However, this reliance on unsupervised learning was a critical limitation, as there was no backpropagation mechanism to send an error signal from the output layer back through the network to optimize all filters. Still, the Neocognitron presented the

architectural idea of a modern CNN, while raising the question of how such a model could be trained more effectively for a given task (Kitishian, 2025).

The term “convolutional neural network” was first coined by Yann LeCun and his team at AT&T Bell Laboratories in 1989. LeCun had worked on and published an early form of his backpropagation algorithm in 1985 while pursuing his PhD at Université Pierre et Marie Curie. He carried forward this work at the Adaptive Systems Research Department at AT&T Bell Laboratories, where he coupled the architectural framework of the Neocognitron with a more robust version of his backpropagation algorithm. This allowed for optimization throughout the network and minimized error upon final classification (LeCun et al., 1989). The culmination of this research was LeNet-5, a multi-layer CNN which utilized max pooling. This groundbreaking 1998 publication detailed the first high-accuracy, end-to-end trainable system for handwriting recognition (LeCun et al., 1998).

Although LeNet was a commercial success in the late 1990s and early 2000s, the research community still deemed neural networks as very difficult to train and too computationally expensive. Traditional machine learning algorithms were heavily favored as they often performed better on smaller datasets that were used at the time. This period was known as a “winter break” in the evolution of neural networks as they were not yet prevalent on a grand scale (Ekundayo & Ezugwu, 2025).

In 2009, the ImageNet dataset was released. This was an open-source dataset that contained 3.2 million annotated images in a densely populated hierarchical structure (Deng et al., 2009). Starting in 2010, a competition called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) took place annually which encouraged

researchers to make significant progress in image classification. The 2012 competition was a critical turning point that broke through the “winter break” of neural networks and called to attention the value of CNNs as a whole. The goal of the 2012 ILSVRC, as written on the challenge website, was to “estimate the content of photographs for the purpose of retrieval and automatic annotation using a subset of the large hand-labeled ImageNet dataset (10,000,000 labeled images depicting 10,000+ object categories) as training” (Stanford Vision Lab, 2012).

Amidst the many submissions, one stood out as a seminal achievement. Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton not only won the challenge, but dominated it. While the next best entry managed a top-5 error rate of 26.2% with more traditional techniques, AlexNet achieved 15.3% with their eight-layer CNN now known as “AlexNet.” This success is largely attributed to three main factors: rectified linear unit (ReLUs) which enabled faster training times without sacrificing accuracy, dropout regularization which combatted overfitting, and, most crucially, the use of multiple Graphics Processing Units (GPUs) for training. AlexNet was parallelized across two NVIDIA GTX 580 3GB GPUs. The training period for the network was only five to six days using this system, a feat which would have been incredibly time consuming using traditional methods (Krizhevsky et al., 2012).

Following AlexNet’s success, CNN architecture rapidly evolved and progressed. In 2014, VGGNet increased depth to improve performance (Simonyan & Zisserman, 2014). Also in 2014, GooLeNet introduced the “inception module” and achieved a top-5 error rate of 6.7% at the ILSVRC that year (Szegedy et al., 2014). ResNet in 2015 introduced “residual connections” which allowed for training of ultra-deep networks (He

et al., 2015). More recently, Vision Transformer in 2020 applied Transformer architecture designed for natural language processing to image classification, beating even advanced CNNs in this task (Dosovitskiy et al, 2020).

As the future of CNNs continues to move towards new architectural paradigms, the contributions of the past may seem less relevant to modern issues. However, AlexNet's concept of a parallelized CNN is still used today. This project takes inspiration from AlexNet's 2012 innovation to learn parallelization concepts and CNN architecture in depth.

Amdahl's Law and the Limits of Parallelization

Although parallelization was a key factor that allowed for significantly faster training times for AlexNet, it is not the ultimate solution to CNN architecture. In fact, this limit was known well before the rapid evolution of CNN architecture as a concept known as Amdahl's Law. Proposed by Gene Amdahl in 1967, Amdahl's Law is the concept that there is a limit to how much speedup can be gained from parallelizing a program. Speedup is defined as how much faster a parallel solution runs in comparison to the sequential solution (Bittla, 2025).

$$Speedup = \frac{Sequential\ Runtime}{Parallel\ Runtime}$$

Figure 1. Speedup equation

Essentially, there are portions of a program's work that cannot be parallelized whatsoever. These portions make up the sequential runtime. The parallel runtime includes portions that can be executed simultaneously among multiple processors. Regardless of how fast the parallel runtime is–with an infinite number of processors, for

instance, it can be considered infinitely fast–the time required to complete the sequential portions acts as the bottleneck of the system. Thus, every parallelized program reaches a certain point where more hardware simply will not increase the speedup. The results of this project will showcase this limitation and its impact.

CUDA and its Utility

Compute Unified Device Architecture (CUDA) is a parallel architecture developed by NVIDIA in 2006 that commercialized general purpose computing on GPUs. CUDA allows programs to effectively use NVIDIA GPUs, allowing for drastically increased performance over CPUs (Ghorpade, 2012). Due to its long history and high adoption in the industry, CUDA and NVIDIA hardware has been a popular choice for deep learning. Thus, it was selected for this project as well.

Implementation

Detailed below is the architecture for this project's parallelized CNN.

CNN Forward Pass

Forward pass is the process of the CNN taking the input image and turning it into a prediction. Below are the steps of this process.

1) Convolution: the most computationally expensive, yet significant step. This is where the filter is applied across the input matrix. Reducing the number of weights needing to be learned helps to make the process more efficient. In this project, instead of learning potentially hundreds of thousand weights, only nine were needed to be learned.

2) ReLU: the activation function used after convolution. Activation functions enable the network to learn complex patterns by introducing non-linearity. ReLU is a simple, but cheap activation function. ReLU = max(0,x)
3) Max Pooling: pooling downsamples, which reduces the amount of computation needed in the following layers. This project uses Max Pooling to preserve edges. Average pooling can blur edges or even eliminate them, which is suboptimal for image classification.
4) Fully Connected: this layer takes the produced feature maps and turns them into decisions that affect classification. It produces 10 raw scores (*10 for each possible class*).
5) Softmax: this process takes the raw scores and normalizes it to a probability without negatives and sums up to 1. The highest value is the selected output of the model.

CNN Training

CNN Training is the process of learning a set of weights that performs well with unseen data. Below are the steps of this process.

1) Forward Pass: the model takes the input and makes a prediction
2) Loss Computation: calculate how wrong the prediction was compared to the true value. Cross-entropy is used to calculate the loss. This loss function is common for classification.
3) Backward Pass: calculate how much each weight contributed to the loss. This is done by applying the chain rule in reverse. In other words, using the calculated

loss and going back through the network to determine what weights contributed to the error. This will produce a gradient for each weight.

4) Optimizer: using Stochastic Gradient Descent(SGD), each weight is updated by subtracting the learning rate times its gradient that was obtained from the backward pass.
5) Repeat: this process is repeated several times over to try to find the optimal weights.

CNN Sequential - C++

This is a simple sequential solution that is intentionally unoptimized, with no multi-threading or libraries called. Nothing is parallelized in this solution. Used nested for loops to iterate through each computation. This project uses this solution as a sequential baseline to compare with the other solutions.

CNN Parallel - CUDA

One kernel per layer is used. This adds some launch overhead, but keeps each kernel small enough to develop and debug in isolation. Most kernels use grid-stride loops, a common pattern that decouples the launch configuration from the problem size. The same kernel runs correctly whether one block or thousands of blocks is launched. Every kernel uses output-stationary parallelization. This means that one thread is responsible for computing one output element. This removes any race conditions in the computation of each kernel. The only exception is a couple of the backward pass kernels, where we need to use atomicAdd for overlapping writes. The convolution kernel stages its filter in shared memory before the main computation. This is because every thread reads all nine weights, so loading them straight from global memory would be

costly. ReLU is fused into the convolution kernel due to it being a simple/cheap operation. For softmax, two parallel reductions are conducted in shared memory. The first finds the maximum logit. The second sums the exponentials to allow normalization into probabilities.

CNN Parallel over multiple GPUs - NCCL

This project extends the CUDA solution to multiple GPUs using synchronous data parallelism. Every GPU holds a copy of the model, and each GPU processes a shard of data and synchronizes gradients after each backward pass. MPI is used for the bootstrap. MPI assigns ranks to each GPU, and rank 0 creates an NCCL unique id and broadcasts it. Every rank uses that unique id as the location of the NCCL communication channel. The calculated gradients will be communicated through this channel. Data is distributed using stride-sharding instead of contiguous chucks. After each GPU completes their backward pass, ncclALLReduce is used to sum their gradients. The optimizer then scales by the global batch size.

Results

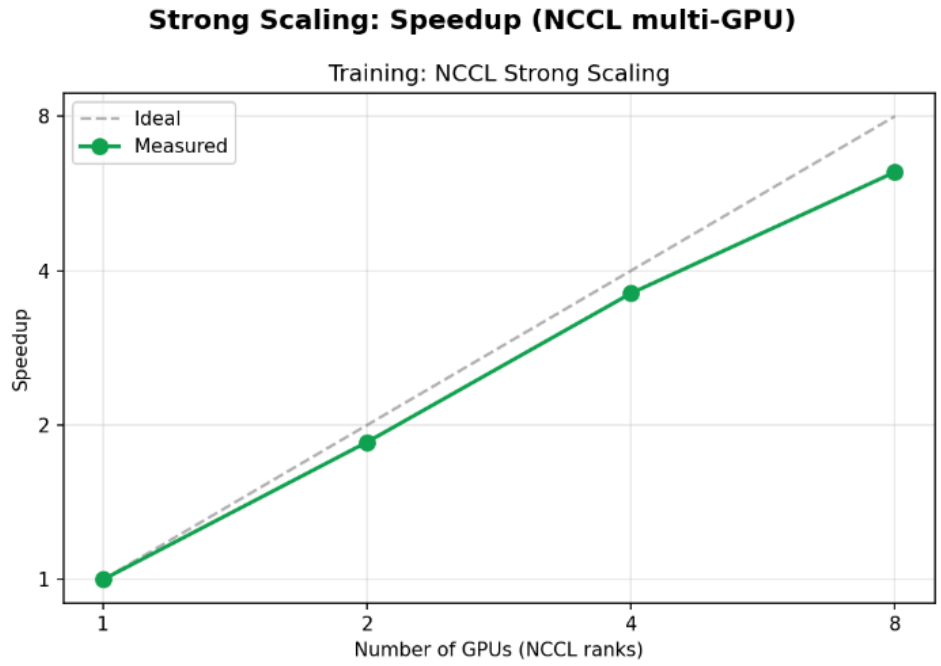

Figure 2. Strong Scaling: Speedup | NCCL multi-GPU

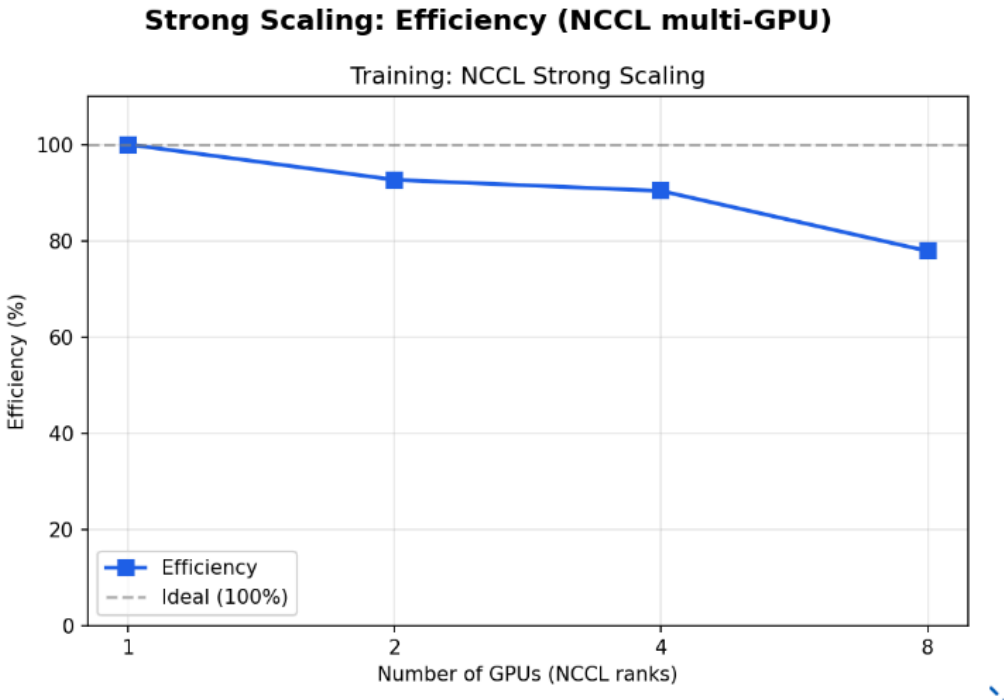

Figure 3. Strong Scaling: Efficiency | NCCL multi-GPU

GPUs	Time (s)	Speedup	Efficiency (%)
1	698.55	1.00	100.00
2	377.04	1.85	92.64
4	193.34	3.61	90.33
8	112.19	6.23	77.83

Table 1. Strong Scaling: Runtimes | NCCL multi-GPU

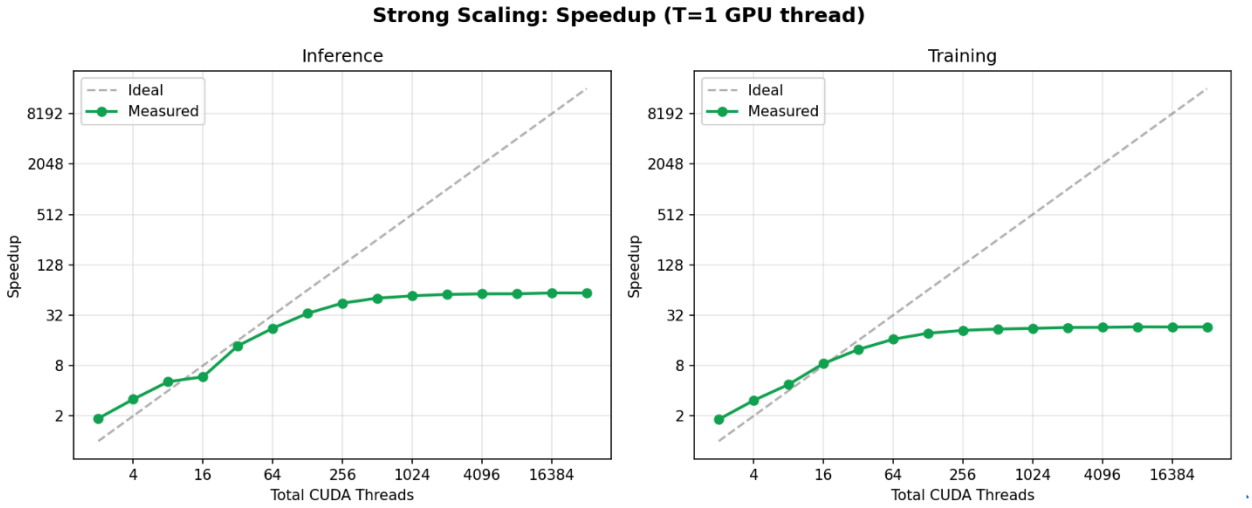

Figure 4. Strong Scaling: Speedup | CUDA single GPU

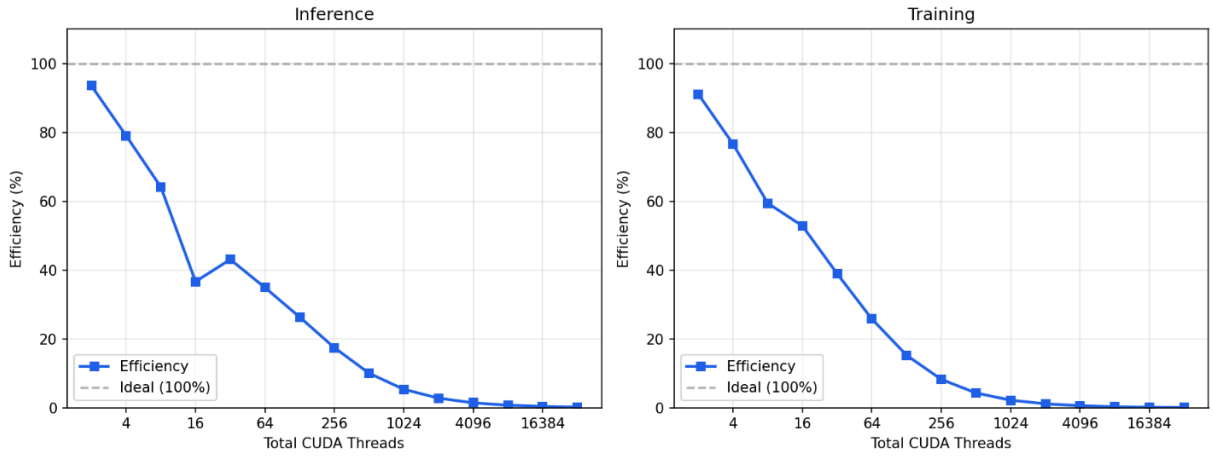

Figure 5. Strong Scaling: Efficiency | CUDA single GPU

Threads	Time (ms)	Speedup	Efficiency (%)
1	1893.11	1.00	100.00
2	1012.52	1.87	93.48
4	599.13	3.16	78.99
8	369.20	5.13	64.10
16	322.98	5.86	36.63
32	137.44	13.77	43.04
64	84.82	22.32	34.87
128	56.22	33.67	26.31
256	42.47	44.58	17.41
512	36.99	51.18	10.00
1024	34.72	54.52	5.32
2048	33.43	56.63	2.76
4096	32.82	57.68	1.41
8192	32.78	57.75	0.71
16384	32.03	59.11	0.36
32768	32.07	59.03	0.18

Table 2. CNN Forward Pass: Runtimes

Threads	Time (s)	Val Acc	Loss	Speedup	Efficiency (%)
1	4079.19	65.33	1.3129	1.00	100.00
2	2239.79	65.33	1.3129	1.82	91.06
4	1331.66	65.33	1.3129	3.06	76.58
8	858.53	65.33	1.3129	4.75	59.39
16	483.03	65.33	1.3129	8.45	52.78
32	327.63	65.33	1.3129	12.45	38.91
64	247.26	65.33	1.3129	16.50	25.78
128	209.27	65.33	1.3129	19.49	15.23
256	193.77	65.33	1.3129	21.05	8.22
512	186.81	65.33	1.3129	21.84	4.26
1024	182.92	65.33	1.3129	22.30	2.18
2048	178.64	65.33	1.3129	22.83	1.11
4096	178.24	65.33	1.3129	22.89	0.56
8192	176.02	65.33	1.3129	23.17	0.28
16384	176.49	65.33	1.3129	23.11	0.14
32768	176.09	65.33	1.3129	23.17	0.07

Table 3. CNN Training: Runtimes

Conclusion

This project aimed to build a CNN from scratch using CUDA. We succeeded in that, creating a five-layer CNN in CUDA. The performance results tell a clear story. Forward Pass scaled well on a single GPU, reaching about 59x over the single-threaded baseline. Training was harder. The single GPU speedup capped at 4.7x and the multi-GPU run topped out at 6.23x. While the difference in these results is large, they still both show a clear demonstration of Amdahl's Law. Compared to PyTorch or TensorFlow, this implementation is much slower, but unsurprising. Both those frameworks are built on years of iterations and optimizations. The main takeaway is that writing parallel code from the ground up brings to life abstract concepts. While the results are not competitive with production frameworks, this project achieved its goal of a hands-on understanding of GPU parallelism and neural networks.

References

Aramendia, A. I. (2024). *Convolutional Neural Networks: A Complete Guide*. Medium. https://medium.com/@alejandro.itoaramendia/convolutional-neural-networks-cnns-a-complete-guide-a803534a1930

Biswas, A. (2024). *The History of Convolutional Neural Networks for Image Classification (1989- Today).* Towards Data Science. https://towardsdatascience.com/the-history-of-convolutional-neural-networks-for-image-classification-1989-today-5ea8a5c5fe20/

Bittla, S. R. (2025). *Mastering Amdahl's Law: Beginner to Advanced with Real Code Examples.* Medium. https://bittla.medium.com/mastering-amdahls-law-beginner-to-advanced-with-real-code-examples-e0037dc35d92

Clark, C. & Storkey, A. (2015). Training Deep Convolutional Neural Networks to Play Go. *Proceedings of the 32nd International Conference on Machine Learning,* in *Proceedings of Machine Learning Research, 37*, 1766-1774. Available from https://proceedings.mlr.press/v37/clark15.html

Deng, J., Socher, R., Fei-Fei, L., Dong, W., Li, K. & Li, L.-J. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 00,* 248-255. doi: 10.1109/CVPR.2009.5206848.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S. and Uszkoreit, J. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929.*

Ekundayo O. S. & Ezugwu A. E. (2025). Deep learning: Historical overview from inception to actualization, models, applications and future trends. *Applied Soft Computing, 181*, 113378. https://doi.org/10.1016/j.asoc.2025.113378.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics, 36*(4), 193–202. https://doi.org/10.1007/bf00344251

Gholamalinezhad, H., & Khosravi, H. (2020). Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485.*

Ghorpade, J. (2012). GPGPU Processing in CUDA Architecture. *Advanced Computing: An International Journal, 3*(1), 105–120. https://doi.org/10.5121/acij.2012.3109

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

IBM. (2021). *Convolutional Neural Networks*. Ibm.com. https://www.ibm.com/think/topics/convolutional-neural-networks

Jia, Z., Lin, S., Qi, C. R., & Aiken, A. (2018). Exploring hidden dimensions in parallelizing convolutional neural networks. *ICML, 2279*, 2288. https://doi.org/10.48550/arXiv.1802.04924

Kaur, J. (2025). *Convolutional Neural Network and its Latest Use Cases.* www.xenonstack.com. Retrieved May 25, 2026, from https://www.xenonstack.com/blog/convolutional-neural-network

Kitishian, D. (2025). *Convolutional Neural Networks (CNN's): History, Impact, Future.*

Klover.ai. https://www.klover.ai/convolutional-neural-networks-cnns-history-impact-future/

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM, 60*(6), 84–90.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation, 1*(4), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Sharma, N., Jain, V., & Mishra, A. (2018). An Analysis Of Convolutional Neural Networks For Image Classification. *Procedia Computer Science, 132*, 377–384. https://doi.org/10.1016/j.procs.2018.05.198

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

Stanford Vision Lab. (2012). *ImageNet*. www.image-Net.org. https://www.image-net.org/challenges/LSVRC/2012/

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2014). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

Wang, Z., Chen, Y., Gu, Y. et al. (2026). The evolution of object detection from CNNs to

transformers and multi-modal fusion. *Sci Rep, 16*, 7517 https://doi.org/10.1038/s41598-026-37052-6

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, *9*(4), 611–629. https://doi.org/10.1007/s13244-018-0639-9