# CNN Architectures and Evolution

Mohammed Luqmaan Akhtar

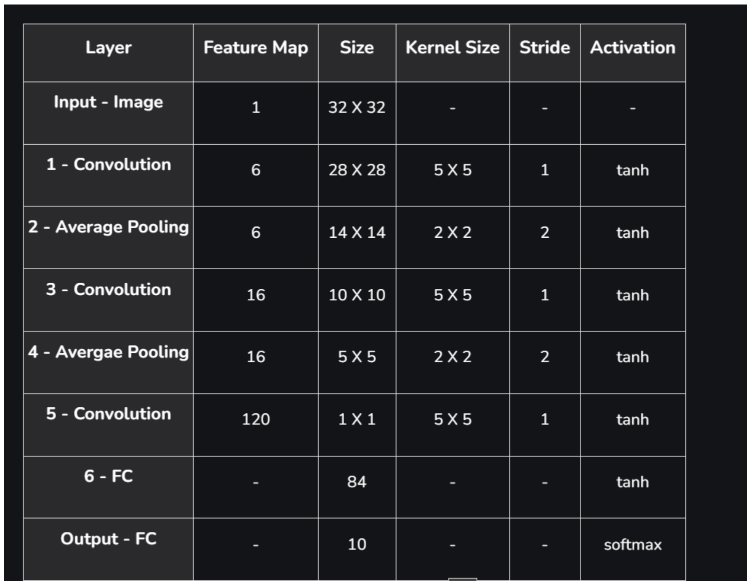
January 7, 2026

## Introduction

Over the years CNN architectures have evolved greatly to overcome challenges such as overfitting, vanishing/exploding gradients and computational costs.

## LeNet-5

This was the first CNN developed and introduced very important concept such as convolution, pooling, and hierarchical feature extraction that underpin modern deep learning models. The LeNet-5 architecture was developed to recognize handwritten and machine-printed characters, a function that showcased the potential of deep learning in practical applications. The activation function was *tanh* and the classification function was softmax.



| Layer | Feature Map | Size | Kernel Size | Stride | Activation |
|---|---|---|---|---|---|
| Input - Image | 1 | 32 X 32 | - | - | - |
| 1 - Convolution | 6 | 28 X 28 | 5 X 5 | 1 | tanh |
| 2 - Average Pooling | 6 | 14 X 14 | 2 X 2 | 2 | tanh |
| 3 - Convolution | 16 | 10 X 10 | 5 X 5 | 1 | tanh |
| 4 - Avergae Pooling | 16 | 5 X 5 | 2 X 2 | 2 | tanh |
| 5 - Convolution | 120 | 1 X 1 | 5 X 5 | 1 | tanh |
| 6 - FC | - | 84 | - | - | tanh |
| Output - FC | - | 10 | - | - | softmax |

Figure 1: LeNet-5 architecture

## AlexNet

AlexNet is a deep learning model that made a big impact in image recognition. It became famous for its ability to classify images accurately. It won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 with a top-5 error rate of 15.3% (beating the runner up which had a top-5 error rate of 26.2%).
AlexNet was revolutionary because it applied $Dropout(0.5)$ to the first two fully connected layers. It also applied data augmentation for training. These two methods reduced over-fitting drastically.
The activation function used was $ReLU$ which led to a $6x$ speedup in training because it prevented activation saturation.
It also split the network across two GPUs, showing how deep learning can benefit from parallel computing for faster training.

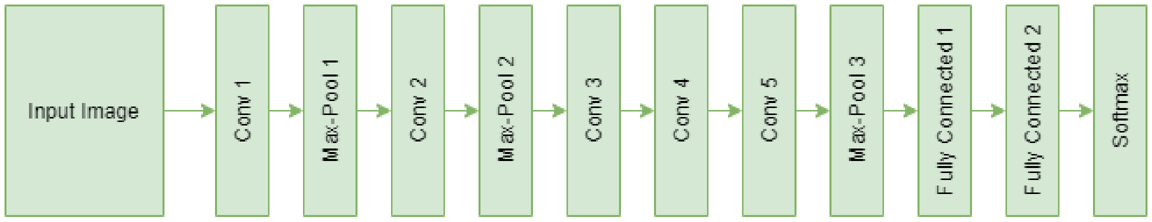
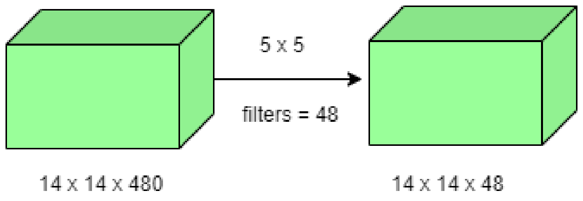


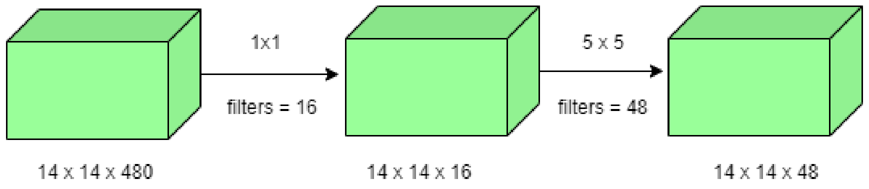
Figure 2: AlexNet architecture

## InceptionNet (GoogLeNet)

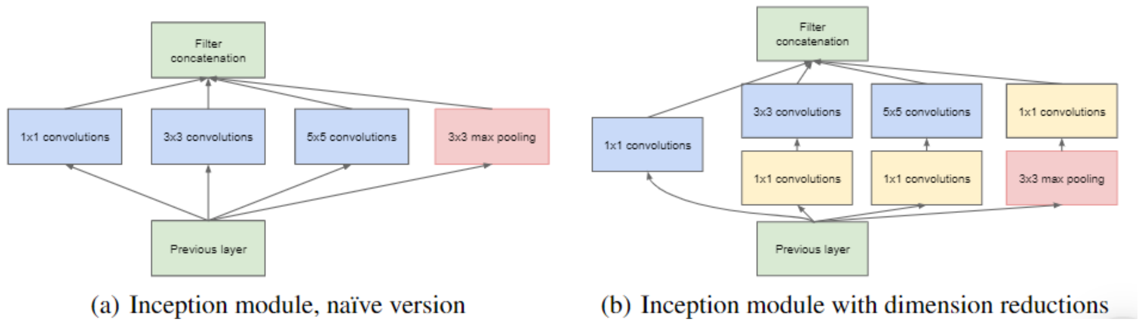
The GoogLeNet architecture is very different from previous architectures such as AlexNet etc. It uses many different kinds of methods such as 1x1 Convolutions, Inception Modules and Auxillary Classifiers.
1×1 convolutions are used for dimensionality reduction. These layers help decrease the number of trainable parameters while enabling deeper and more efficient architectures. For example, in a certain case without 1×1 Convolution there would be (14×14×48)×(5×5×480)=112.9M operations



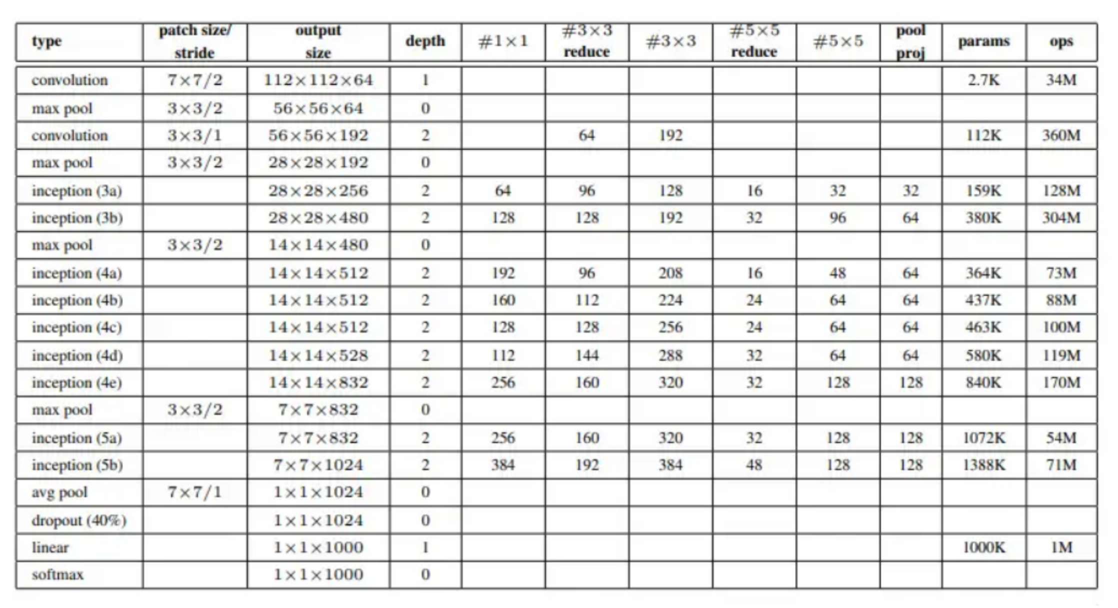
and with 1×1 Convolution there would be (14×14×16)×(1×1×480)+(14×14×48)×(5×5×16)=5.3M operations.



GoogleNet uses Global Average Pooling which significantly reduces the number of parameters and solves overfitting.
The Inception module is the architectural core of GoogLeNet. It processes the input using multiple types of operations in parallel, including 1×1, 3×3, 5×5 convolutions and 3×3 max pooling. The outputs from all paths are concatenated depth-wise. This enables the network to capture features at multiple scales effectively and improves representational power without significantly increasing computation.



GoogLeNet introduces auxiliary classifiers(intermediate branches that act as smaller classifiers) to address the vanishing gradient problem during training. These are active only during training and help regularize the network. Structure of Each Auxiliary Classifier: average pooling layer (5×5, stride 3), 1×1 convolution (128 filters, ReLU), fully connected layer (1024 units, ReLU), dropout layer (dropout rate = 0.7) and a fully connected softmax layer (1000 classes). The auxiliary losses are added to the main loss with a weight of 0.3 to stabilize training.

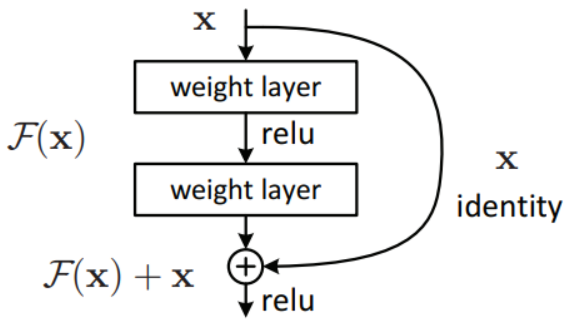


| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

Figure 3: GoogleNet architecture

## ResNet

As we train deeper layers the problem of vanishing/exploding gradient increases and eventually we reach a point where increasing the number of layers actually increases the error instead of reducing it.
In order to solve this problem, a new architecture called Residual Network was introduced by Microsoft researchers in 2015.
In this network, we use a technique called skip connections. The skip connection connects activations of a layer to further layers by skipping some layers in between. This forms a residual block. ResNets are made by stacking these residual blocks together.
The approach behind this network is instead of layers learning the underlying mapping, we allow the network to fit the residual mapping. So, instead of say H(x), initial mapping, let the network fit, *F(x) = H(x) - x is the "residual", which is usually smaller and easier to learn The network combines this residual F(x) with the input x using a shortcut or skip connection: which gives H(x) = F(x) + x.*


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

The advantage of adding this type of skip connection is that if any layer hurt the performance of architecture then it will be skipped by regularization. So, this results in training a very deep neural network without the problems caused by vanishing/exploding gradient.

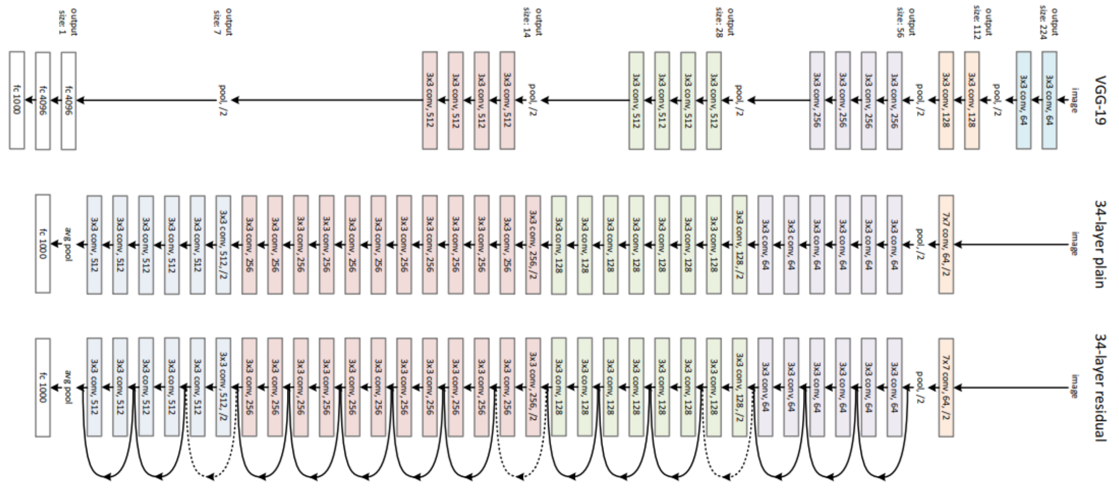


Figure 4: ResNet architecture

## MobileNet

Mobilenet is a model which does the same convolution as done by CNN to filter images but in a different way than those done by the previous CNN. It uses the idea of Depth convolution and point convolution which is different from the normal convolution as done by normal CNNs. Normal convolution applies a single filter across all input channels simultaneously, combining spatial and channel information, while depthwise separable convolution splits this into two steps: Depthwise Convolution (spatial filtering on each channel individually) and Pointwise Convolution (1x1 convolution to mix channel outputs), drastically reducing parameters and computation for efficiency, making it ideal for mobile/edge devices. Since these ways of convolution reduce the comparison and recognition time a lot, so it provides a better response in a very short time and hence they are widely used as image recognition model.

## EfficientNet

EfficientNet is a family of convolutional neural networks designed to achieve high accuracy with fewer parameters and lower computational cost compared to traditional CNN architectures.

Instead of scaling network depth, width, or input resolution independently, EfficientNet scales all three dimensions together in a balanced manner using a single scaling coefficient. The scaling is defined as:

$$\text{Depth} = \alpha^{\phi}, \quad \text{Width} = \beta^{\phi}, \quad \text{Resolution} = \gamma^{\phi}$$

where $\phi$ controls the model size and $\alpha, \beta, \gamma$ are constants chosen to maintain computational efficiency.

EfficientNet consists of models EfficientNet-B0 to EfficientNet-B7, where higher versions scale the network systematically to improve accuracy.