Bert Moons · Daniel Bankman
Marian Verhelst

Embedded Deep Learning

Algorithms, Architectures and Circuits for Always-on Neural Network Processing

Springer

Embedded Deep Learning

Bert Moons • Daniel Bankman • Marian Verhelst

Embedded Deep Learning

Algorithms, Architectures and Circuits for Always-on Neural Network Processing

Springer

Bert Moons
ESAT-MICAS
KU Leuven
Leuven, Belgium

Daniel Bankman
Department of Electrical Engineering
Stanford University
Stanford, CA, USA

Marian Verhelst
ESAT-MICAS
KU Leuven
Leuven, Belgium

ISBN 978-3-319-99222-8 ISBN 978-3-319-99223-5 (eBook)
https://doi.org/10.1007/978-3-319-99223-5

Library of Congress Control Number: 2018956280

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To Lise

Preface

Although state of the art in many typical machine learning tasks, deep learning algorithms are very costly in terms of energy consumption, due to their large amount of required computations and huge model sizes. Because of this, deep learning applications on battery-constrained wearables have only been possible through wireless connections with a resourceful *cloud*. This setup has several drawbacks. First, there are privacy concerns. Cloud computing requires users to share their raw data—images, video, locations, speech—with a remote system. Most users are not willing to do this. Second, the cloud-setup requires users to be connected all the time, which is unfeasible given current cellular coverage. Furthermore, real-time applications require low latency connections, which cannot be guaranteed using the current communication infrastructure. Finally, wireless connections are very inefficient—requiring too much energy per transferred bit for real-time data transfer on energy-constrained platforms. All these issues—privacy, latency/connectivity, and costly wireless connections—can be resolved by moving towards computing at the *edge*. Finding ways to do this is the main topic of this dissertation. It focuses on techniques to minimize the energy consumption of deep learning algorithms for embedded applications on battery-constrained wearable edge devices.

Computing in the edge is only possible if these deep learning algorithms can be run in a more energy-efficient way, within the energy and power budget of the computing platform available on a wearable device. In order to achieve this, several innovations are necessary on all levels of an application's design hierarchy. Smarter **applications** can be developed for more statistically efficient deep learning **algorithms**, which in turn should run on optimized **hardware** platforms built on specifically tailored **circuits**. Finally, designers should not focus on any of these fields separately but should **co-optimize** hardware and software to create minimum energy deep learning platforms. This book is an overview of possible solutions towards designing such systems.

Leuven, Belgium — Bert Moons
Stanford, CA, USA — Daniel Bankman
Leuven, Belgium — Marian Verhelst
May 2018

Acknowledgments

The authors would like to thank several people and institutions for their valuable contributions to this work.

We thank Tom Michiels, Edith Beigne, Boris Murmann, Wim Dehaene, Tinne Tuytelaars, Ludo Froyen, and Hugo Hens for their comments and discussion. IWT, intel, Qualcomm, and Synopsys for their funding, software, and support. Thanks to Florian Darve and Marie-Sophie Redon at CEA-LETI and to Etienne Wouters and Luc Folens at imec/IC-Link for their back-end support in ASIC design.

We would also like to acknowledge all coauthors for their contributions to this manuscript. Thanks to Koen Goetschalckx, Nick Van Berckelaer, Bert De Brabandere, Lita Yang, Roel Uytterhoeven, Martin Andraud, and Steven Lauwereins.

Contents

Acronyms

as	Accuracy-scalable
AC	Approximate computing
ANN	Artificial neural network
ANT	Algorithmic noise tolerance
ASIC	Application-specific integrated circuit
ASIP	Application-specific instruction-set processor
BW	Bandwidth
CDAC	Capacitive digital to analog converter
CM	Common-mode
CNN	Convolutional neural network
CONVL	Convolutional layer
CPU	Central processing unit
CSA	Carry save adder
DAS	Dynamic-accuracy-scaling
DVAS	Dynamic-voltage-accuracy-scaling
DVAFS	Dynamic-voltage-accuracy-frequency-scaling
EDA	Electronic design automation
EDP	Energy delay product
FCL	Fully connected layer
FCN	Fully connected network
FPNN	Fixed-point neural networks
GB	Gigabyte
GOPS	Giga-operations per second
GOPS/W	Giga-operations per second per Watt
GPU	Graphical processing unit
HPC	High-performance computing
I2l	Input-to-label
IC	Integrated circuit
IoT	Internet of Things
ISA	Instruction set architecture
LSB	Least significant bit

LSTM	Long short-term memory
MAC	Multiply-accumulate
MSB	Most significant bit
MSBNN	Mixed-signal binary neural network ASIC
NLP	Natural language processing
NPU	Neural processing unit
nas	Non-accuracy-scalable
nvas	Non-voltage-accuracy-scalable
QNN	Quantized neural networks
RMS	Recognition, mining, and synthesis
RNN	Recurrent neural networks
SC	Switched-capacitor or switch-cap
SGD	Stochastic gradient descent
SotA	State of the art
STE	Straight-through estimator
TOPS	Tera-operations per second
TOPS/W	Tera-operations per second per Watt
V_t	Threshold voltage of CMOS transistors
VDD	Typical name for the supply voltage
VLIW	Variable length instruction word
VOS	Voltage over-scaling
vas	Voltage-accuracy-scalable

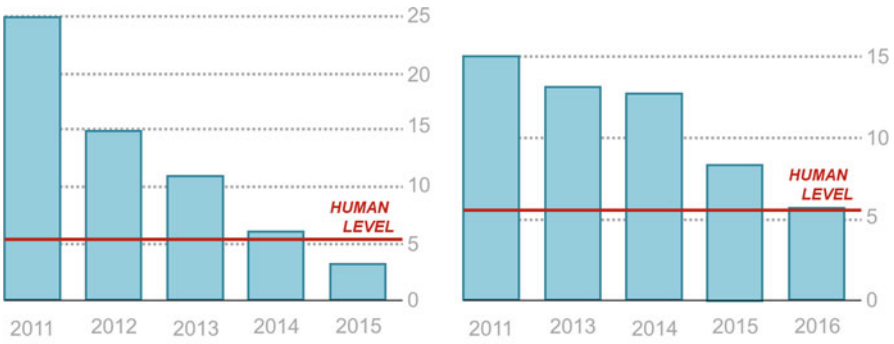
Chapter 1
Embedded Deep Neural Networks

1.1 Introduction

Humankind has long dreamed of creating machines that think. Ever since the conception of programmable computers, people have wondered whether these machines could become intelligent. Today, some of these goals in creating **artificial intelligence** have been achieved. Intelligent software can automate intuitive tasks such as understanding and translating speech (Chiu et al. 2017) and interpreting images (Krizhevsky et al. 2012a), even in order to make reliable image-based diagnoses in medicine (Esteva et al. 2017). Solutions to less intuitive problems in AI, such as human-level machine intelligence and consciousness, are still very far off.

The class of more intuitive problems, such as speech and image recognition, is currently solved through letting computers learn from experience and by modeling reality through a hierarchy of concepts. The mathematical representation of these models typically is a deep graph, with many layers. Hence, for this reason, this set of state-of-the-art (SotA) techniques is generally referred to as deep learning. More specifically, deep learning uses neural networks with many layers to represent these abstract models. Neural networks have been a source of study since 1940s, but have only recently broken through and advanced the SotA in a number of recognition tasks. Since 2012, deep learning algorithms have shown unprecedented performance in a number of tasks and have broken record after record in AI challenges and competitions. As of 2014, they have surpassed the human performance in visual recognition and since 2016 in speech recognition, as shown in Fig. 1.1. Now, for the first time in human history, we have developed machines that can reliably replace some of our senses and the cognitive ability necessary to interpret those signals. This is crucial in developing novel and emerging applications in robotics, self-driving cars, and many more.

B. Moons et al., *Embedded Deep Learning*,
https://doi.org/10.1007/978-3-319-99223-5_1



(a) Image classification on IMAGENET (Russakovsky et al, 2015)

(b) Speech recognition on Switchboard (Godfrey et al, 1992)

Fig. 1.1 Deep neural networks outperform trained humans on (**a**) 1000-class image recognition and (**b**) real-time speech captioning

Although these networks are extremely powerful, they are also very expensive from a computational and hardware perspective. Most deep learning algorithms are extremely computationally and memory intensive, requiring tens of megabytes for filter coefficient storage and hundreds of millions of operations per input. This high cost makes them difficult to employ on always-on embedded or battery-constrained systems, such as smartphones, smart-glasses, and even autonomous vehicles such as robots, cars, and drones. This book investigates novel applications, adaptations to neural networks, and hardware architectures and circuits that could make this vision of always-on state-of-the-art wearable perceptive capabilities a reality.

This chapter consists of four main parts. Section 1.2 is a quick introduction to some general concepts in machine learning. Section 1.3 is an introduction to deep learning techniques. It discusses artificial neural networks (ANN), convolutional neural networks (CNN), recurrent neural networks (RNN), and how to efficiently train them. Throughout the rest of this text, knowledge of these algorithms is considered a prerequisite. Section 1.4 is a literature overview of the challenges of embedded deep learning on battery-constrained devices and of the currently existing methods and research directions that could make this a reality. Finally, Sect. 1.5 is an overview of my research contributions towards this end-goal of embedded and always-on neural network-based inference.

1.2 Machine Learning

A machine learning algorithm is an algorithm that is able to learn from data (Goodfellow et al. 2016). Here, learning is understood as in Mitchell (1997): "a computer program is said to learn from experience E with respect to some class of tasks T and

performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." The following overview is based on Goodfellow et al. (2016), which is an excellent introduction to the field.

1.2.1 Tasks, T

Several tasks *T* are targeted in machine learning applications. Machine learning is used for regression analyses (Huang et al. 2012), transcription (Shipp et al. 2002), machine translation (Cho et al. 2014; Bahdanau et al. 2014), anomaly detection (Chandola et al. 2009; Erfani et al. 2016), denoising (Vincent et al. 2010), and synthesis techniques (Ze et al. 2013; Rokach et al. 2012). Most important in the context of this dissertation is the **classification** (Glorot et al. 2011; LeCun et al. 2015) task.

In this task, the machine is asked to specify which of *k* categories some input belongs to. An example of a classification task is object recognition, where the input is an input image and the output is a numeric code identifying the object in the image. Another example is speech recognition, where an input audio waveform is translated into a sequence of classified phonemes or words. Most modern object classification is best accomplished with deep learning algorithms. Deep learning-based classification applications are hence the main scope of this work.

1.2.2 Performance Measures, P

The performance measure *P* is a quantitative measure of a machine learning algorithm's performance.

P differs depending on the task at hand. For detection, a two-class classification problem, measures for *P* are **precision** (true returned positives divided by all returned positives) and **recall** (true returned positives divided by all positives). More generally, in classification and transcription tasks, P is the **accuracy** or error rate. Here, accuracy is the proportion of correctly classified examples.

In order to estimate the performance of the machine learning algorithm on data that it has not seen before, as would happen in the real world, it is crucial to verify *P* on a **test set** of data that is separate from the data used for training the system.

1.2.3 Experience, E

A machine learning algorithm can increase its performance *P* on a task *T* by gaining more experience *E*. Algorithms can be categorized as **supervised** and **unsupervised** depending on the type of *E* they have access to during their learning process.

1.2.3.1 Supervised Learning

Supervised learning algorithms experience a data set, where each example is associated with a label or target. They learn to **predict** a target **y** from an input **x**. This process is analogous to the concept of a teacher telling a student what to do, hence supervising him to predict the correct result.

1.2.3.2 Unsupervised Learning

Supervised learning algorithms experience a data set and learn useful properties of the structure of this data. They basically model the probability distribution of a property of a data set. In unsupervised learning, no labels or targets are provided, making it a significantly harder task than supervised learning. The algorithm must make sense of the data without any supervision. Examples of algorithms in unsupervised learning are principal component analysis and k-means clustering.

The rest of this text will only focus on algorithms and applications that make use of supervised learning.

1.3 Deep Learning

The rapidly developing field of deep learning is introduced in several excellent works (LeCun et al. 2015; Goodfellow et al. 2016; Li et al. 2016). For decades, conventional machine learning techniques were SotA in object recognition, although they were limited in their ability to process natural data in their raw form. Figure 1.2a illustrates how in conventional machine learning, constructing

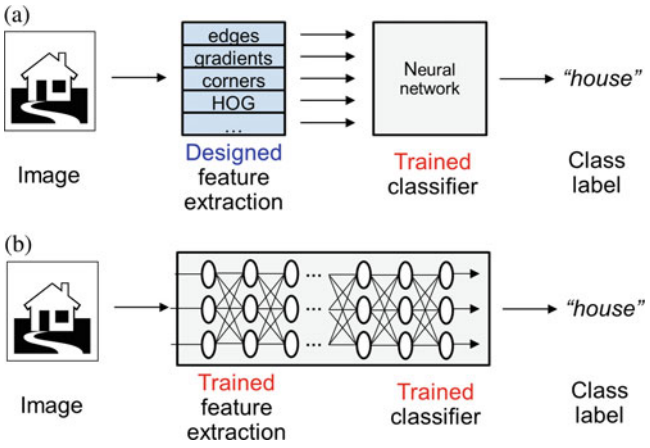


Fig. 1.2 Comparing the (**a**) classical approach to machine learning using hand-designed features to the (**b**) multi-layered representational approach of deep learning operating on raw inputs

a pattern recognition or machine learning system required careful engineering and considerable domain expertise to design a feature extractor that could transform raw data (such as pixel values of an image) into a suitable internal representation or feature vector from which a learning subsystem, typically a classifier, could detect or classify patterns in the input. Examples of such features in the computer vision context can be edges, gradients, specific colors, or more automated forms such as SURF (Bay et al. 2006) or HOG (Dalal and Triggs 2005).

The concept of deep learning breaks with this tradition of using hand-designed features, as it is a type of representation learning (LeCun et al. 2015). Representation learning is a set of methods that allow a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. This is illustrated in Fig. 1.2b, where a raw input image is fed to a deep learning system. In deep learning algorithms, multiple levels of representation are used. They are a multi-layered system composed of nonlinear modules each transforming an input representation (starting with the input images, for example) into an output representation at a higher, more abstract level. If many such layers, or many such transformations, are combined, very complex functions can be learned.

The difference between classical approaches to machine learning and representational deep learning is illustrated in Fig. 1.2. The key aspect of deep learning is that these layers of features are not designed by hand by human engineers: they are learned from data using a general purpose learning procedure, typically a form of stochastic gradient descent (see Sect. 1.3.4). This strategy has been proven to be very powerful and crucial to many of the recent breakthroughs in Machine Learning and pattern recognition, as shown in Fig. 1.1.

Instead of hand-designed by a human expert, the intermediate features generated in representational deep neural network are a mathematical optimum, within the constraints of the optimization method (see Sect. 1.3.4). These features can be visualized through the powerful tools from Olah et al. (2017), which were used to generate Fig. 1.3. Figure 1.3 shows how GoogleNet (Szegedy et al. 2015) builds up its understanding of images over many layers. The first layers extract low-level

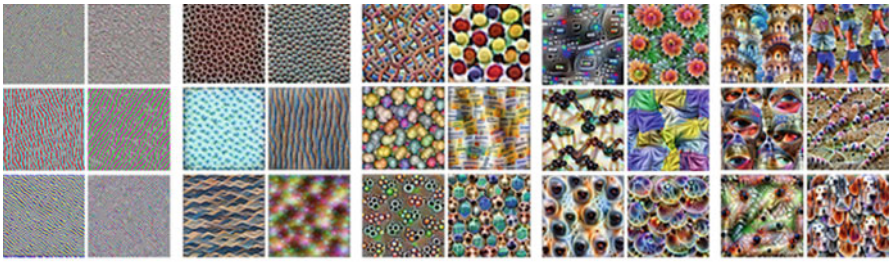


Fig. 1.3 Visualizations of features learned by GoogleNet (Szegedy et al. 2015) on IMAGENET (Russakovsky et al. 2015), taken from Olah et al. (2017). Learned features vary from simple (**a**) in the first layers to more complex and abstract (**e**) in the final layers

features such as edges and contrast changes, as shown in Fig. 1.3a. Deeper layers use that information to extract more complex features, such as textures (Fig. 1.3b), patterns (Fig. 1.3c), parts (Fig. 1.3d), and ultimately objects (Fig. 1.3e).

The rest of this section introduces and discusses three types of typical neural network architectures that are used in deep learning to perform this automated feature extraction. Basic deep feed-forward neural networks are discussed in Sect. 1.3.1. Convolutional neural networks, their optimized form useful for modeling spatial correlations and leveraging sparse interactions, parameter sharing, and equivariant representations, are discussed in Sect. 1.3.2. Recurrent neural networks, used to model sequential correlations, are discussed in Sect. 1.3.3, although they are somewhat out of scope of this text. Finally, Sect. 1.3.4 dives into some details on how to efficiently train and regularize large-scale neural networks.

1.3.1 Deep Feed-Forward Neural Networks

Deep feed-forward networks are also referred to as multi-layer perceptrons, dense or fully connected networks (FCN), or artificial neural networks (ANN). They are the quintessential deep learning models, which have been around since the 1940s and 1960s. A deep feed-forward neural network has no internal feedback connections, hence its name. A network with a depth of n layers would compute the following function: $f(x) = f^{(n)}(\ldots f^{(2)}(f^{(1)}(x)))$, where $f^{(1)}$ is the networks **input** layer, $f^{2\ldots n-1}$ are called **hidden** layers, and $f^{(n)}$ is the networks **output** layer. During training, $f(x)$ is trained to come as close as possible to $f^*(x)$, which is the evaluated ground truth for a given problem. These networks are called *neural*, as they are loosely inspired by neuroscience.

Figure 1.4 is an illustration of a three layer **deep** feed-forward neural network. It has one input and output layer and a single hidden layer. The dimension of each of

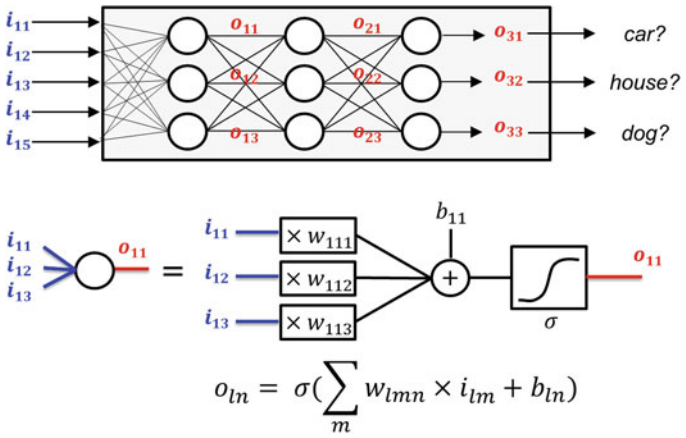


Fig. 1.4 Graphical representation of a simple deep feed-forward neural network

the layers is referred to as the **width** of the layer. Here each layer has a width of 3. Each of the units, or neurons, in such a layer performs a dot product or vector-to-scalar function on an input feature vector, generating a single output feature. A full layer contains a vector of neurons and hence performs a vector-to-vector function on an input feature vector x. In order for such networks to model nonlinear functions f, a nonlinear activation function has to be added to the neuron functionality. Typically used options are the sigmoid function $a(z) = \sigma(z)$, the hyperbolic tangent function $a(z) = tanh(z)$, or various forms of rectified linear units (ReLU) $a(z) = max(0, z)$. The functionality of a single neuron can hence be described as in the following equation:

$$O = a\left(\sum_{m=0}^{M} W[m] \times x[m] + B\right) \tag{1.1}$$

where a can be any of the activation functions listed above. From an algebraic point of view, a single fully connected layer (FCL) can hence be seen as a vector–matrix (input vector x, weight matrix W) product with a per-neuron additive scalar bias B after which an element-wise activation function a is applied. This means FCNs are already supported through heavily optimized BLAS and GEMM libraries for x86 and RISC CPUs.

According to the **universal approximation theorem** (Hornik et al. 1989) the feedforward network above can be proven to be able to approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$. This is the case only if the network contains at least one hidden layer with *enough* hidden units and if it uses a "squashing" activation function, such as the σ and ReLU functions. Hence, any function we are trying to learn can be represented by a deep feed-forward neural network of sufficient size. However, this does not mean we'll also be able to train this network to map this function. The training algorithm might get stuck in a local minimum or it could converge to the wrong function when it is overfitting. On top of that, deeper networks suffer from *vanishing gradients* (Hochreiter et al. 2001) when plain topologies and optimization techniques are used. This means the gradients for the weights in the first layers become too small for them to be significantly updated. In general, the giant networks that might be necessary to learn complex nonlinear functions are infeasible to train with current technologies. Details on regularization to prevent overfitting and on optimization algorithms are given in Sect. 1.3.4.

1.3.2 Convolutional Neural Networks

Albeit a universal function approximator, deep feed-forward neural networks do suffer from several flaws, mainly linked to the difficulty to train them. Convolutional neural networks (CNN) are a specialized form of deep feed-forward neural networks that try solving these flaws. First, on high-dimensional input data such as images,

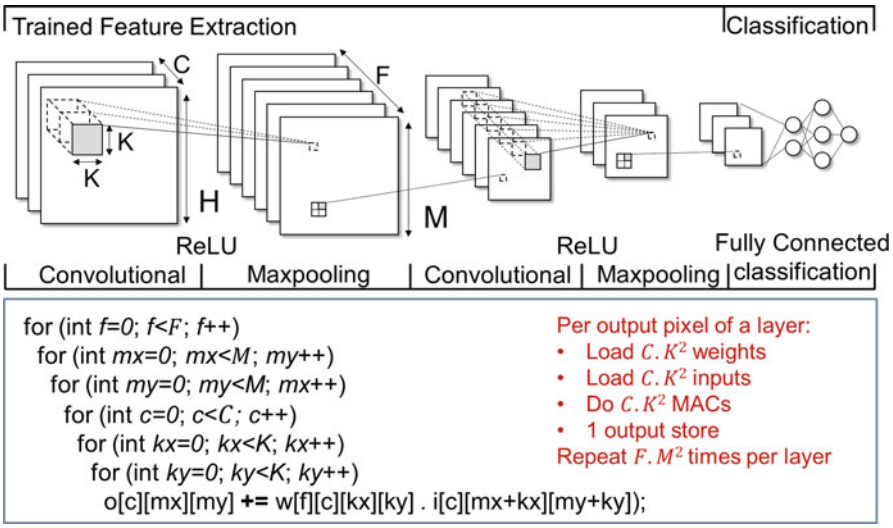


Fig. 1.5 A typical example of a multi-layer convolutional neural network

deep feed-forward input layers will become huge as they are fully connected. For example, a single neuron operating on a VGA image requires almost a million weights. This large amount of parameters leads to overfitting, even when using strong regularizers (Sect. 1.3.4). For many, especially visual applications, this full connectivity is overkill. In visual classification, initially only local connections of 3–10×3–10 modeling relevant edges are important. In such applications, exploiting **sparse connectivity** is a way to reduce the number of weights in a neural model. On top of that, the same pattern can appear anywhere in an image, therefore it makes sense to use the same filter on multiple position of an input image. This **parameter sharing** is another way to reduce the memory requirements for neural models, and to improve their statistical efficiency, which indicates the number of required operations to achieve a given accuracy.

CNNs, visualized in Fig. 1.5, do exploit the previous characteristics—**sparse connectivity** and **parameter sharing**—to improve the statistical efficiency of neural network models and increase their trainability. These networks are a type of artificial neural networks inspired by visual neuroscience. They are a cascade of multiple stacked convolutional, nonlinearity, and pooling-layers used for feature extraction, followed by a smaller number of fully connected neural network layers used for classification. The number of cascaded stages in recent CNN models varies anywhere from 2 (LeCun et al. 1998), typically 10–20 (Simonyan and Zisserman 2014a) to more than one hundred (He et al. 2016a), ending with typically 1–3 fully connected layers (Krizhevsky et al. 2012a) for classification.

A **convolutional layer** (CONVL), with topology parameters listed in Fig. 1.5 and Table 1.1, transforms input feature maps (I) into output feature maps (O), each containing multiple units. Each unit in an output feature map ($M \times M \times F$) is

Table 1.1 Parameters of a CONV layer

Parameter	Description	Range
F	Number of filters per layer	16–512
H	Width and height of input feature map	16–227
C	Number of channels in input feature map	3–512
K	Width and height of filter plane	1–11
M	Width and height of output feature map	16–227

connected to local patches of units ($K \times K \times C$) in the input feature maps through a filter $W[F]$ ($K \times K \times C \times 1$) in a filter bank W ($K \times K \times C \times F$) existing out of a set of machine-learned weights and a bias (B) per output feature map. A formal mathematical description is given in the following equation:

$$O[f][x][y] = a\left(\sum_{c=0}^{C}\sum_{i=0}^{K}\sum_{j=0}^{K} I[c][Sx+i][Sy+j] \times W[f][c][i][j] + B[f]\right) \tag{1.2}$$

where a is a typical activation function such as ReLU, S is a stride, and x, y, f are bounded by: $x, y \in [0, \ldots, M[$ and $f \in [0, \ldots, F[$. Figure 1.5 shows Eq. (1.2) can be naively implemented as a deep nested loop.

The result of the local sum computed in this filter bank is then passed through a **nonlinearity layer**, typically a ReLU, using the nonlinear activation function $a(z) = max(0, z)$, where u is a feature map unit. This activation function reduces the vanishing gradient problem (Hochreiter et al. 2001) in the backpropagation-based training phase of the network and leads to high degrees of sparsity due to non-activated outputs.

Max-pooling layers compute and output only the maximum of a local patch (typically 2×2 or 3×3) of output units in a feature map. They thereby reduce the dimension of the feature representation and create an invariance to small shifts and distortions in the inputs.

Finally, **fully connected layers** (FC) are used as classifiers in the CNN algorithm. An FC layer is also here described as the matrix–vector product $O[z] = \sum_{m=0}^{M} W[z, m] \times I[m] + B[z]$, equivalent to Eq. (1.1), where M is the size of vectorized input feature map and $z \in [0, \ldots, Z[$ is the number of neurons in the fully connected layer. As all the used weights are only used once in a forward pass, there is no weight reuse in these layers. Due to this observation, architectures proposed for FC layers, as Han et al. (2016) and Reagen et al. (2016), are different from architectures for CONV layers. In general, the optimal network architecture, characterized by the number of cascading stages and the values of model parameters F, H, C, K, and M, varies for each specific application.