Universidad del País Vasco
Euskal Herriko Unibertsitatea

Máster Universitario en
Ingeniería Computacional y Sistemas Inteligentes

Konputazio Zientziak eta Adimen Artifiziala Saila –
Departamento de Ciencias de la Computación e Inteligencia Artificial

K I S A
I C S I

Master's Thesis

Deep learning review and its applications

Andoni Azkarate Saiz

Supervisor

Alicia d'Anjou d'Anjou
Department of Computer Science and Artificial Intelligence
Computer Science Faculty

informatika fakultatea
facultad de informática

KZAA /CCIA

September 2015

Deep learning review and its applications

Andoni Azkarate

September 2015

Abstract

Deep neural networks have recently gained popularity for improving state-of-the-art machine learning algorithms in diverse areas such as speech recognition, computer vision and bioinformatics. Convolutional networks especially have shown prowess in visual recognition tasks such as object recognition and detection in which this work is focused on. Modern award-winning architectures have systematically surpassed previous attempts at tackling computer vision problems and keep winning most current competitions. After a brief study of deep learning architectures and readily available frameworks and libraries, the LeNet handwriting digit recognition network study case is developed, and lastly a deep learning network for playing simple videogames is reviewed.

Keywords: deep learning, machine learning, artificial neural network, visual recognition, object recognition, object mining, pattern recognition, computer vision, convolutional neural network, caffe

Contents

1 Introduction

The field of artificial neural networks has progressed much since its creation in the 1940s. Computational models of neural networks have been around for over half a century, beginning with a simple model McCulloch and Pitts developed in 1943[1]. Following Donald Hebb's Hebbian learning, researchers began applying these ideas to computational models in the late 1940s.

1957 saw the creation of the perceptron by Frank Rosenblatt, a type of binary classifier, which was used for pattern recognition based on a two-layer computer learning network. However, following Marvin Minsky and Seymour Papert's work that found issues in state-of-the-art computational neural models (issues regarding to the impossibility of processing the XOR circuit and the expensive processing power requirements) the field of artificial neural networks stagnated until multilayered perceptrons were tried, now able to solve the XOR function.

In the 1970's Paul Werbos developed the backpropagation algorithm, and together with multiple layers of perceptron-like neurons such as the Neocognitron the field of artificial neural networks blossomed again. This algorithm is an abbreviation for "backward propagation of errors", used in conjunction with an optimization method such as gradient descent updates the weights of the neurons in different layers depending on the result of the loss function, the network's measure of successful classification, bringing the network closer to the desired output; it is thus considered a supervised learning method. Nevertheless, the more the layers the network has, the bigger the problem of vanishing gradient gets.

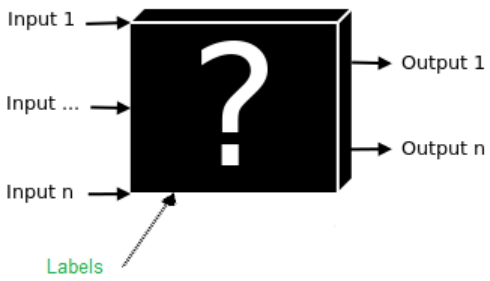

Figure 1: A neural network that may sit inside a neural box performing machine learning

From 1980s and on, different architectures were researched and tried, some of them were few layers deep, but some researches went on creating deep networks; so the field of deep learning was born. The popularization of deep learning however, came in around 2006 with Geoff Hinton, Yann LeCun and other's publications around the topic of deep networks in machine learning[2, 3, 4].

2 Deep Learning overview

Deep learning is a branch of machine learning, an ensemble of learning techniques and algorithms that usually employ non-linear transformation units (such as artificial neurons or Restricted Boltzmann Machines) in a hierarchical layered scheme. By means of learning algorithms, different abstractions of input data are learned from lower to higher abstraction, lower in layers near the input data, higher in layers further away.

Deep learning research is mainly motivated by intuition, theoretical arguments from circuit theory, empirical results, and current knowledge of neuroscience. It is a growing trend in machine learning due to favorable results in applications where the target function is complex and the datasets are large. Informally speaking, deep learning methods extract some significant structure from a data set instead of doing fairly shallow or flat statistical inference on structureless data representations such as vectors and matrices of values.

One key difference with other machine learning algorithms is its depth. Support Vector Machines, Naïve Bayes and Decision Trees are other machine learning techniques that are not very deep in terms of stacked layered units, thus they are called shallow architectures. Shallow architectures apply less parameterized transformations to input signals/data as it travels from the input to the output layer. This chain of transformations is named Credit Assignment Path[5], and the usual consensus is that a CAP > 2 means no longer a shallow network and a CAP > 10 means very deep learning.

2.1 Types of machine learning

An important property of neural networks is that they should be teachable. It's simple to show on a small scale how you can supply a series of example inputs and expected outputs and go through a mechanical process to take the weights from initial random values to progressively better numbers that produce more accurate predictions. The problem is to do it in increasingly complex problems such as image and speech recognition where required networks grow in size and along with that the number of weights.

As deep learning is a subfield inside machine learning, it is convenient to cite the types of learning that exist. The usual way of describing an algorithm of machine learning is the supervised/unsupervised classification depending of the goals and the previously known information about input data.

- Supervised learning: it is the task of learning a classification from previously known labeled data. The data consists of the input object and the label or class the object belongs to. The supervised learning algorithm uses the training data to produce a function that will map new examples.

In the best case scenario the algorithm correctly classifies unseen instances into their belonging classes.

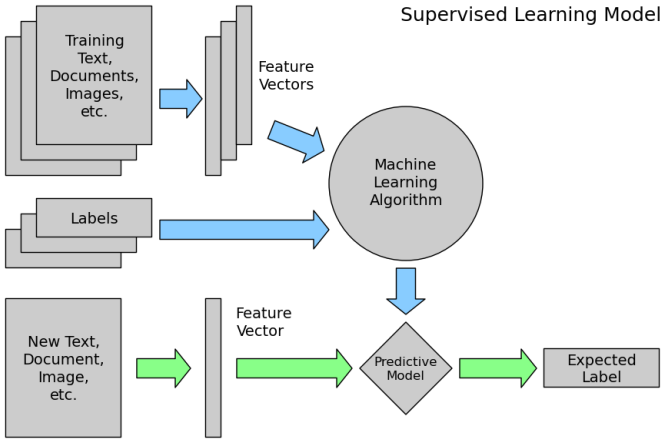

Figure 2: Supervised learning flow chart

- Unsupervised learning: it is the task of trying to find an underlying structure in unlabeled data. Cluster analysis is a common method that will produce clusters of objects depending on the similarity with other object's variables.

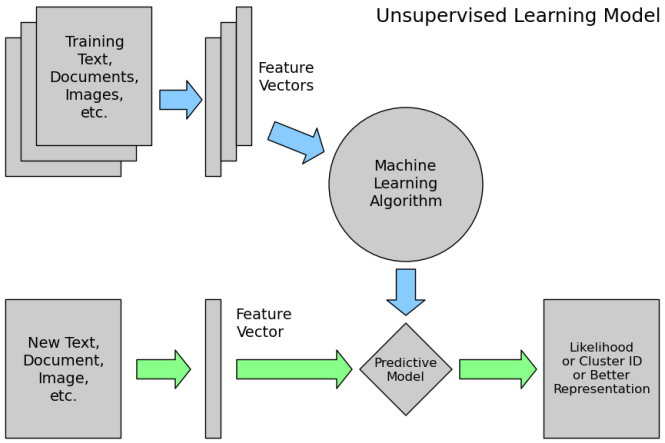

Figure 3: Unsupervised learning flow chart

- Semi-supervised learning: when having a mixed set of labeled and unlabeled data, usually a combination of unsupervised an supervised learning

algorithms.

- Reinforcement learning: it is the task of learning the correct output based on input training data and a signal (reinforcement). It is usually used in agents that need to take actions in an environment. This learning will allow the agent to act in order to maximize a cumulative reward (positive reinforcement), or minimize a penalty (negative reinforcement).

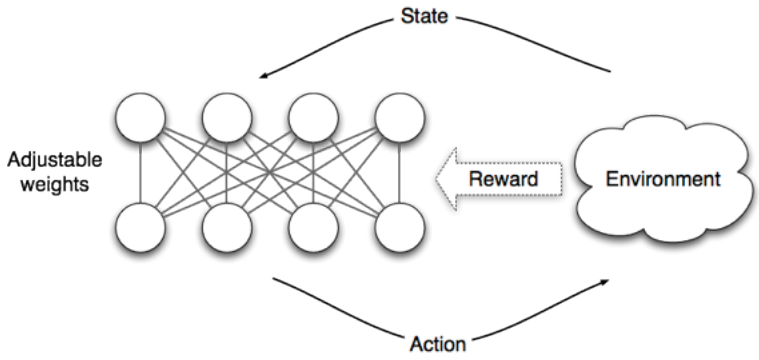

Figure 4: Reinforcement learning

2.2 Unsupervised learning as a deep learning facilitator

A key aspect of latest deep learning research that makes it different from previous multiple layer architectures is its effective feature extraction via initial unsupervised learning. This receives different names such as pre-training, initial learning of features and representation learning and mimics the biological brain's capability of extracting multiple levels of representation from sensory input[6]. It is usually carried out by traditional shallow unsupervised algorithms such as RBMs[1], SVMs[2] or autoencoders.

By encoding the raw input data into a more compact and less redundant way, the unsupervised phase of learning allows later layers to become smaller and shallower than if the network was to deal with untreated data.

As cited earlier, previous multiple layered networks such as the multi-layer perceptron suffered some problems such as the exponentially escalating computational cost with the number of layers and the vanishing of the gradient along the layers. Along this, most deep learning architectures are not fully connected ones; they usually present some sparse connectivity among some of the layers, whereas others may be fully connected feed-forward layers.

[1] Restricted Boltzmann Machine
[2] Support Vector Machine

2.3 Problems when applying deep learning techniques

Deep neural network share most of the issues of standard neural networks, and solutions are proposed for each of these.

Vanishing gradient problem Also called the fundamental deep learning problem, it happens because the gradient is unstable, tending to vanish in earlier (nearer the input) layers because of the use of logistic sigmoid functions as activation functions in the neurons. As the derivative tends to 0 with extreme values of input, it happens that the signal backpropagated to the previous layer gets smaller, vanishes, and consequently the earliest layers learning rate is slowed. The backpropagation algorithm works by first doing a forward pass through the network to get the actual output and then propagating backwards the calculated error (delta) of the output, usually a mean squared error: $E = \frac{1}{2}(t - y)^2$ where t is the target value and y is the actual output. Last, weights are updated according to the gradient obtained multiplying the each neuron's input and it's delta, usually multiplied by a percentage called the learning rate. Stochasticity here means that all training samples are not used in each update, but only randomly chosen one instead.

Several proposed solutions include the cited initial unsupervised learning, LSTM-like[3] networks, increasing computational power by means of modern GPUs, Hessian-free optimization, and lastly searching the networks' space of weights by means of optimization methods and heuristics. Most award winning solutions nowadays employ GPUs combined with the unsupervised representation learning.

Overfitting This happens when the network is trained to the point of (almost) becoming and identity function of the input. This means the network is overfitted to the data initially trained with and will usually perform poorly on new unseen data. An intuitive explanation is that the weights of units become too co-adapted to edges in the network (connections to next layers). With this in mind the solution becomes simple actually: ignore the units with some probability with each of the training samples and then using the averaged weights for new data.

[3] Long Short-Term Memory

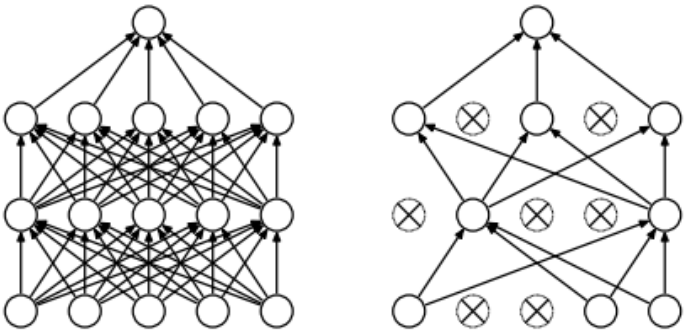

Figure 5: 2 hidden layer neural network and the same network with some units switched off

This regularization technique is called dropout and breaks up these mentioned co-adaptations by making the presence of particular hidden layer units in the network unreliable. Although it may increase two or three times the time required for training the network it eventually reduces overfitting and improves accuracy[7].

Computational cost Even though not fully connected, the computational resources required for learning and also for later classification tasks tend to be very computationally expensive due to the high number of matrix operations for the multiple layers of the network. This decade has seen a rise in the use of massively parallel computing elements present in every computer: the GPU[4]. Initially devised for the massively parallel calculations required by modern 3D applications and especially 3D games, this hardware has shown to be well suited for the parallel calculations also required by deep learning, reaching an average 20x speedup in execution times for common deep learning algorithms[8]. The level of suppport and performance varies with different frameworks and libraries used when building deep learning systems, but most popular options do off-the-shelf support CUDA or OpenCL, two of the most used parallel computing platform and APIs.

2.4 Good deep learning results in competitions

The use of deep learning architectures has improved the results of previous algorithms in fields such as speech recognition, genomics and object recognition. From 2012 and onwards deep learning algorithms are present in top performers and most of the winners of contests and benchmarks such as Kaggle, ImageNet, NIST's OpenHaRT, TIMIT speech recognition, ICDAR Chinese handwriting recognition benchmark or PASCAL object detection, and many recent competitions have deep learning architectures pitted against themselves or similar

[4]Graphics Processing Unit or coloquially graphics card/chip

ones[5].

The fields deep learning techniques may be applied to is increasingly big as data scientist start applying them for data and also for control.

3 Frameworks for deep learning

Deep learning can be applied to different kind of problems, the most popular being classification and lately control and robotics. Even though the field of deep learning is less than a decade old, there are already many frameworks and libraries available for development purposes from where to choose from, with varying levels of complexity, ease of use and requirements.

3.1 Theano

Theano is an open source project developed by the machine learning group at Université de Montréal[6], Yoshua Bengio being the most important researcher. It is a numerical computation library for Python with heavy use of NumPy syntax. It allows to define, optimize and evaluate mathematical expressions involving multi-dimensional arrays efficiently. As stated on its inaugural paper, *"Theano is a compiler for mathematical expressions in Python that combines the convenience of NumPy's syntax with the speed of optimized native machine language"*. Cited advantages are:

- Tight integration with NumPy
- Transparent use of a GPU
- Efficient symbolic differentiation: does derivatives for function with one or many inputs.
- Speed and stability optimizations.
- Dynamic C code generation – Evaluate expressions faster.
- Extensive unit-testing and self-verification.

Following the tutorials a simple experiment is tried to check the performance gain of GPU computation on available computer:

```
from theano import function, config, shared, sandbox
import theano.tensor as T
import numpy
import time
```

[5] For a good compilation of achievements refer to [5], [9] and [10]
[6] http://deeplearning.net/software/theano

```
vlen = 10 * 30 * 768  # 10 x #cores x # threads per core
iters = 1000

rng = numpy.random.RandomState(22)
x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
f = function([], T.exp(x))
print f.maker.fgraph.toposort()
t0 = time.time()
for i in xrange(iters):
r = f()
t1 = time.time()
print 'Looping %d times took' % iters, t1 - t0, 'seconds'
print 'Result is', r
```

Hardware	Avg. execution time
GPU	764ms
CPU	11305ms

Table 1: CPU vs GPU test

The results are averaged in three executions. It is noticeably faster, almost 15x increase in speed with the use of GPU computation. The hardware used is an Intel i5-3570 CPU and a nVidia GeForce 660 GTX GPU in a 64 bits Windows 7 environment.

Theano is very used in academic environments but the use and definition of tensors for most operations is not the most intuitive and can have a steep learning curve. Theano is not a programming language in the normal sense because it involves writing a Python program which builds expressions for Theano. This program is then compiled to special instruction sets, which can include CUDA and OpenCL for GPU computation. Installing Theano and its dependencies is not an easy process at the moment despite numerous tutorials, testing for errors and learning curve seem to be prevalent issues among users[12] [13].

3.1.1 Libraries using Theano

Theano is used for researching and as a base for further building more user-friendly libraries and frameworks. At the time of this writing, two promising libraries using Theano are gaining some attention: Lasagne[14] and Blocks[15]. These two projects aim to make use of Theano's capabilities but presenting them in an easier way.

Pylearn2 is another machine learning framework built with Theano as a base and offers many deep learning methods readily implemented. It is one of the first

frameworks built on top of Theano, being a couple years old. However, as cited on its published paper [17] *"Pylearn2 is a machine learning research library - its users are researchers. This means (...) it is acceptable to assume that the user has some technical sophistication and knowledge of machine learning"*. What this means is that the library is not for production environments, instead knowledge about deep learning and writing some data wrappers in python are required to be able to use it correctly. The main advantage of Pylearn2 is its speed.

3.2 Torch

Using the LUA, a somewhat uncommon scripting language, Torch7, the latest iteration of the Torch framework, provides a similar to MATLAB environment for machine learning algorithms. Torch was developed as a numerical/scientific computing extension of LuaJIT with an ML/neural net library on top by Yann LeCun and his team at NYU. Internally developed in C, it can easily target different machine compilations[18]. Torch has a large ecosystem of community-driven packages for machine learning, computer vision, signal processing and networking among others, so its focus is not solely as a deep learning framework, although a package exists for neural networks. A summary of features is:

- Efficient Tensor library (like NumPy) with an efficient CUDA backend
- Numeric optimization routines
- Neural Networks package with fast CUDA and CPU backends
- Interface to C, via LuaJIT
- Good community and industry support - several hundred community-built and maintained packages
- Easy to use Multi-GPU support and parallelizing neural networks
- Embeddable, with ports to iOS, Android and FPGA back-ends

Top companies are currently using Torch7 to develop deep learning solutions, Google/Deepmind, Facebook and Twitter among others. The main advantages cited are the efficiency of FFT[7] performed in convolutional nets (convnets)[19] and availability of common deep learning algorithms. Here is some sample code for a definition of handwritten digit recognition deep network architecture[8]:

```
net = nn.Sequential()
net:add(nn.SpatialConvolution(1, 6, 5, 5))
-- 1 input image channel, 6 output channels, 5x5 convolution kernel
```

[7] Fast Fourier Transform

[8] https://github.com/soumith/cvpr2015/blob/master/Deep%20Learning%20with%20Torch.ipynb

```
net:add(nn.SpatialMaxPooling(2,2,2,2))
-- A max-pooling operation that looks at 2x2 windows and
-- finds the max.
net:add(nn.SpatialConvolution(6, 16, 5, 5))
net:add(nn.SpatialMaxPooling(2,2,2,2))
net:add(nn.View(16*5*5))
-- reshapes from a 3D tensor of 16x5x5 into 1D tensor of 16*5*5

net:add(nn.Linear(16*5*5, 120))
-- fully connected layer (matrix multiplication between input
-- and weights)

net:add(nn.Linear(120, 84))
net:add(nn.Linear(84, 10))
-- 10 is the number of outputs of the network (in this case,
-- 10 digits)

net:add(nn.LogSoftMax())
-- converts the output to a log-probability.
-- Useful for classification problems.

print('Lenet5\n' .. net:__tostring());
```

The code is pretty straightforward but input data processing must be coded, this is only a definition of the neural network.

Similar to Theano, it is focused on developing and researching novel deep learning network architectures, and using an uncommon language as LUA can be hard to learn initially, distancing a bit the average user from the environment. However, top software companies committing to its use is a sign of maturity of the framework and there are useful guides available to install and use Torch7 [20].

3.3 Caffe

A widely used machine vision library, it was developed by the BVCL[9]with expression, speed, and modularity in mind. It also has a strong community of contributors behind. Cited features are:

- Expressive modular architecture that defines models by configuration and not hard-coding.
- Extensible code easy to extend and contribute to.
- High speed, able to process 60 million ILSVRC2012 images per day[21]

[9]Berkeley Vision and Learning Center

- Community that extends to academic, startup and large-scale industrial projects.
- Python and MATLAB bindings.
- A good array of pre-trained models: modelzoo[10].

Its main applications are in computer vision, as image classification mostly. It is written in C++ with CUDA used for GPU computation and with bindings to Python and MATLAB. As stated in its inaugural paper of 2014 [22] it is very focused on providing a clear separation of model representation and implementation, allowing easy research and experimentation on deep networks.

Framework	License	Core language	Binding(s)	CPU	GPU	Open source	Training	Pretrained models	Development
Caffe	BSD	C++	Python, MATLAB	✓	✓	✓	✓	✓	distributed
cuda-convnet	unspecified	C++	Python		✓	✓	✓		discontinued
Decaf	BSD	Python		✓		✓	✓	✓	discontinued
OverFeat	unspecified	Lua	C++,Python	✓				✓	centralized
Theano/Pylearn2	BSD	Python		✓	✓	✓	✓		distributed
Torch7	BSD	Lua		✓	✓	✓	✓		distributed

Figure 6: Caffe compared to other popular frameworks[22]

As seen in figure 6 caffe prides itself to provide the most features of the compared frameworks, and one of those is deemed to be very important as described in the next subsection.

3.3.1 Use of pretrained models

The modularity and *concern separation* (as seen in object oriented programming) of Caffe, along with its popularity and portability, makes it an ideal platform for testing and tweaking existing models. It is perhaps the best framework to begin using existing deep learning networks, suchs as the publicly available award-winning ones available on the modelzoo website. This allows not only to experiment with the results of famous competitions as ImageNet, ILSVRC or CIFAR, but to employ them on real world situations, industry being one of the most benefited. Popular networks are named after their creators: AlexNet, CaffeNet, LeNet or GoogLeNet.

This feature, along with its popularity and no lack of important features present in other frameworks to experiment with deep neural network is enough to tip the scales in favor of the caffe framework.

[10] http://caffe.berkeleyvision.org/model_zoo.html

A simplified LeNet[23] network is defined in Python code along with the training and testing datasets in LMDB format, and then writen in two a human-readable Google's protocol buffer[11] format configuration files named *lenet_auto_train.prototxt* and *lenet_auto_test.prototxt* for training and testing respectively:

```
from caffe import layers as L
from caffe import params as P

def lenet(lmdb, batch_size):
# a simple version of LeNet
n = caffe.NetSpec()
n.data, n.label = L.Data(batch_size=batch_size, backend=P.Data.LMDB,
  source=lmdb,
transform_param=dict(scale=1./255), ntop=2)
n.conv1 = L.Convolution(n.data, kernel_size=5, num_output=20,
  weight_filler=dict(type='xavier'))
n.pool1 = L.Pooling(n.conv1, kernel_size=2, stride=2, pool=P.Pooling.MAX)
n.conv2 = L.Convolution(n.pool1, kernel_size=5, num_output=50,
  weight_filler=dict(type='xavier'))
n.pool2 = L.Pooling(n.conv2, kernel_size=2, stride=2, pool=P.Pooling.MAX)
n.ip1 = L.InnerProduct(n.pool2, num_output=500,
  weight_filler=dict(type='xavier'))
n.relu1 = L.ReLU(n.ip1, in_place=True)
n.ip2 = L.InnerProduct(n.relu1, num_output=10,
  weight_filler=dict(type='xavier'))
n.loss = L.SoftmaxWithLoss(n.ip2, n.label)
return n.to_proto()

with open('examples/mnist/lenet_auto_train.prototxt', 'w') as f:
f.write(str(lenet('examples/mnist/mnist_train_lmdb', 64)))

with open('examples/mnist/lenet_auto_test.prototxt', 'w') as f:
f.write(str(lenet('examples/mnist/mnist_test_lmdb', 100)))
```

Next, the network's training and learning parameters are specified in a similar protocol buffer file, where parameters such as learning rate, optimization method (SGD) and weight decay are specified:

```
train_net: "examples/mnist/lenet_auto_train.prototxt"
test_net: "examples/mnist/lenet_auto_test.prototxt"
test_iter: 100
test_interval: 500
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
```

[11] https://developers.google.com/protocol-buffers/?hl=en

```
lr_policy: "inv"
gamma: 0.0001
power: 0.75
display: 100
max_iter: 10000
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
```

And lastly, CPU or GPU usage may be specified before executing the training:

```
#caffe.set_device(0)
#caffe.set_mode_gpu()
caffe.set_mode_cpu()
solver = caffe.SGDSolver('examples/mnist/lenet_auto_solver.prototxt')

for it in range(niter):
  solver.step(1)
```

As seen, caffe looks well as an initial approach to experimenting with deep learning, not overly complex, with easy Python integration and with a lot of readily available materials as networks and training sets by the community, and also is seen in almost all top computer vision competitions.

4 Convolutional neural networks for visual object recognition

Vision tasks fundamentally rely on the ability to recognize scenes, objects and categories. Learning and matching visual objects is difficult for a number of reasons[24]:

- Variable illumination, pose, alignment, viewpoint...
- Variations among same category objects (brands of cars, cat breeds...)
- Variable backgrounds that may clutter the object
- Possible ambiguities

In summary, even the same identical object can cast an infinite amount of different bidimensional projections onto the retina or camera sensor. Hence, high robustness to these distortions is required for any good visual object recognition system.

4.1 Overview

When it comes to computer vision and visual object recognition many techniques have been tried in the past with increasing levels of success. Edge detectors, template matching, Gabor filters, Bag of Words, SIFT ,SURF... some techniques involved some kind of learning, such as support vector machines, but usually very shallow learning happened [25]. With the use of modern object recognition architectures such as the deep convolutional neural network have achieved great success in benchmarks, to the point of surpassing human accuracy levels [36]

Convolutional neural networks or convnets in short, were significantly invented and pushed forward by Yann LeCun from NYU and Joshua Bengio from the Université de Montréal. A 1998 seminar paper summarizes his and his colleagues' work on document recognition and especially OCR[12] or handwritten letter recognition[27]. Three factors are cited as the reasons for advances in object recognitionf:

- Availability of low-cost machines with fast arithmetic units allows for reliance on more brute-force "numerical" methods than on algorithmic refinements.
- Availability of large databases for problems with a large market and wide interest, such as handwriting recognition, allowing experimentation on more real data.

[12] Optical Character Recognition

- Availability of powerful machine learning techniques that can handle high-dimensional inputs and can generate intricate decision functions when fed with these large data sets.

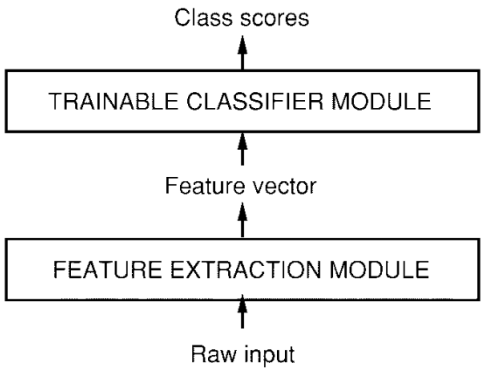

Figure 7: The two parts of pattern recognition

At the time convolutional neural networks were used to analyze bank cheque letter and numbers that were written by hand. Previously, hand-crafted features were used for patter recognition systems, and the paper showed that better systems can be built by relying more on automatic learning and less on hand-designed heuristics. This may sound familiar to the deep learning's successful unsupervised feature learning, and it is, in fact, related.

4.2 Issues with traditional neural networks

The ability of multilayer networks such as the MLP[13] trained with gradient descent to learn complex, high-dimensional, nonlinear mappings from large collections of examples makes them obvious candidates for image recognition tasks. Nevertheless, as seen in the overview of deep learning issues arise with standard MLP-like neural networks.

First, images being usually hundreds of pixels large, fully connected networks find themselves with hundreds of thousands of weights in the first hidden layer. Also, an important deficiency with of these nets is that they have no built-in invariance with respect to translations or distortions of the inputs. Before being fed into the network, the input must be centered and size normalized. Unfortunately, this preprocessing can never be perfect.

In relation to the distortions of the inputs, different writing styles can be assumed. This causes variations in the position of distinctive features of the

[13] Multi-layer Perceptron

input characters, vital for discriminating among them. In principle, a fully connected network of sufficient size is able to learn to produce outputs that are invariant with respect to such variations. However, learning such a task would probably result in multiple units with similar weight patterns positioned at various locations in the input in order to detect distinctive features wherever they appear on the input (that common strike in the handwritten 7 number for instance). Learning these weight configurations requires a very large number of training instances to cover the space of possible variations.

The last issue of fully connected architectures is that the topology of the input is ignored. The input variables can be presented in any (fixed) order without affecting the outcome of the training. On the contrary, images have a strong bidimensional local structure: variables (or pixels) that are spatially or temporally nearby are highly correlated. Local correlations are the reasons for the well-known advantages of extracting and combining local features before recognizing spatial or temporal objects, because configurations of neighboring variables can be classified into a small number of categories (e.g., edges, corners, etc.).

Summarizing, standard fully connected feed-forward neural network architectures for visual image recognition:

- Increase size exponentially with the size of the input objects.
- Tend to learn similar weights in multiple locations of the weight space because of the localization of features of same class objects.
- Ignore the topology of the input, where high correlations exist in nearby pixels.

4.3 The convolutional filter

A convolutional neural network is a specialized neural network architecture incorporating knowledge about the invariances of bidimensional shapes by using local connection patterns and imposing constraints on the weights.

A convolutional neural network overcomes the problems cited in the previous paragraphs by tricks inspired by the study of biological visual systems[28]. It uses a series of filters that are repeatedly applied multiple times at different sub-regions of the image, thus, by convolution of the input image with a linear filter. These filters share the same weights and are able to capture features from the input image and learn in an unsupervised fashion. So, overcoming the three problems cited:

- Convolutional networks have shared weights, thus less weights.
- Convolutional networks automatically obtain shift invariance by forcing the replication of weight configurations across space.

- Convolutional networks force the extraction of local features by restricting the receptive fields of hidden units to be local.

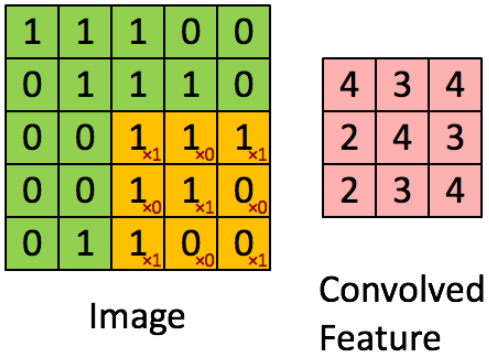

Figure 8: The convolution filter

The convolutional filter works by convolving a kernel on the raw input image and extracting a feature. As the filtering is done in multiple subregions of the image, a feature map is obtained. A single layer contains multiple convolutional filters of a certain size, 3x3 or 5x5 for example. The feature map is slightly smaller in size due to this process. The exact values of the filter are the weights of the convolutional unit, and every convolutional filter uses these same weights, thus the weight sharing. These filters could be typically used gaussian blur kernels or gabor kernels in order to get a smoothed image or gradient features of the original image. However, the goal of convolutional networks is not to use pre-existing kernels but to learn set-specific filters from training sets.

The second part of the layer is the subsampling filtering. This reduces the dimensionality of the features and helps increase the translation invariance of the subsequent filters[29]. Several methods can be used, 2x2 subsampling and max-pooling being common.

The deep part of the network comes from stacking these layers several times. The last layer of the convolutional networks is always a classifying layer, a typical fully connected layer that outputs the category detected by the whole network in terms of ouput unit activations. The figure 9 shows a simple two layer convolutional layer network.

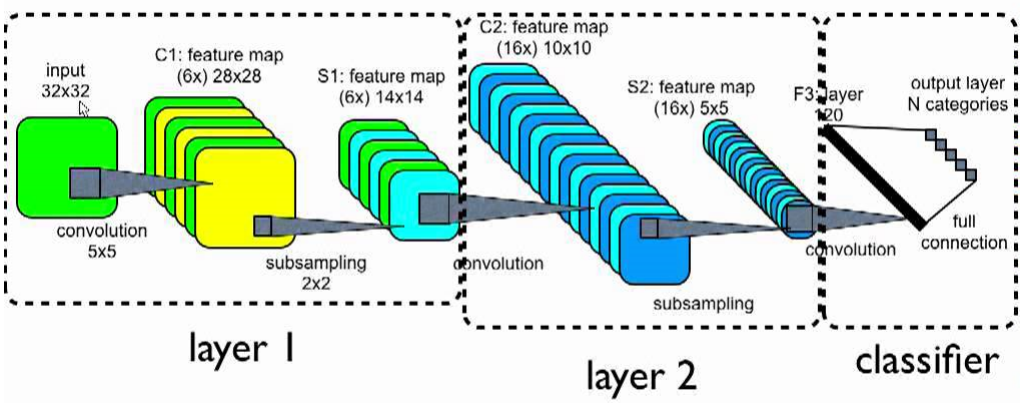

Figure 9: The simple (by today standards) two convolution layer LeNet neural network[23]

As seen, a convolutional neural network has many hyperparameters that are left to the user's discretion:

- Number of filters per convolutional layer
- Filter size and stride: 5x5, 3x3, 7x7 are common kernel sizes and 1 or 2 common strides[14].
- Subsampling size and method: Maxpooling,
- Number of layers

These hyperparameters are often responsible for the success (or lack of thereof) of convolutional neural network architectures. The best way of developing a robust object recognition system is to imitate the trends of recent successes in competitions such as CIFAR-10 or ImageNet; these models are usually available in the form of papers or downloadable trained models for caffe in its modelzoo[15].

4.4 Modern deep convolutional networks

The LeNet model seen in the previous section is over a decade old, but it is still fundamentally the basis for modern award-winning deep convnets. It is a field of ongoing research and gradual advances according to successes in object recognition competitions as ImageNet or CIFAR-10. The figure 10 shows the results of modern convnets in past VOC challenges, they undoubtedly surpass previous visual object recognition methods.

[14]Stride measures the distance between filtered subregions
[15]http://caffe.berkeleyvision.org/model_zoo.html

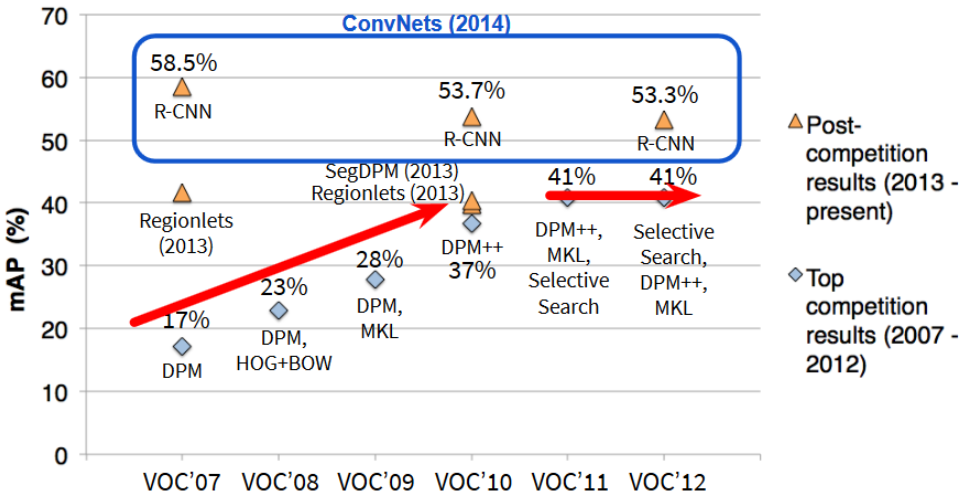

Figure 10: Modern deep convolutional network successes in the PASCAL Visual Object Challenge[36]

5 LeNet example with caffe

A tweaked LeNet network is designed and trained for classification of the MNIST handwritten digit set. The network is shown in figure 12 and the listing in Appendix A is the protocol buffer caffe format description of it. The network has two convolutional layers with their subsampling, performed by max-pooling. Max-pooling extracts the maximum value for a subregion and is useful because of two reasons. First, it is a good subsampling technique because reduces computation by eliminating non-maximal values for the subregions. Second, it helps with the translation invariance.

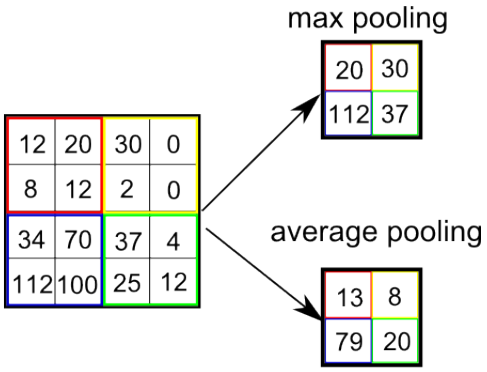

Figure 11: 2x2 subregion max-pooling compared to average-pooling

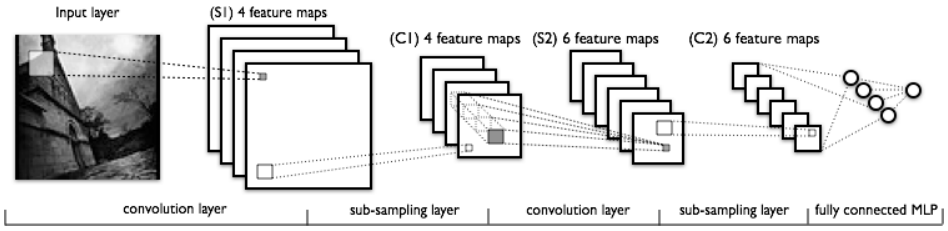

Figure 12: Another look at the LeNet convolutional network[23]

By means of the straightforward python and caffe interaction, courtesy of the pycaffe project, experimenting and visualizing all the training and evaluation processes of the LeNet is quite simple. First, the MNIST training and test data sets are downloaded and converted into LMDB[16] format by means of a shell script. This format is useful when dealing with datasets, as it is offers good performance for reading and writing key-value pairs.

[16]Lightning Memory-Mapped Database

These two datasets are fed into the network as specified in the learning parameters file, also listed in Appendix A. It is possible to visualize the first two elements of each dataset thanks to python's image visualizing tools:

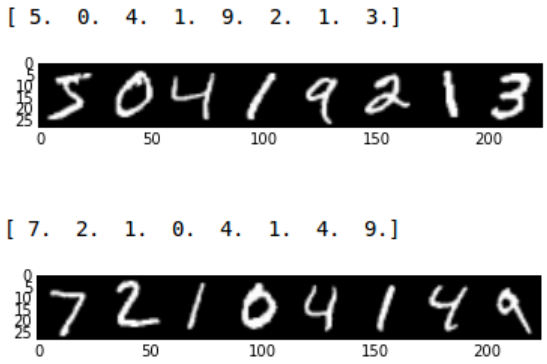

Figure 13: First 8 training (top) and testing (bottom) inputs

Afterwards, a single minibatch of 100 input values training is performed and first convolutional filters visualized in order to check the training is taking place. Batch training approximates the gradient of multiple values (100 in this case), thereby reducing some of the noise that happens when updating every single input value.

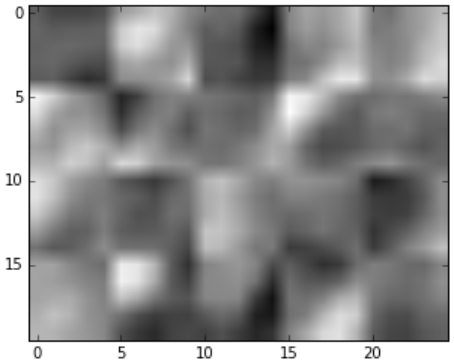

Figure 14: Conv1 layer visualization

This means that learning is being done and the convolutional layers are learning their filters (weights). 200 minibatch iterations are performed for a total 200 000 training images shown to the network. The test set is used to see check the accuracy of the network.

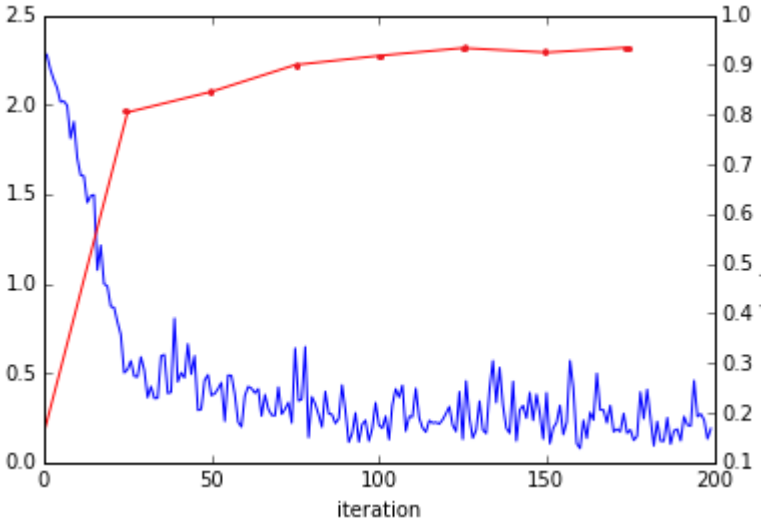

Figure 15: Accuracy (red, right axis) and train loss (blue, left axis) across 200 minibatch iterations

Lastly, every 25 iterations the first training set is checked to see the gradual improvements to the neural network. Brighter means higher activation values for the output neurons. Figure 16 shows the accuracy with same three input digits across the 10 output units. This can be interpreted as 7 and 1 being easily classified correctly whereas withe the digit 2 the network's confidence is lower.

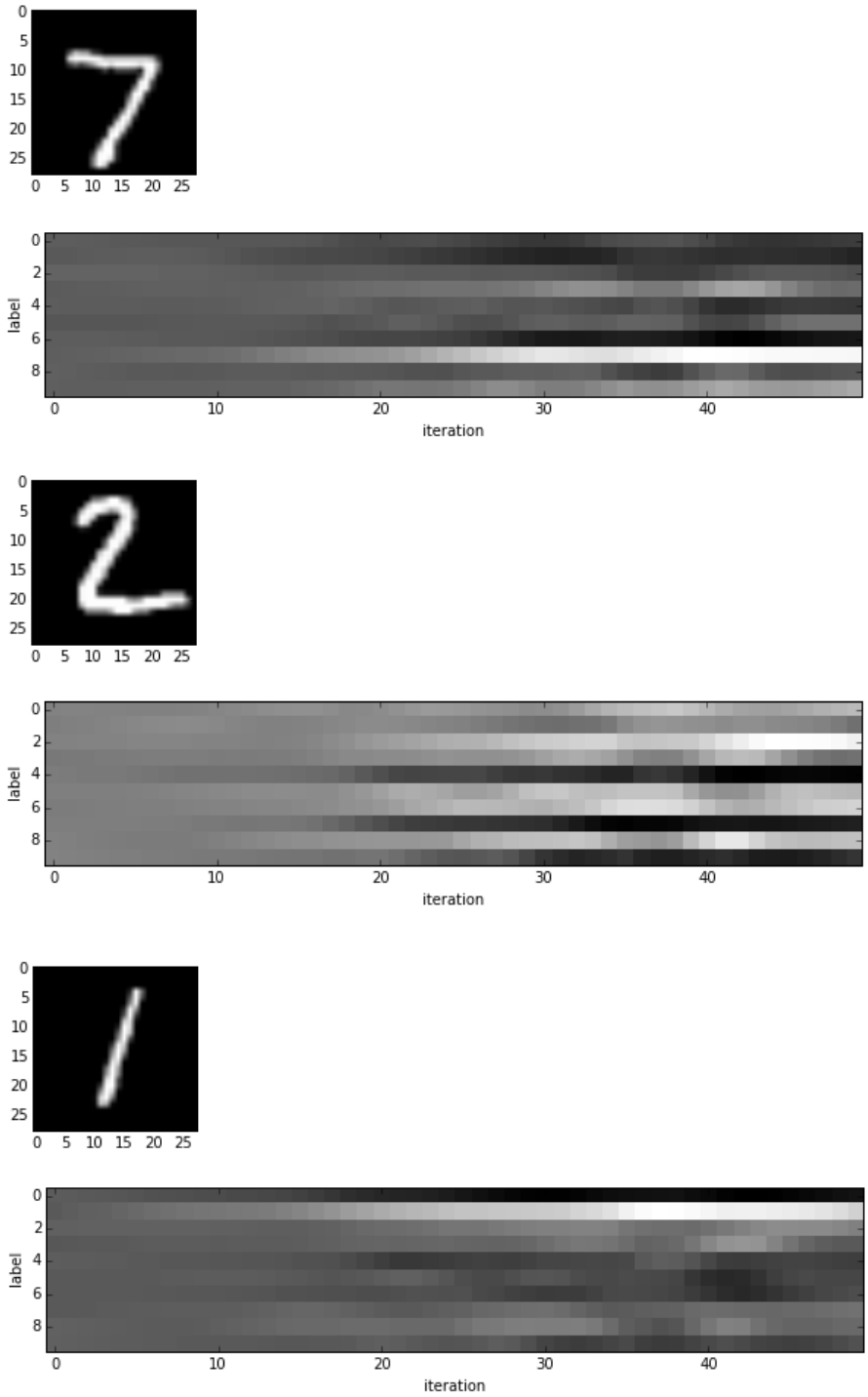

Figure 16: 3 input accuracy across the training iterations

6 Deep learning use cases

This section will present two examples of state-of-the-art deep learning use cases. The first one belongs to a AI research company recently purchased by Google for $400 million[17], and the second one is small trial performed in collaboration with Ibermatica for image recognition and location in aerial drone images. Reviewing these two deep learning applications helps gaining understand of the detailed inner workings of convolutional neural networks used in both cases, along with many other implementation details necessary in order to successfully apply deep learning methods in any domain.

6.1 Playing atari games

Google's recent success in the area of reinforcement learning with deep neural networks is an interesting case. The task to develop intelligent agents acting in a virtual environment is not an easy one, especially if the environment consists of decades old videogames' emulator. In this case, the access to the environment is done solely via visual recognition; the agent has no access to the emulated memory or any other variables other than those a human player would have standing in front of the atari videogames console. This is an important constraint and demonstrates improving levels of intelligence and visual recognition.

Google's recently purchased Deepmind London-based AI company [30], has recently succeeded in showing a system with human-level proficiency at playing simple videogames. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards[31]. This kind of reinforcement learning is different from most classification problems; most deep learning algorithms assume independent data samples, whereas in reinforcement learning there are frequently sequences of highly correlated states. Also, as the intelligent agent interacts with the evironment, data distribution changes as the algorithm learns new behaviors.

Figure 17: Screenshots of five of the Atari 2600 games tested

Focusing on the visual recognition part of the architecture, a raw atari game outputs 210x160 pixel images with a 128 color palette to the screen. Dimen-

[17] http://www.technologyreview.com/news/524026/is-google-cornering-the-market-on-deep-learning/

sionality is reduced for decreasing the computational demand, down to a 110x84 grayscale image and then cropping it to a square 84x84 region that roughly captures the playing area[33, 34]. Two convolutional layers are used, the first one with 16 8x8 filters with a step size of 4, producing 16 20x20 feature maps, and the second consists in 32 4x4 filters with a step size of 2, yielding 32 9x9 values. The last two layers are traditional fully connected ones. The output layer is a unit per possible 18 actions in the game: 8 directions or a 9th no direction, with a possible button press.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 18: Deep Q learning algorithm

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]. \quad (3)$$

Figure 19: A deep Q network can be trained by minimizing L loss functions that change at each i iteration. Differentiating with respect to the weights the shown gradient is arrived at.

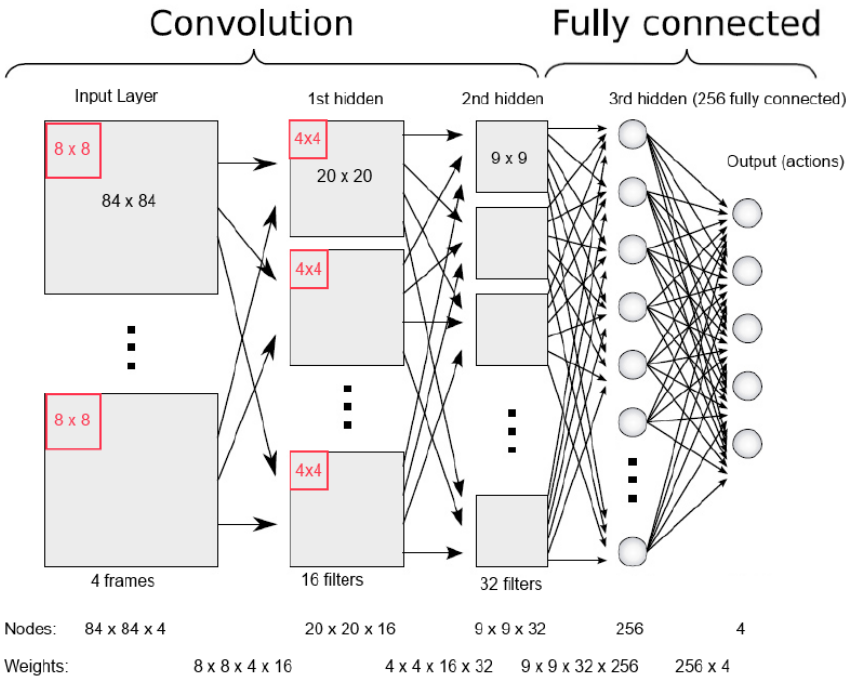

Figure 20: The details of the convolutional neural network[34]

The results are quite impressive: out of 47 games tested, the algorithm achieves more than the 75% of the human score on 29 games. The games are varied in their nature, from side scrollers to pseudo-3D racing games and boxing games. In some of the games such as breakout, the the deep Q network is able to discover long-term strategies. The network, however, fails to achieve good performance in games requiring more temporally extended planning strategies. The algorithm also outperforms past algorithms.

7 Conclusions and future work

This work has summarized the current state of the art of convolutional neural network and has shown two applications, the original handwritting digit recognition for which the LeNet architecture was first designed, and one of the most modern examples, Google's Atari-playing AI which gained widespread attention this last year.

It has been shown that deep neural networks not only promise good results in theory, overcoming many problems of traditional feed-forward neural networks, they do deliver and surpass other algorithms in competitions thanks to their abstraction capabilities.

Increasing computational power thanks to Moore's law and the devise of better and better algorithms such as the convolutional filters combined in deep architectures, seem to hold the key for even better results in the short term future.

In more practical matters, applying modern award-winning architectures to visual recognition problems would be the next step, testing in the process the effectiveness of modern deep learning enhancing hardware such as GPUs. Different architectures could be tried and compared for their performance, computational cost and other factors. The simplicity the caffe framework provides when it comes to download, test and tweak pretrained models is a positive impact in any future work.

The potential of making sense about images can be used to train networks to recognize physical objects from video and images. Due to the training and computational cost of using deep neural networks, a study of their performance according to different hardware is needed in order to ascertain their usefulness in on-line environments where other shallow algorithms are common.

Object location is another possible future work. As convolutional architectures work well identifying and classifying objects, their position on the input images must be worked out in an external way, probably by computing bounding boxes and feeding different image regions to the network, resulting in an object locator architecture. See [37] and [38] for current work on this subject.

References

[1] McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics 5 (4)

[2] Hinton, G. E., Osindero, S. and Teh, Y., A fast learning algorithm for deep belief nets Neural Computation 18:1527-1554, 2006

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici and Hugo Larochelle, Greedy Layer-Wise Training of Deep Networks, in J. Platt et al. (Eds), Advances in Neural Information Processing Systems 19 (NIPS 2006), pp. 153-160, MIT Press, 2007

[4] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra and Yann LeCun Efficient Learning of Sparse Representations with an Energy-Based Model, in J. Platt et al. (Eds), Advances in Neural Information Processing Systems (NIPS 2006), MIT Press, 2007

[5] Schmidhuber, Juergen. "Deep Learning in Neural Networks: An Overview". Neural Networks, Vol 61, pp 85-117, Jan 2015 http://arxiv.org/abs/1404.7828

[6] Hinton, Geoffrey E. "Learning multiple layers of representation." Trends in cognitive sciences 11.10 (2007): 428-434.

[7] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

[8] GPU for Deep Learning Algorithm - http://www.cs.rochester.edu/ yli/files/report-gpu.pdf

[9] Deep Learning Successes Obtained by IDSIA - http://deeplearning.net/2013/10/08/deep-learning-successes-obtained-by-idsia/

[10] Deep Learning's Accuracy - http://deeplearning4j.org/accuracy.html

[11] Bergstra, James, et al. "Theano: A CPU and GPU math compiler in Python." Proc. 9th Python in Science Conf. 2010.

[12] https://www.quora.com/What-is-it-like-to-use-Theano-in-Python

[13] https://www.reddit.com/r/MachineLearning/comments/2c9x0s/best_framework_for_deep_neural_nets/

[14] Lasagne home site - https://github.com/Lasagne/Lasagne

[15] Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio, "Blocks and Fuel: Frameworks for deep learning," arXiv preprint arXiv:1506.00619 [cs.LG], 2015.

[16] Pylearn2 in practice - http://fastml.com/pylearn2-in-practice/

[17] Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., ... & Bengio, Y. (2013). Pylearn2: a machine learning research library. arXiv preprint arXiv:1308.4214.

[18] http://torch.ch/

[19] https://research.facebook.com/blog/879898285375829/fair-open-sources-deep-learning-modules-for-torch/

[20] "Getting Started with Torch7" - http://code.madbits.com/wiki/doku.php?id=tutorial_basics

[21] " Caffe Performance and Hardware Configuration" - http://caffe.berkeleyvision.org/performance_hardware.html

[22] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... & Darrell, T. (2014, November). Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the ACM International Conference on Multimedia (pp. 675-678). ACM.

[23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Comput., 1(4):541–551, December 1989.

[24] Pinto N, Cox DD, DiCarlo JJ (2008) Why is Real-World Visual Object Recognition Hard? PLoS Comput Biol 4(1): e27. doi:10.1371/journal.pcbi.0040027

[25] Grauman, K., & Leibe, B. (2010). Visual object recognition (No. 11). Morgan & Claypool Publishers.

[26] "The Revolutionary Technique That Quietly Changed Machine Vision Forever" MIT Technology Review 2014 - http://www.technologyreview.com/view/530561/the-revolutionary-technique-that-quietly-changed-machine-vision-forever/

[27] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[28] Matsugu, M., Mori, K., Mitari, Y., & Kaneda, Y. (2003). "Subject independent facial expression recognition with robust face detection using a convolutional neural network" (PDF). Neural Networks 16 (5): 555–559. doi:10.1016/S0893-6080(03)00115-1.

[29] Scherer, D., Müller, A.,& Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In Artificial Neural Networks–ICANN 2010 (pp. 92-101). Springer Berlin Heidelberg.

[30] "Google buys UK artificial intelligence startup Deepmind for £400m" The Guardian - http://www.theguardian.com/technology/2014/jan/27/google-acquires-uk-artificial-intelligence-startup-deepmind

[31] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[32] Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

[33] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ...& Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

[34] Korjus, K., Kuzovkin, I., Tampuu, A., & Pungas, T. Replicating the Paper "Playing Atari with Deep Reinforcement Learning"[MKS.

[35] Ciresan, D., Meier, U., & Schmidhuber, J. (2012, June). Multi-column deep neural networks for image classification. In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on (pp. 3642-3649). IEEE.

[36] Pierre Sermanet, Google Research - Object Detection with Deep Learning http://bigdata.memect.com/?p=11488

[37] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014, June). Rich feature hierarchies for accurate object detection and semantic segmentation. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on (pp. 580-587). IEEE.

[38] "R-CNN: Regions with Convolutional Neural Network Features" - GitHub - https://github.com/rbgirshick/rcnn

8 Appendix A - Caffe description model of the LeNet network

```
layer {
        name: "data"
        type: "Data"
        top: "data"
        top: "label"
        transform_param {
                scale: 0.00392156862745
        }
        data_param {
                source: "examples/mnist/mnist_train_lmdb"
                batch_size: 64
                backend: LMDB
        }
}
layer {
        name: "conv1"
        type: "Convolution"
        bottom: "data"
        top: "conv1"
        convolution_param {
                num_output: 20
                kernel_size: 5
                weight_filler {
                        type: "xavier"
                }
        }
}
layer {
        name: "pool1"
        type: "Pooling"
        bottom: "conv1"
        top: "pool1"
        pooling_param {
                pool: MAX
                kernel_size: 2
                stride: 2
        }
}
layer {
```

```
        name: "conv2"
        type: "Convolution"
        bottom: "pool1"
        top: "conv2"
        convolution_param {
                num_output: 50
                kernel_size: 5
                weight_filler {
                        type: "xavier"
                }
        }
}
layer {
        name: "pool2"
        type: "Pooling"
        bottom: "conv2"
        top: "pool2"
        pooling_param {
                pool: MAX
                kernel_size: 2
                stride: 2
        }
}
layer {
        name: "ip1"
        type: "InnerProduct"
        bottom: "pool2"
        top: "ip1"
        inner_product_param {
                num_output: 500
                weight_filler {
                        type: "xavier"
                }
        }
}
layer {
        name: "relu1"
        type: "ReLU"
        bottom: "ip1"
        top: "ip1"
}
layer {
        name: "ip2"
        type: "InnerProduct"
        bottom: "ip1"
        top: "ip2"
```

```
        inner_product_param {
                num_output: 10
                weight_filler {
                        type: "xavier"
                }
        }
}
layer {
        name: "loss"
        type: "SoftmaxWithLoss"
        bottom: "ip2"
        bottom: "label"
        top: "loss"
}


## The learning parameters listing:

train_net: "examples/mnist/lenet_auto_train.prototxt"
test_net: "examples/mnist/lenet_auto_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
```