Deep Learning for Improved Consumer Devices & Services

1. INTRODUCTION TO DEEP LEARNING 1

1.1 DEEP LEARNING & CONSUMER ELECTRONICS 2
1.2 NEURAL NETWORKS - WHAT THEY ARE AND WHAT THEY ARE USED FOR 2
1.3 TRAINING THE NETWORK 2
SUPERVISED VS UNSUPERVISED LEARNING 3
1.4 INSIDE THE NEURAL NETWORK 4
1.5 A RENAISSANCE FOR NEURAL NETWORKS? 5

2. CONVOLUTIONAL NEURAL NETWORKS 6

2.1 CONVOLUTION AND ITS ROLE IN A NEURAL NETWORK: 6
2.2 DEEP NEURAL NETWORKS 9
2.2.1 TYPICAL DEEP NEURAL NETWORKS: 9
2.2.2 FULLY CONVOLUTIONAL NETWORKS (FCN): 10
2.2.2 AUTO-ENCODERS: 11
2.2.4 RECURRENT NEURAL NETWORKS (RNN): 12

3. STATE OF ART TODAY 12

3.1 MAIN ENABLING TECHNOLOGIES (OVERVIEW) 12
3.2 EXAMPLE APPLICATIONS 14

4. CONCLUSIONS 15

REFERENCES 15

1. Introduction to Deep Learning

In the last few years we have witnessed an exponential growth in research activity into the advanced training of convolutional neural networks (CNN) – a field which has become known as “Deep Learning”. This has been triggered by a combination of the availability of significantly larger data-sets, thanks in part to a corresponding growth in “Big Data”, and the arrival of new GPU-based hardware that enables these large data-sets to be processed in reasonable time-scales. Suddenly a myriad of long-standing problems in machine learning, artificial intelligence and computer vision have seen significant improvements, often sufficient to break through long-standing performance thresholds. Across multiple fields these achievements have inspired the development of improved tools and methodologies leading to even broader applicability of Deep Learning. The new generation of smart assistants – Alexa, Hello Google and others – have their roots and learning algorithms tied into deep learning. In this article we review the current state of Deep Learning, explaining what it is, why it has managed to improve on the longstanding techniques of conventional neural networks and, most importantly, how you can get started with adopting deep learning into your own research

activities to solve both new and old problems and build better, smarter consumer devices & services.

1.1 Deep Learning & Consumer Electronics

There has perhaps never been a better time to take advantage of the power of deep learning in consumer products. In retrospect, we may consider 2016 the year that deep learning toolkits and techniques matured from tools that were mostly oriented to researchers into easily used product-enabling technology that can be used to add "intelligence" to almost any consumer device even by non-experts. We expect to see an explosion in products that take advantage of these resources in the coming years, with early adopters differentiating themselves from competitors and further refinement of technology and deep learning methods.

In this article, we hope to provide you with the tools and understanding to start using deep learning today, or to understand what consumer devices that are built using this technology can do and how they work.

1.2 Neural Networks - what they are and what they are used for

Artificial neural networks (ANN) are able to learn something about what they "see" and then "**generalize**" that knowledge to examples (or samples) that they have never seen before [1]. This is a very powerful capability that humans often take for granted because our brains do it so well automatically. You are able to understand the concept of a rock after seeing and perhaps touching very few examples of rocks. From that point on you can identify any rock, even those that are shaped differently or have different colours or textures from the rocks you've seen before. This approach can be seen as opposed to the traditional method of "teaching" or explicitly programming computers based on detailed "rules" that must cover every possible outcome.

The process of discerning the category to which a piece of data belongs is called a **classification task**; one of the more famous uses of this technique is that of training a neural network. The ability to classify unseen examples is referred to as "**generalization**".

Not surprisingly, Artificial Neural Networks are especially powerful in tasks for which the appropriate outcome cannot be determined beforehand and thus cannot use traditional pre-programmed rules [2].

1.3 Training the Network

Artificial Neural Networks (ANN) are not the only techniques that can do this, and much of the terminology we use when talking about ANNs comes from other fields such as statistics. The process of "teaching" our network is called "**training**". When we train a network, what we are formally doing is "**fitting**" a network to our **training** (data-)**set**. This language is borrowed from mathematics where we may try to find the best way to "**fit**" information to a regression line or other mathematical model. The "**training set**" is the sample information that we think is sufficiently representative that our network should be able to learn from it. The thing we want our network to learn to do is called the "**task**".

When training Artificial Neural Networks, we want the network to perform well at a given task on unseen information. When an ANN is not trained sufficiently to do this, we call it "**underfitting**". Which means that the network did not sufficiently learn the training set. The opposite of this is called **overfitting**, where the network learns the training set so well that it cannot effectively be applied to data that it has not seen before [3]. These concepts can be best understood by referring

to Figure 1 which illustrates a simple 2D data-set. Note that in practice we deal with multi-dimensional data leading to far more complex fitting problems.

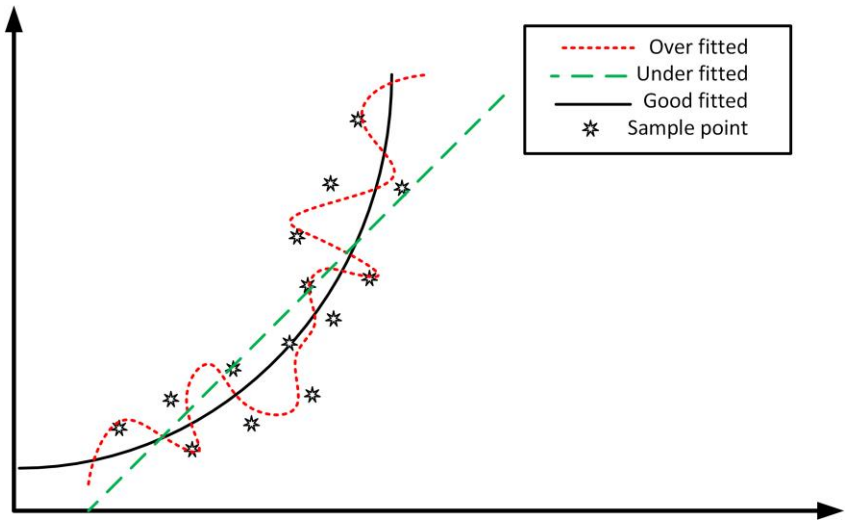

Figure 1- The red dashed line is an example of overfitting. It's not likely to work as well as the solid black line at predicting future values. The straight green dashed line is an example of under fitting. Choosing an overly complex model (such as a high degree polynomial that generates the red line leads to overfitting but picking a model that is too simple (the straight line) causes under fitting. Our goal is often to find the best model to fit the data.

If a model is underfitting, we can increase the number or size of parameters or improve the type of model. If a model is overfitting, we can reduce the size of the model or apply other techniques which we will describe later. A key challenge is how to accurately identify if the model is close to optimal fitting, and this is one of the reasons that very large data-sets are needed to achieve working solutions for these problems.

We measure the suitability of a model for a given task by withholding some of the information that we have from the training process so that we can evaluate the model during and after training. We typically separate this withheld information into two parts: the **validation set** and the **training set**. The information that we use to evaluate our model on during training is called the "**validation set**". This data is never used directly for training. It is only used to provide us with information about how well the network performs during the training process.

Supervised Vs Unsupervised Learning

There are two primary types of learning that neural networks can do: supervised learning, in which the data is labelled or annotated in some way and the task is to somehow learn to match the data with the labels, and unsupervised learning where there are no labels and the neural network learns to find relationships between data [4].

A common example of supervised learning is "classification" where we try to find a category (or label) that can successfully **discriminate** between each class using the supplied labels and data. An example of unsupervised learning is "**clustering**" where the neural network tries to separate data into "chunks" or groups without anyone giving the network any labels to apply to the groups it finds. The appeal of the latter approach is that creating "**ground truth**" labels is a time consuming and often expensive process which requires humans to create appropriate labels for the training, validation, and testing sets before a model can be trained and later placed into products in the real world with unseen (and unlabelled) data.

Despite the cost and difficulty in data preparation, much of the success of neural networks comes from supervised learning.

1.4 Inside the Neural Network

We've discussed what neural networks can do but we've not discussed the details of how they do it. We will now describe the details of how this works. Neural networks typically have an **input layer**, an **output layer**, and 1 or more so called "hidden layers".

These layers are full of nodes, often called neurons which are connected to subsequent and previous layers using a number of schemes.

Perhaps the most common connection scheme is the **fully connected** layer where every neuron in a layer is connected to every neuron in the previous and next layer.

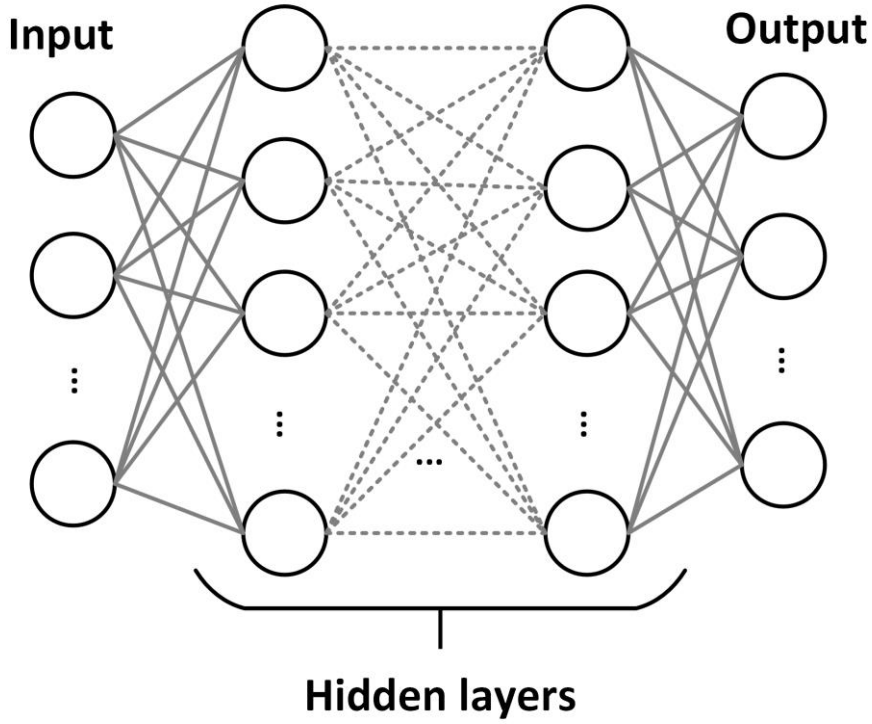

Figure 2 Typical Artificial Neural Network with a fully connected hidden layer.

This idea is inspired from biological neurons where we have axons and dendrites that connect individual neurons to each other. In the biology model axons receive input from other neurons and dendrites transmit information to other cells [5]. This corresponds to the input and output connections in a neural network. The concept of multiple connection schemes also comes from biology where we see Unipolar, Biploar, Multipolar, and Pseudounipolar connection mechanisms.

The body of a biological neuron is called the Soma which can decide to when and what to transmit based on various criteria. Artificial neurons have a similar mechanism called the "activation function".

This model of neurons was first invented in 1943 by Waren McCulloch and Walter Pitts [6]. The first popular implementation of these on computers was formalized in the idea of the Perceptron in 1957 by Frank Rosenblatt [7].

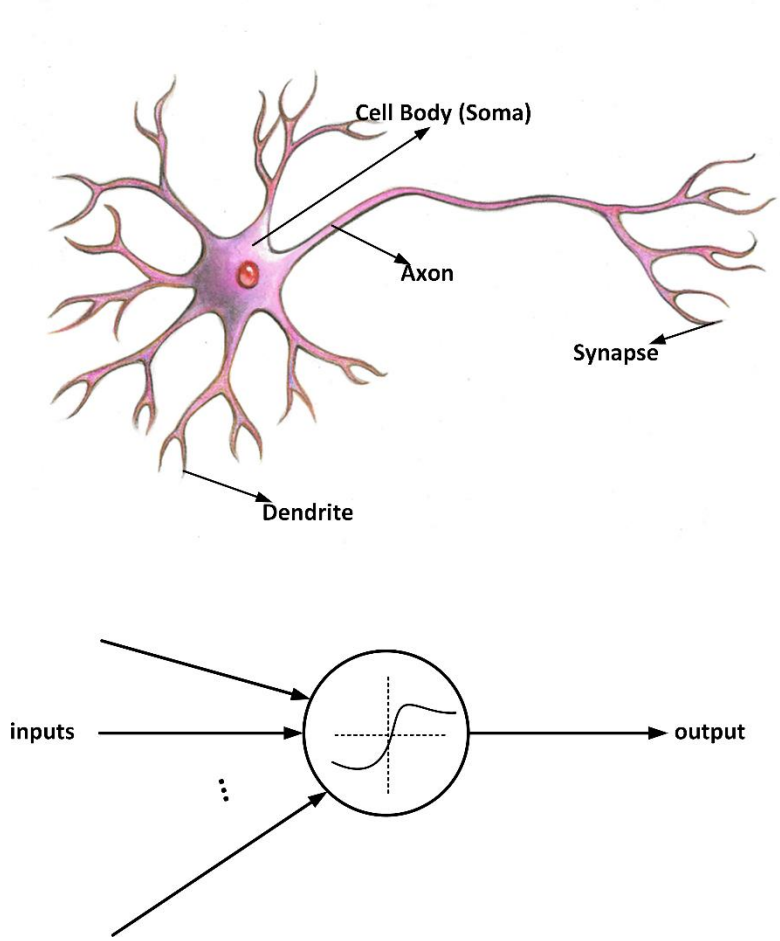

Figure 3: Diagram of biological neuron (Illustrated by Kimberly Sowell, permission obtained) above diagram of artificial neuron.

All artificial neural networks have **layers**, **weights**, **inputs**, **biases** (or thresholds), **activation functions** and some connection mechanism. But this is not enough to effectively train a neural network. When training a neural network we calculate an error (or loss). There are many ways to calculate the loss, but the simplest way is the difference between the predicted (learned) and the true outputs.

Training algorithms work by updating the weights and measuring the way that the loss changes over time. This is usually accomplished by some form of **gradient descent** optimizer. It's possible to get "stuck" or quickly converge to a non-optimal solution when strictly following the steepest gradients, and because of this, we typically use some form of stochastic gradient descent, which is simply a stochastic (or slightly randomized) form of gradient descent.

Back propagation is used to allow us to "propagate" the error information from the last layer to the first layer to modify the weights, and is often used in a way that is synonymous with training. Methods that can be used to improve a networks results are called "**learning rules**".

1.5 A Renaissance for Neural Networks?

Although these methods have been successfully used for decades, they have seen a very recent resurgence as refinements upon existing techniques, combined with newer hardware, and the growth of "Big Data", have created an AI boom that shows no signs of slowing down.

The global market for smart machines is expected to grow to +15 billion by 2019, with an average annual growth rate of nearly 20% [8]. The set of techniques that have led to this growth has been coined “deep learning”.

In the following sections we will discuss some of the techniques that enable deep learning, why they work, and how they are used to make smart machines.

2. Convolutional Neural Networks

In order store and process analog signals (for example image, voice and biological signals), on a computer, one must first convert these inputs into a digital form in a process called Analog to Digital conversion (A2D). This transforms the information from a continuous space to a discrete space, losing some information in the process. Often that information isn’t critical to understanding the underlying analog signal but in some instances we could lose critical data – e.g. high frequency information.

In digital processing we often refer to a “piece” of data, such as a picture, as a *sample*. It is convenient, but computationally expensive, to represent these samples in a high dimensional space where each unit (or pixel in the case of images), is considered as being located on a specific axis with the range of possible values (for example 0-255 in the case of an 8 bit color-channel in an image’) being the size of that axis.

An image that is 100x100 would be represented as a vector that is a point in 10,000 dimensional space. We call this space the “feature space”.

Since even the most powerful computers can have trouble with such high dimensional space, we try to reduce the number of dimensions to just those that are critical to a task or to change the way these features are represented.

In the traditional pattern recognition approach, to perform a given task we separate our process into two steps, **Feature generation** and **Feature selection**. Feature generation generates new features from the pixel space. Feature selection reduces the dimensionality of the feature space.

Examples of feature generation include morphological, fourier and wavelet transforms which create more useful features for specific tasks and feature selection includes methods like Principal Component Analysis (PCA) and Linear Fisher Discriminant (LDA) [9].

There is a newer technique based on sparse mapping where instead of trying to reduce the dimensionally, we expand it with the goal of representing more abstract features. This is inspired by models of the visual cortex of animals [10]. Convolutional layers, a key component of deep learning, make use of this sparse mapping approach.

2.1 Convolution and its Role in a Neural Network:

One of the most novel and useful aspects of convolutional neural networks is that they can learn the filters that previously had to be custom designed by the researcher (a task that would often take years of trial & error). These convolutional layers are essential to this task in modern deep neural networks.

Convolutional layers make use of the convolution operator. The Convolution operator is used on two functions. One is the signal from the sample space and the other, called the filter, is applied to the sample. On a GPU, convolutions are implemented as matrix multiplications.

This operator has a long history in image processing applications and dates back to the time when digital image processing started. The Convolution operation can be discussed in both spatial and transform space. In the spatial space the convolution operator is the equivalent operation of correlation with the reversed filter, i.e., this operator calculates the similarity of the input function with the filter. For example, the edge detection and corner detection filters are using the similarity of the input image with a pre-defined filter mimicking the edge or corner shape. Looking at the function in the transform space the convolution is performing frequency filtering. For example, low pass or high pass filters have their equivalent spatial filters which could be applied to the image using convolution operations.

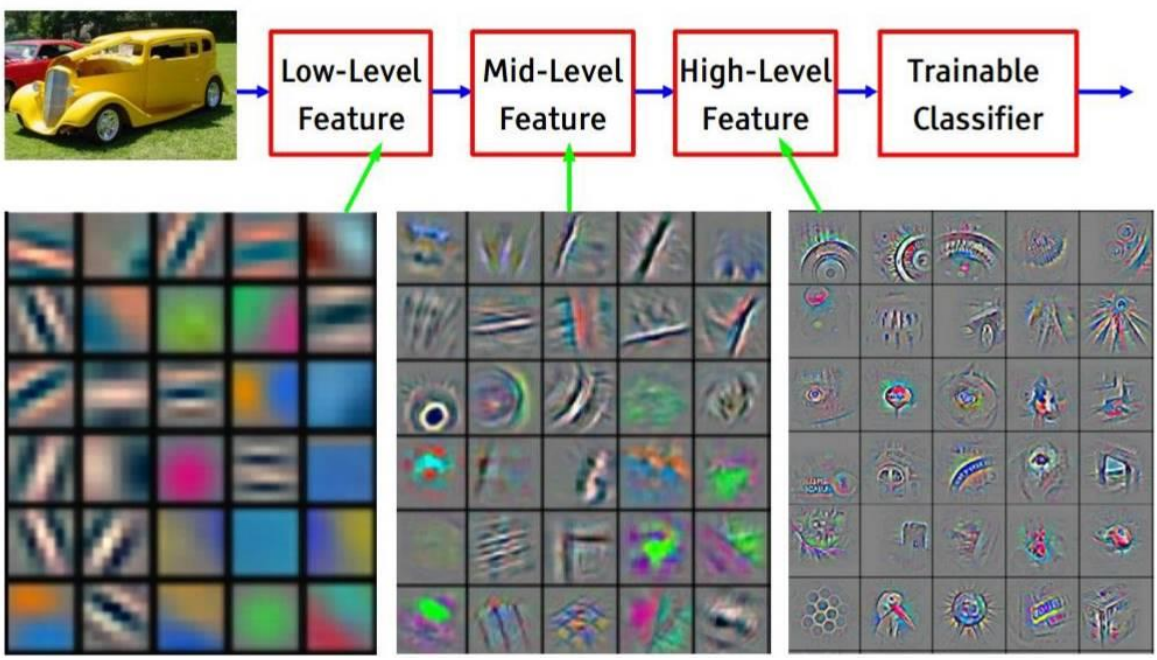

Figure 4 Visualization of the learned convolutional filters at different layers. Copyright Movidius, used with permission.

In the deep learning approach, the filter is learned and applied to the data during the training process with the hope that after training, the learned filter will be the best choice for the task. One difference between the way Convolutions are used in CNN's from more traditional uses is that the convolution operator is applied using a 4 dimensional filter. This is essentially a set of 3D filters that are stacked in the fourth dimension. We use the 4 dimensional filter to map a 3d space to another 3d space. See figure 5

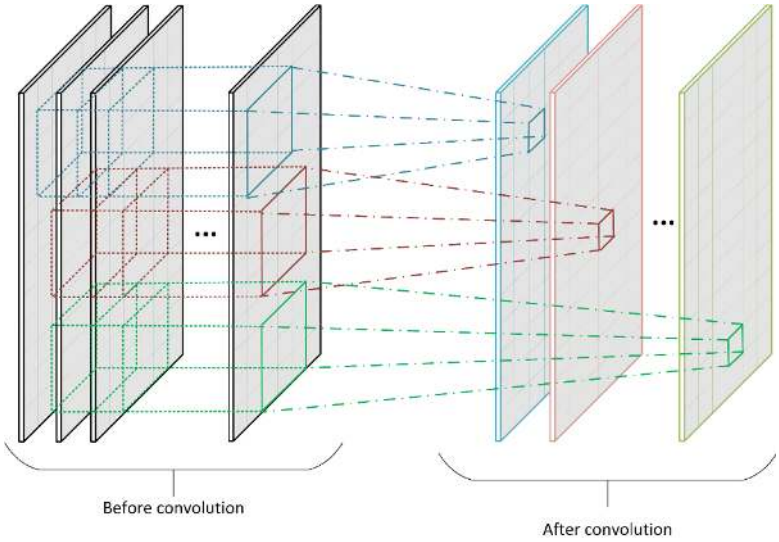

Figure 5. A 4 dimensional filter maps 3d space to another 3d space using convolutions.

Convolutional layer: in general for an n dimensional signal, the convolutional layer is an *n* or *n+1* dimensional feature space mapping with *n+1* or *n+2* dimensional kernels (filters). For example, given a 2 dimensional image, the convolutional layer would be 3 dimensional with a 4 dimensional kernel. In this case the 4 dimensions of the kernel are correspond to 1. width and 2. height of the input, 3. the number of channels of the input and 4. the number of channels of the output. In figure 5 you can see two different convolutional layers. The k channel layer on the left side is mapped to a p channel layer on the right side using a 4D kernel. This kernel is shown using different 3D kernels with different colors.

Pooling layer: pooling is an operation which accepts a pool of data values as input, and generates one value from them to be passed to the next layer. For example this operation could be mean or maximum of the input values. There are two important purposes for pooling operations. One is reducing the size of the data space to reduce overfitting and the other is transition invariance. A pooling layer is performing the pooling operation on its inputs. Figure 6 shows how a pooling operation is applied to a one channel input to reduce the dimensionality of the dataspace. In this figure we have a 3x3 pooling operation applied to a 12x6 one channel feature space. The most used pooling operation is max-pooling wherein the maximum value of the features in the pool is selected to be mapped to the next layer.

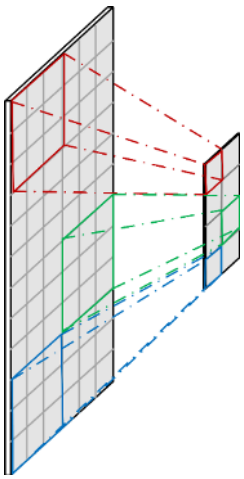

Figure 6.A pooling operation is reducing the size of the feature space.

The importance of this layer is described in the next example. In generic deep neural networks (described in 2.2.1 Generic Deep Neural Networks:)a dense, fully connected layer emerges from the convolutional layers. For example, consider a convolutional layer with 10 channels and a 100x100 feature space. Placing a fully connected layer after it would result in computing 100000 weights from this layer to each neuron in the dense layer which requires significant memory and computation resources. Using a pooling layer helps to reduce resource demands. Additionally, without a pooling layer, a network this big might suffer from over-fitting, especially if there is not enough representative data in the training set. Pooling also helps to provide transition invariance by helping each kernel to cover more space.

Nonlinearities: Each neuron in the deep neural network model is taking advantage of a non-linear activation function to calculate the output value. This would lead to one of the most advantageous properties of the deep neural networks which is their ability to describe highly non-linear systems. Being highly non-linear helps the model to be suitable for real-life problems and gives solutions in pattern recognition that cannot be achieved through more classical methods. In the early days of the neural networks the *tanh* and *sigmoid* activation functions were most popular. Realizing that most of the data tends to be concentrated around zero, newer techniques such as rectified linear units (RELU) and Exponential linear units (ELU) [11] become popular because they are nonlinear near zero. They also benefit from an infinite range.

2.2 Deep Neural Networks

The power of deep learning first made worldwide news in 2011, when a deep learning algorithm achieved better than human visual pattern recognition in an international competition. The accuracy was 6 times better than the nearest non neural network approach and twice as accurate as human experts [12].

In this section we discuss four of the better-known deep learning architectures that have made the most impact in recent research.

2.2.1 Generic Deep Neural Networks:

are usually made of one or more convolutional layers, wherein each convolutional layer is usually accompanied by a pooling (max-pooling) operation. One can also use bigger strides in the convolution layer in order to reduce the data dimensionality (the stride of a convolution operation is the number of the pixels the kernel window is sliding before calculating the convolution in each location).

In generic deep neural networks, the convolutional layers are usually followed by one or more dense fully connected layers. See figure 7. The rectified linear unit is the most common activation function used in these kind of networks. The last layer is typically taking advantage of other nonlinearities based on the task.

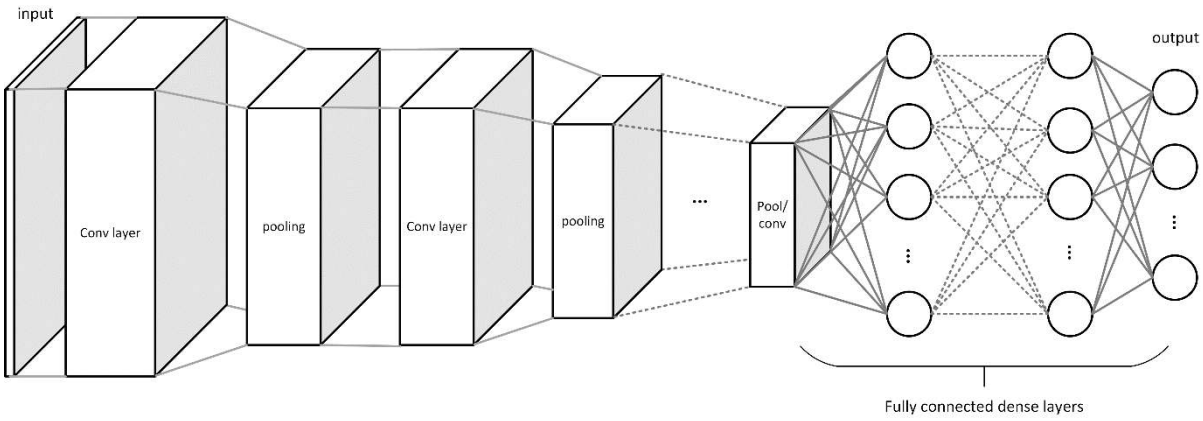

Figure 7: A generic deep neural network. Convolutional and pooling layers followed by fully conneceted dense layers.

2.2.2 Fully Convolutional Networks (FCN):

These are deep neural networks where all the layers are convolutional, pooling, and Un-pooling layers wherein the pooling and un-pooling layers are usually placed between two convolutional layers. They are similar to typical deep neural networks except they have no dense layer.

The output of a fully convolutional network is as the same type as its input. For example if the input is a k channel image the output of the network could be a p channel image but not something else entirely.

Un-pooling layers are designed to perform the inverse operation of pooling layers by increasing the size of the feature space. These layers operate on a single pixel and expand that pixel into a pool of data. There are different implementations of un-pooling layers. The most popular of which are repeating values and sparse un-pooling in DNN designs. The repeating value technique is expanding the data from one value to a pool of data with the same value (figure 8). With sparse un-pooling, the values of the original data are mapped to a larger space in sparse form. Figure 9 illustrates 2x2 sparse un-pooling. There are different methods for choosing the location where the value is mapped in sparse un-pooling. For example in [13] the indices have been memorized while applying the pooling layer and in the un-pooling layer those indices are used to map the values.

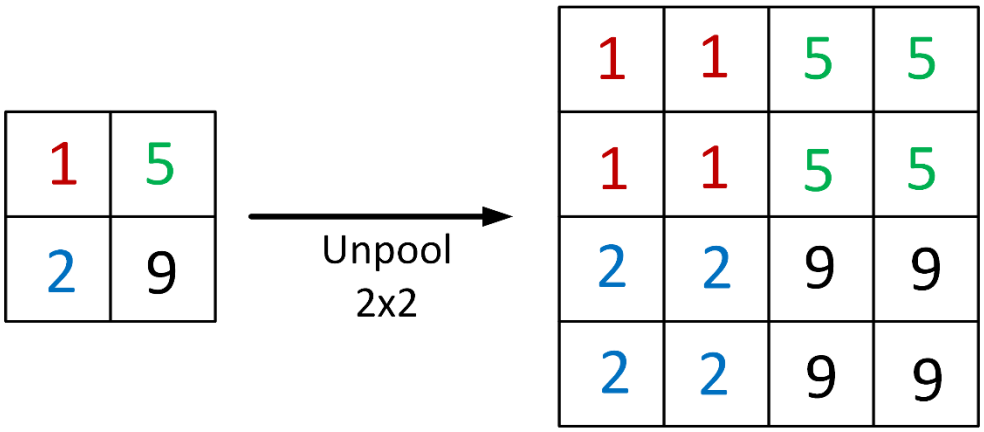

Figure 8: a 2x2 un-pooling operation with repeating values

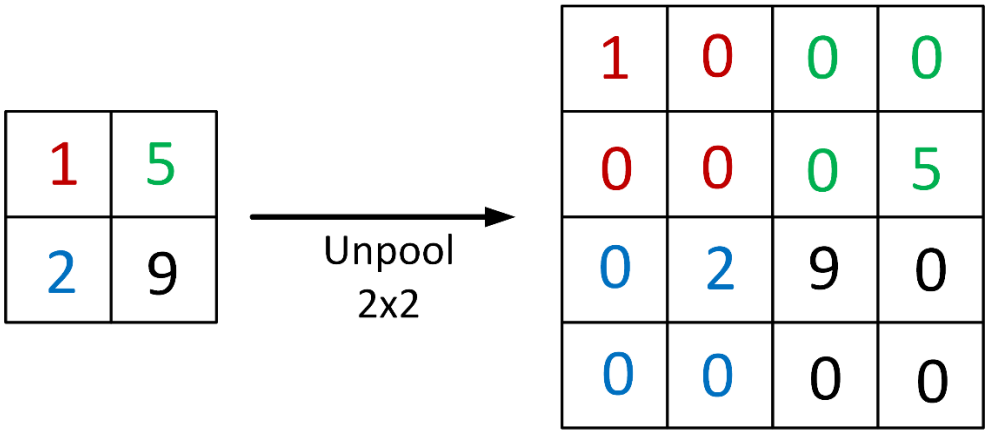

Figure 9: a 2x2 sparse un-pooling operation

2.2.2 Auto-Encoders:

Auto-Encoders are a design of a deep neural network wherein the input and output data are from the same class and also have the same data structure. For example if the input of the network is a 3 channel 128x128 image the output of the auto-encoder is also a 3 channel image with the size 128x128.

The auto-encoder network could be a fully convolutional network or it can have one or several fully connected layers in the middle of the network which is usually known as the bottleneck of the network. The idea with this kind of network is to create a compressed version of the input in the bottleneck. The data can then be reconstructed from the bottleneck. The part of the network that does the compression is called the encoder. The part that decompresses the data from the bottleneck is called the decoder.

The auto-encoder refers to the merged structure of the encoder and decoder in a model. See figure 10.

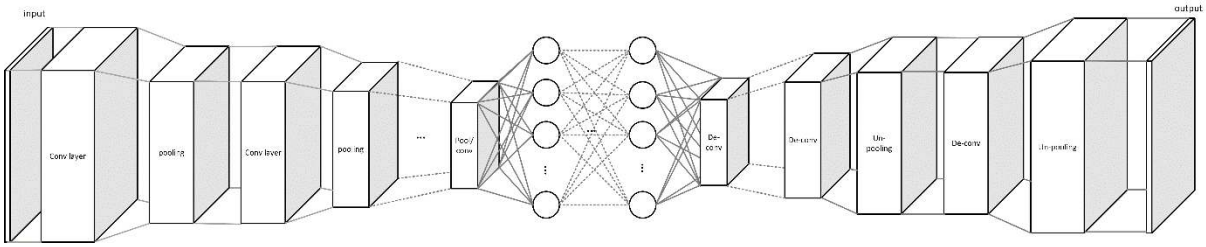

Figure 10: Auto encoder is the merged structure of the encoder and decoder in a model

2.2.4 Recurrent Neural Networks (RNN):

These networks are designed to operate on a sequence of the data as input, for example, a sequence of frames from a video. This can be considered as a system with memory that can remember the input at the previous stage and make decision for the present input based on the sequence of the data before.

The memory of an RNN networks is known as the hidden state of the network. The hidden state of the network is updated based on the current state. The input to the network and the output of the RNN is calculated based on the state of the network at each sequence. Recurrent Neural networks have had great success in speech and video scene recognition where they are often combined with convolutional layers.

3. State of Art Today

3.1 Main Enabling Technologies (overview)

GPU – how they impact training

Despite advancements in parallel pipelines, hyperthreading, and multiple cores, modern CPU's are optimized primarily for problems that are best solved sequentially. This is ideally suited for many common algorithms and tasks, but is not performant for tasks with inherent parallelism. GPU (graphics processing units) are designed for highly parallel graphical processes which can normally be reduced to a limited set of matrix or tensor operations. As GPUs became more and more powerful to cope with the increasing sophistication of 3D graphical models used in gaming, and in augmented and virtual reality, they became the preferred option for deep learning research. A typical GPU has hundreds or thousands of cores, and although each core is much slower than a typical CPU core, together they are able to train networks (especially deep neural networks) at the level of 1 or 2 orders of magnitude faster than on a CPU. This makes intuitive sense. Human neurons are significantly slower than GPU or CPU cores, but our brains are able to perform many recognition and classification tasks faster and with more accuracy than most computer models. This is because human brains have billions of individual, asynchronous neurons, and with highly parallel analog electro-chemical communication.

GPU vs CPU training benchmarks.

For some tasks, a GPU can be on the order of 100X faster while using less power and at a much lower cost. Generally, any task which requires the same operation to be performed thousands of times (as in matrix multiplication) will greatly benefit from the use of a GPU [14].

GPU models and setups:

NVIDIA's latest consumer series is the Pascal series, labelled with a 10 prefix. This series represents a significant improvement in both performance and power consumption over previous models.

	Titan X Pascal	1080	1070	1060
CUDA cores	3584	2560	2048	1280
Memory Capacity	12 GDDR5X	8 GB GDDR5X	8 GB GDDR5	Up to 6 GB GDDR5
Memory Speed	10 Gbps	10 Gbps	8 Gbps	8 Gbps

(source: [15], [16])

Multiple NVIDIA GPUs can be combined with the use of an SLI bridge, but currently, generic drivers only exist for 2 GPU SLI for the 10 series. NVIDIA also sells non consumer GPU's that are specialized for deep learning that do not have this driver limitation.

When choosing a GPU setup for deep learning, it is important to choose a device that has enough memory to fit your model and enough cores to efficiently solve the problem. One also needs to decide on multi GPU vs single GPU setups. It is also important that enough system memory exists to easily copy the model between GPU ram and system ram with room to spare for the operating system and any running applications and services.

Deep Learning on a stick

Many companies are deploying hardware customized for deep learning, often targeted at deployment in smart objects. There are significant speed and power consumption improvements that can be realized in hardware implementations of DL operations. Movidius, for example, released the "fathom neural compute stick" which uses approximately 12.5 times less power than an equivalent NVIDIA GPU in a small USB compatible device [17]. It is targeted at robots, drones and surveillance cameras and can drastically improve recognition accuracy. It is a good choice for "off the shelf" deep learning hardware when one is ready to deploy a model.

Figure 7: Movidus fathom USB stick. Copyright Movidius, used with permission.

CUDNN/CUDA

NVIDIA is the most popular consumer GPU manufacturer with Deep Learning researchers. Although their products are targeted primarily at gaming and 3D graphics processing, they also invest significant resources in deep learning research. NVIDIA provides CUDNN, a low level library of deep learning primitives that run on top of CUDA [17].

Developing and evaluating Deep Learning models.

When deploying or designing a deep neural network, it is useful to try out several variations with many parameters. Although speed and power consumption are critical to end products, people often design their networks using slightly slower but higher level frameworks. This allows us to test out ideas and "prove" products before creating highly optimized implementations in hardware or software. Although deep learning can be implemented in nearly any programing language, most resources and community support available is intended for python. Although most deep learning tools are now compatible with python 3X, the deep learning community still primarily uses python

2.7.

The most popular high level frameworks for this are Caffe, Theano, and Tensorflow.

Caffe [18] **–** Created at the Berkely Vision and Learning center. Models are defined in a configuration file instead of being coded directly. They claim to have the fastest convolutional neural network implementation available.

Theano [19]**–** Initially designed for fast, stable symbolic operations (including symbolic differentiation). It dynamically generates optimized C code which can be transparently executed on the GPU up to 140 times faster than an equivalent CPU **implementation.**

Tensorflow [20]– The newest of the three, tensorflow was designed by Google and is currently experiencing the fastest growth in usage. As of this writing, tensorflow performs slightly slower than the other two on benchmarks but is easier to deploy on multiple GPUs.

There are a number of frameworks built on top of these platforms, such as Lasagne and Keras that allow further abstraction and thus ease and speed of development. Keras is a good choice, especially for beginners since it can use either Tensorflow and or Theano as a back end and it provides a simpler model for development.

3.2 Example applications

Recently, Google provided a mobile Tensorflow API which allows developers to deploy deep learning models directly to smart phones [21]. Google uses it for "Translate's instant visual translation" and recommend it for any application where processing needs to be done on the device

FotoNation (www.fotonation.com), a company whose software runs in over 2.7 Billion smartphones worldwide, has released a new Face Recognition product based on the latest deep learning techniques to achieve rapid and accurate face recognition.

Nuance uses deep learning for their "Dragon Naturally Speaking" line of voice recognition applications [22]. Deep learning is used in both initial product development and later for custom refinement on consumer electronics.

Prisma, with 27 million users, is a popular cell phone app that uses deep learning to transform photos into paintings in the style of famous artists [23].

Soundhound, inc is collaborating with NVIDIA using deep learning to create "to add a smart, voice-enabled, conversational interface to every technology that humans interact with" based on their platform called "Houndafiy" [24].

At the WWDC 2016 keynote, Apple's John Gruber revealed that they use deep learning in their photo app, first in their own datacenter where they pre-train a model based on a vast base of labeled photos, and later on the iPhone to annotate images as they are taken. They also analyze all photos on a phone when the device is plugged in and not in use so as to avoid draining battery power. Google takes a different approach, avoiding "on device" classification, and instead processing users' photos with their cloud infrastructure [25].

Neurala released an API that uses deep learning to perform real time object tracking, recognition, and other tasks. As of September 2016 they are the largest supplier of deep learning software for Consumer Drones. Their deep learning software allows drones to operate autonomously, enabling

tasks such as “follow this person” or “find this car”. Good performance on such tasks were the realm of sci-fi only a few years ago [26].

4. Thoughts & Conclusions

Today deep learning is being used in our cell phones, in our cars, in tablets and computers. It has pushed the boundaries of what is possible for tasks such as Image segmentation [13], object detection [27], face recognition [28], voice analyzing [29], emotion detection [30] and gender recognition [31].

Why has Deep Learning suddenly catalyzed research across so many fields? Well it is a combination of many factors: the recent emergence of highly affordable high-density, GPU-based computational hardware has provided the engines to process very large datasets and implement the advanced training methodologies required to develop accurate CNNs; the widespread availability of GPUs in today’s devices, coupled with cloud-based data processing services provides the means to apply these CNN architectures to everyday applications such as voice or image processing. Big data provides the fuel to drive research activity and refine results to the point where Deep Learning solutions typically outperform even the best of human-designed pattern recognition tools.

Today we are at a point where many new problems can be tackled through the use of Deep Learning techniques – many of these are long-standing problems such as voice recognition which was never quite good enough to make its way out of the laboratory and into everyday use. But this year we have seen several launches of ‘smart speakers’ that can control your home – and behind these new devices lies a web of deep learning technologies – both analyzing your needs, translating your voice requests and marshalling the necessary logistics to deliver everything to your doorstep or onto your TV set.

To get started with solving your own problem you’ll need a state-of-art GPU – the same technology going into the latest gaming PCs – and most of the core software is freely available on the Internet. There are several software packages trending in the Deep Learning field, including but not limited to Theano (on python), Lasagne (on Theano), Tensorflow (on python and C++), Caffe (on python and MATLAB), and MatConvNet (on MATLAB).

And the good news is that all of these tools are relatively inexpensive and readily available!

Finally you’ll need a large dataset – for your particular application we’ll assume you have your own specialized data sources, but there are many public sources of suitable data. And from here its is deep oceans of learning ahoy!

References

[1] B. Yegnanarayana, *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[2] S. Lawrence, C. L. Giles, and A. C. Tsoi, “What Size Neural Network Gives Optimal Generalization ? Convergence Properties of Backpropagation,” *Networks*, no. UMIACS-TR-96-22 and CS-TR-3617, pp. 1–37, 1996.

[3] Y. Chauvin, “Generalization performance of overtrained back-propagation networks,” in *Neural Networks*, Springer, 1990, pp. 45–55.

[4] S. Theodoridis and K. Koutroumbas, “Pattern recognition and neural networks,” in *Machine Learning and Its Applications*, Springer, 2001, pp. 169–195.

[5] P. Lewis, “Analogy between human and artificial neural nets.” [Online]. Available: http://users.ecs.soton.ac.uk/phl/ctit/nn/node2.html.

[6] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943.

[7] F. Rosenblatt, “The Perceptron - A Perceiving and Recognizing Automaton,” *Report 85, Cornell Aeronautical Laboratory*. pp. 460–1, 1957.

[8] A. McWilliams, “Smart machines: Technologies and global markets,” *BCC Res.*, no. May, 2014.

[9] G. Chandrashekar and F. Sahin, “A survey on feature selection methods,” *Comput. Electr. Eng.*, vol. 40, no. 1, pp. 16–28, 2014.

[10] “Convolutional Neural Networks (LeNet).” [Online]. Available: http://deeplearning.net/tutorial/lenet.html.

[11] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),” *CoRR*, vol. abs/1511.0, 2015.

[12] J. Schmidhuber, “Who Invented Backpropagation?,” 2014. [Online]. Available: http://people.idsia.ch/~juergen/who-invented-backpropagation.html.

[13] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: {A} Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *CoRR*, vol. abs/1511.0, 2015.

[14] K. Krewell, “What’s the Difference Between a CPU and a GPU,” 2009. [Online]. Available: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/.

[15] “GEFORCE GTX 10-SERIES NOTEBOOKS.” [Online]. Available: http://www.geforce.com/hardware/10series/notebook.

[16] “NVIDIA TitanX Graphics Card with Pascal.” [Online]. Available: http://www.geforce.com/hardware/10series/titan-x-pascal.

[17] “NVIDIA cuDNN.” [Online]. Available: https://developer.nvidia.com/cudnn.

[18] “Caffe.” [Online]. Available: http://caffe.berkeleyvision.org/.

[19] “Theano.” [Online]. Available: http://deeplearning.net/software/theano/.

[20] “TensorFlow is an Open Source Software Library for Machine Intelligence.” [Online]. Available: https://www.tensorflow.org/.

[21] “Mobile TensorFlow.” [Online]. Available: https://www.tensorflow.org/mobile.html.

[22] N. Lenke, “Why we’re using Deep Learning for our Dragon speech recognition engine,” 2016. [Online]. Available: http://whatsnext.nuance.com/in-the-labs/dragon-deep-learning-speech-recognition/.

[23] “The Technology Behind the Viral Prisma Photo App,” 2016. [Online]. Available: https://news.developer.nvidia.com/the-technology-behind-the-viral-prisma-photo-app/.

[24] “SoundHound Inc. Collaborates With NVIDIA to Bring Deep Learning-Based Natural Language Understanding to Cars,” 2016. [Online]. Available: http://www.businesswire.com/news/home/20160105006247/en/SoundHound-Collaborates-NVIDIA-Bring-Deep-Learning-Based-Natural.

[25] “The Technology Behind Apple Photos And The Future Of Deep Learning And Privacy,” 2016. [Online]. Available: http://highscalability.com/blog/2016/6/20/the-technology-behind-apple-

photos-and-the-future-of-deep-le.html.

[26] “Neurala Becomes Largest Supplier od Deep Learning Software Running in Real Time on a Drone,” 2016. [Online]. Available: http://www.neurala.com/top-deep-learning-software/.

[27] C. Szegedy *et al.*, “Going Deeper with Convolutions,” 2014.

[28] Y. Sun, Y. Chen, X. Wang, and X. Tang, “Deep learning face representation by joint identification-verification,” in *Advances in Neural Information Processing Systems*, 2014, pp. 1988–1996.

[29] X.-L. Zhang and J. Wu, “Deep belief networks based voice activity detection,” *IEEE Trans. Audio. Speech. Lang. Processing*, vol. 21, no. 4, pp. 697–710, 2013.

[30] S. Bazrafkan, T. Nedelcu, P. Filipczuk, and P. Corcoran, “Deep Learning for Facial Expression Recognition: A step closer to a SmartPhone that Knows your Moods,” in *IEEE International Conference on Consumer Electronics (ICCE)*, 2017.

[31] J. Lemley, S. Abdul-Wahid, D. Banik, and R. Andonie, “Comparison of Recent Machine Learning Techniques for Gender Recognition from Facial Images,” in *Modern Artificial Intelligence and Cognitive Science (MAICS 2016)*, 2016.