Institute for Parallel and Distributed Systems
Machine Learning and Robotics Lab

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit No. xxxx

IMAGE RECOSNTRUCTION FROM COMPRESSIVE SENSING MEASUREMENTS USING DEEP LEARNING

Luis Manuel Bracamontes Hernandez

Course of Study: Computer Science M.Sc.

Examiner: Prof. Dr. rer. nat. Marc Toussaint

Supervisor: Supervisor

Commenced: Date

Completed: Date

CR-Classification: ???

Abstract

Compressed sensing (CS) is a novel signal processing theory stating that a signal can be fully recovered from a number of samples lower than the boundary specified by Nyquist–Shannon sampling theorem, as long as certain conditions are met. In compressed sensing the sampling and compression occur at the same time. While that allows to have signals sampled at lower rates, it creates the necessity to put more workload on the reconstruction side. Most algorithms that are used for recovering the original signal are called iterative, that is because they solve an optimization problem that is computationally expensive. Not only that, but in some cases the reconstruction does not have good quality. This thesis proposes a non-iterative method ~~based on a Deep Learning (DL) framework~~ to recover signals from CS samples in a faster way, while maintaining a reasonable quality. DL has already proved its potential in different image applications. ~~As a result,~~ this approach is tested using grayscale images and recent DL software packages. The results are compared against iterative methods in terms of the amount of time needed for full reconstruction, as well as the quality of the reconstructed image. The experiments showed both the effectiveness of this method for speeding up the recovery process ~~and also the quality was maintained at a considerable quality level.~~

while maintaining a similar quality level.

method

machine learning using DL in order to recover ...

1 Introduction

1.1 Motivation

Compressed sensing (CS) is a novel technique based on the discoveries done by Candes *et al.* [13] and Donoho [25]. They showed that as long as certain conditions are met, a signal can be fully recoverd from a small number of measurements. Due to this fact, CS is suitable for many Signal Processing (SP) applications and specifically image processing. An example of this are the publications made by Duearte *et al.* and Wakin *et al.* [26, 51]. In their work they explained their ideas for architecting an imaging sensor capable of sampling and compressing the signal at the same time. Such a sensor offers the advantage of taking less samples than those needed by concentional sensors and there is no extra compression fase after the measuremnts are taken, thereby saving computation worload. As a result, the industry and the research community have been interested and actively worked during the last years in order to design sensors capable of giving both better energy efficiency as well as good algorithms capable of yielding high quality for the recovered images. Unfortunately, everything comes with a cost and recovering the original image from such compressed measurents is a problem that has high complexity and computationally expensive to solve. This problems are commonly referred to as inverse problems since one tries to recover a signal from under-sampled measurements, which translates into trying to find a solution to an underdetermined system.

Many algorithms have been proposed for image reconstruction, among the most widely known and recognized one can find [43, 24, 41, 45, 18, 27]. Nevertheless, most of them suffer from the same disadavnatages because they follow the most general approach for the reconstruction process. First, they solve an optimization problem, which implies the use of iterations until a possible solution is obtained. Because of that, those algorithms are called iterative.Furthermore, they may need up to 10 minutes to reconstruct only one image. Second, they only focus on obtaining raw pixel values that, hopefully, will corespond to the original image. That makes such algorithms not useful for other common tasks like object detection and segmentation. Although those algorithms are not fast and thefore not suitable for real-time applciations they render reconstructed images with high quality and good visual

impact. Designing algorithms that can reconstruct images from compressed measurements in a simpler faster way while keeping or increasing the final quality of the image is still an active topic for research and one of the major motivations for this thesis.

Another justification for this thesis is the state-of-the-art results that Deep Neural Networks (DNNs) [40] have proved for several image processing tasks like classification [38], image denoising [8] and superresolution [23]. Based on the promising performance of those examples, applying DNNs for reconstructing images from compressed measurments seems to be another application that should be investigated. Namely, we focus on the use of Convolutional Neural Networks (CNNs) for the afore mentioned task and because of its nature the reconstruction process may take less time and the quality as well as the visual impact could also be preserved or even increased.

In this thesis a different method for image reconstruction that does not follow the conventional approach is proposed, that is we do not try to find a solution for an optimization problem. As a result, this is a non-iterative solution for recovering images from compressed measurements. In order to reconstruct the image we exploit the proven capacity of CNNs for image processing tasks. Several network architectures are evaluated and compared with iterative methods in terms of time and final quality of the reconstructed image.

1.2 Outline

The thesis follows this organization:

Chapter 2 - Theorical foundations introduces the important background in compressed sensing, deep learning and CNNs. It also presents the common problems that are faced when trying to recover images from compressed measurements using traditional algorithms.
Chapter 3 - Implementation methodology gives an overview of the datasets and tools that are used in order to build and implement the neural network. It also explains the data preparation and post processing that yields the final outcome.
Chapter 4 - Evaluation shows the comparison between state-of-the-art iterative algorithms for the recovery process against the proposed method using CNNs. The evaluation is made in terms of the speed and quality. It also explains the differences between several network architectures.
Chapter 5 - Conclusion states the lessons learned throughout the development of the thesis as well as the future work that could lead to better results and extend its application.

2 Theorical foundations

In this chapter we will explain the necessary background that is used throughout the thesis. In the following we will introduce the theoretical foundations of compressed sensing and some of the problems one faces when dealing with it. Deep Learning will also be introduced along with convolutional neural networks which will be discussed in more detail.

2.1 Compressed Sensing

Compressed sensing is mathematical theory that deals with the problem of recovering a signal from a small number of measurements, the number of measurements is less than the minimum number of samples defined by Shanon-Nyquist theorem (sampling acquisition must be done at least twice the highest frequency in the signal). For many applications, imaging and video, the sampling rate specified by Nyquist might end up being very large that the amount of samples that have to be compressed and transmitted increases the complexity of the system and makes it costly. Compressed sensing contradicts the previous statements since it claims that a signal may be recovered with lesser samples or measurements than conventional approaches. That is possible because it proposes a generalization of a linear mapping paired with optimization in order do the sampling and recovery process at notably inferior rates than that imposed by Nyquist rate.

Compressed sensing heavily relies on two principles: sparsity of the signal, and incoherence, which refers to sampling/sensing representation; both terms will be further discussed. In addition to that, CS tries to overcome two of the major incapabilities of sample-compress schemes: First, the number of samples or measurements is considerably reduced. Second, the compression stage occurs inside the sensor (hardware), therefore, there is no need to add extra encoding computation.

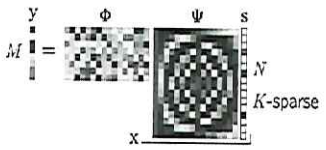

Figure 2.1 – Measurement process in compressed sensing.

2.1.1 Sparsity of a signal

The mathematical formulation of sparsity is defined as follows: a signal $\mathbf{x} \in \mathbf{R^N}$ (for instance n-pixels of an image) is interpreted in terms of its basis representation as a linear combination of the orthonormal basis $\{\psi\}_{i=1}^{N}$ and coefficients α as

$$x = \sum_{i=1}^{N} \alpha_i \psi_i \qquad or \qquad \mathbf{x} = \Psi\alpha \tag{2.1}$$

CS takes advantage of the certainty that plentiful natural signals are sparse or compressible when stated in a condensed representation. For example, images are easily compressed using the discrete cosine transform (DCT) and wavelet bases [42]. Namely, a signal is said to be compressible or k-sparse if there exists a convenient basis ψ for which $\mathbf{x}$ is a linear combination of only K basis vectors, obeying $K \ll N$. That means, only K elements of α present in 2.1 are nonzero.

2.1.2 Incoherence

Measurements in CS are obtained by using a linear operator that takes $M < N$ inner products between $\mathbf{x}$ and a set of vectors $\{\phi\}_{i=1}^{M}$. Figure 2.1 depicts the operation. Putting everything together, each measurement y_i is an $M \times 1$ vector and $\{\phi\}_i^T$ representing the rows as a $M \times N$ matrix Φ; then the sampling process is

$$y = \Phi x = \Phi\Psi\alpha \tag{2.2}$$

From that one can see that the product of matrices $\Phi\Psi$ has size $M \times N$, and the measurement matrix Φ is independent from the signal $\mathbf{x}$. The previous is important since the choice of the sensing matrix plays an important role for the reconstruction process, that is recovering $\mathbf{x}$ from measurements $\mathbf{y}$). In particular, Φ and Ψ should be incoherent. Coherence of two matrices is a measure that asserts the level of correlation of Φ and Ψ and is computed as follows:

$$\mu(\Phi, \Psi) = \max_{k,j} | < \phi_k, \psi_j > | \tag{2.3}$$

the lower μ the more incoherent the matrices are and therefore it is more probable the successful reconstruction of the original signal.

2.1.3 Sensing Matrix and RIP

The sensing matrix Φ should be chosen so that the number M of its rows is bigger than the number of nonzeros entries in the sparse signal K, that is $M \geq K$. Due to the fact that defining a number K for natural signals is unknown, a constraint commonly referred as *restricted isometry property* (RIP) [15, 14, 10] was proposed. It ensures that the matrix Φ retains the length of the k-sparse vectors and therefore the signal is not corrupted by the transformation going from $\mathbf{x} \in R^N$ to $\mathbf{y} \in R^M$. The mathematical representation of RIP reads

$$(1-\delta_k)\|x\|_{l_2}^2 \leq \|\Phi x\|_{l_2}^2 \leq (1+\delta_k)\|x\|_{l_2}^2 \tag{2.4}$$

where δ_k, referred to as *restricted isometry constant*, is the smallest number preserving the inequality for the matrix Φ.

Designing optimal sensing matrices goes beyond the scope of this thesis and because the RIP and incoherence may be obtained with high probability by taking Φ as a random Gaussian matrix with independent and identically (iid) distributed elements[12], we will not devote more time for this topic. Particularly, we will use an iid Gaussian sensing matrix throughout the thesis unless otherwise specified.

2.1.4 Signal Recovery

Even though RIP (2.4) and incoherence theoretically ensure that a K-sparse signal might be entirely described with only M measurements, it is still needed to restore the original signal $\mathbf{x}$. A great deal of algorithms alredy existing accomplish the recontruction process by reading the measurements $\mathbf{x}$, the matrix Φ and solving an optimization problem. Concretely, most recovery algorithms try to find the best approximation $\hat{x} = \Psi x$ for some transform basis Ψ. It has been proved [16, 25] that the optimal solution for that problem is getting $\hat{x}$ with the smallest l_0 norm from measurements y given by

$$\hat{x} = arg\min_x \|\Psi x\|_0 \qquad s.t. \quad y = \Phi x \tag{2.5}$$

Nevertheless, the solution for 2.5 is an NP-hard that becomes numerically unstable and requires comprehensive computation. As a result, a convex relaxation was introduced and instead the l_1 norm is used, reducing the problem to a linear program of the form

$$\hat{x} = arg\min_x \|\Psi x\|_1 \qquad s.t. \quad y = \Phi x \tag{2.6}$$

This holds true because measurements $\mathbf{y}$ were taken using an iid Gausian sensing matrix and therefore a sucessful reconstruction is highly probable to occur [25, 11]. Algorithms trying

to solve this problem are dubbed iterative given the nature of convex optimization. Among state-of-the-art iterative algorithms one can find [24, 41, 43], and they are used as a baseline to compare our results.

In this thesis we do not aim to find an iterative solution for the aforementioned optimization problem in equation 2.6, instead we explore the promising capabilities of using deep learning for image processing tasks.

2.2 Deep Learning

Deep learning is a modern field of machine learning that makes used of deep neural networks in order to learn representations of data with several layers of abstraction. It has been an active research topic during recent years because they have achieved state-of-the-art results in image processing, video, speech and audio applications. It also solves the inability of traditional machine-learning methods to process plain raw data. Moreover, there is no need to have extensive experience in order to engineer features that transform data into useful representations or feature vectors that will ultimately allow the learning algorithm detect or classify the data.

Deep-learning techniques, whether supervised or unsupervised, learn multiple level representations. Such representations are realized by simple but non-linear entities transforming the representation level by level into a higher and more abstract one. By stacking a sufficient number of such layers and by implementing a learning process, highly complex functions can be learned and accurately approximated. The main singularity of such layers is that the features are learned without much human intervention, that is clever engineering from a person with very specialized knowledge. Rather, those features are learned directly from the data.

Because deep learning has proved to be effective in overcoming difficulties that have caused trouble in the artificial intelligence community for years and because more and more data is becoming available, it is highly expectated that deep learning will achieve more accomplishments in the near future [40].

2.2.1 Supervised learning

Supervised machine learning algorithms are trained using data composed of a group of inputs $\mathbf{x}$ and an analogous label $\mathbf{y}$. The main reason for this learning approach is to produce the most accurate mapping $f : \mathbf{x} \rightarrow \mathbf{y}$ that is used to reproduce more labels. Examples for this kind of task are classifying hand-written digits. Normally, we have a set of given hand-written numbers paired with the correct discrete class and we would like to train a model so that in the future we receive more hand-written digits and hopefully they will be correctly classified. On the other hand, we have a set of compressed measurements and its respective original

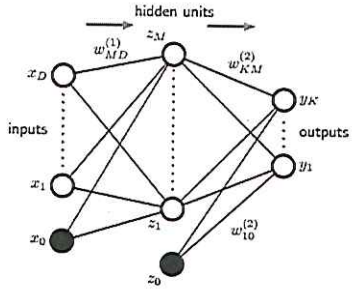

Figure 2.2 – A neural network takes x_D inputs and gives y_K outputs. Another important term is the number of hidden units which refers to the number of layers between the input and output layers. The term deep comes when the number of layers ~~and its size~~ grows larger. Most commonly $\omega^{(l)}_{MD}$ ~~is represented as the~~ are called weight parameters

images. In this case, we have a regression problem and our model will be trained so that new compressed measurements are fed into it and successful images are reconstructed.

2.2.2 Unsupervised learning

Unlike supervised learning unsupervised learning is not given an explicit label **y**. That means, the learning algorithm only has access to the input **x** as the training data. The applications of unsupervised learning are clustering, that is finding similarities on the data for a particular number of groups, density estimation, which aims to discover a distribution explaining the input data. Finally, it is used to reduce the dimensionality of the input into a lower one that retains as much information as possible.

In our case, we use the unsupervised approach to both learn a sensing matrix Φ and reconstruct the image back to its original size from ~~those~~ those measurement.

2.2.3 Neural Networks

Neural networks (NN's), in the field of machine learning, are a mathematical attempt to approximate and estimate functions in the same way that many biological systems do. Even though its origins dates back to the 40's and 60's, they have found effective and active usage during recent years. There are three major concepts concerning NN's that one has to deal with to make use of them: network architecture, activation function and network training.

- **Network architecture** describes the number of parameters that build the network, the number of hidden layers and input and output size. Figure 2.2 shows a node-like

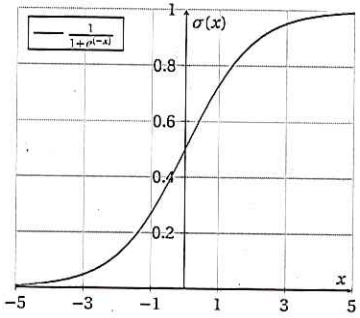

Figure 2.3 – Sigmoid function plot.

generalization of a fully connected network architecture. The green arrows show how the information flows. Outputs y_K are parameterized as a weighted sum of a predefined non-linear function f, ω and a bias b

$$y_K = f(\sum_{i=1}^{D} \omega_i x_i + b) \tag{2.7}$$

Normally, b is embedded into ω as an extra dimension to simplify its training and the representation becomes

$$y_K = f(\sum_{i=1}^{D} \omega_i x_i) \;\; with \;\; \omega \in \mathbb{R}^{d+1} \tag{2.8}$$

- **Activation function** specifies the output of each neuron in the network according to a given input. Such input, is the product between ω and x which is subsequently transformed by a function commonly referred to as nonlinearity because of its behavior. Most popular functions are plotted in Figures sigmoid2.3, tanh 2.4 and more recently linear rectifier (ReLU) 2.5 [32].
- **Network training** is the process for learning the values of parameters ω of the network. The simplest way to solve the problem is by minimizing an error function, for example least squares error. That is achieved from a set of input vectors x_n, for $n = 1, ..., N$, along with target vectors y_n. Using optimization theory we try to minimize

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} \| f(x_n, w) - y_n \|^2 \tag{2.9}$$

thereby obtaining the optimum values for ω. The actual minimization step uses of an

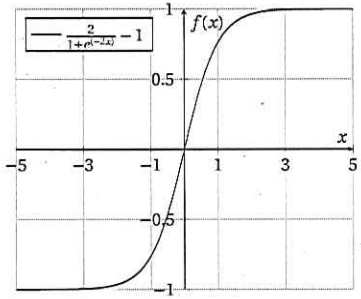

Figure 2.4 – Tanh function plot.

algorithm called backpropagation, which is a method that computes the gradients by recursively applying chain rule. The error is computed by doing a forward pass in the network.

- **Chain rule** is a mathematical formula allowing to compute the gradient of a function with respect to another function. It is of importance for NN's because it helps find the derivative of the error between input and the output. Figure 2.6 shows how the gradient is computed through the network.
- **Forward pass** is the process of taking the desire input and transforming it according to the current network parameters. The output is then compared with a desired target to get the error. Figure 2.7 provides a graphical representation.
- **Backpropagation** is a technique that permits NN's learn function approximations. Once the forward pass is carried out, the error is obtained ~~using as the evaluation metric~~ the loss function. Afterwards, the gradient of the loss function with respect to the given input is recursively propagated through the network. The weights ω are updated by moving ~~in the~~ towards the minimum of the loss function. Figure 2.8 depicts the process. The process of completing a forwards pass, error computation, loss gradient and backpropagation over all training examples is called epoch.
- **Stochastic gradient descent (SGD)** is used to find the minimum of a the loss function by iterating over each input example. Because of that, if the training examples are very large only one iteration would take a long time which would make the process slow. As a result, SGD is not scalable and limits its application to small datasets. Algorithm 1 shows a generalization of how it is implemented. Another practical difficulty is to find a good learning rate α. As a result, many algorithms try to adapt the learning on the go, to avoid overshooting and allow for faster convergence.
- **Mini-batch Stochastic gradient descent (SGD)** is the most commonly used algorithm

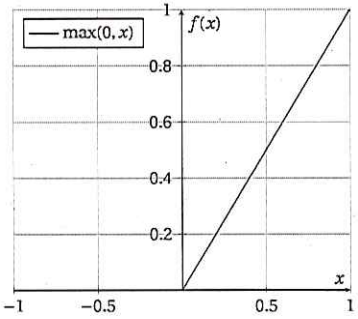

Figure 2.5 – ReLu function plot.

for large neural networks because it overcomes the limitation of SGD for big datasets. Unlike ~~SGD~~, it starts moving towards the minimum by just evaluating a small number of samples and updating the parameters accordingly. That is, not all training examples are used before ω values are changed. It introduces a new hyperparameter for the training specifiying the number m of samples to be used during each iteration and it is referred to as batch size.

2.2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN's/ConvNets) are a subset of the previously described neural networks. They are also built of neurons that aim to learn weight parameters ω and bias parameters b by performing dot products and normally followed by one of the afore mention nonlinear functions. The main difference, however, is that CNN's assume that the input are images that are convolved through the network during the forward pass and the number of parameters is lesser because the dot products are only computed in certain regions and not in the image as a whole.

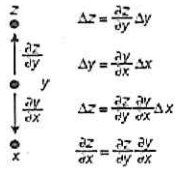

Figure 2.6 – Chain rule computation in the network.

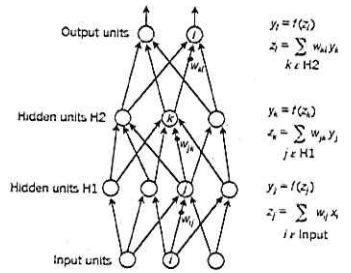

Figure 2.7 – Forward pass is used to compute the predictions using the current values of weight parameters.

CNN architecture

In contrast with fully connected networks, the layers of CNN's are arranged in a 3D fashion seen as: width, height, depth. It differentiates in the sense that only a small subset of a layer connects to the one before it, thus reducing the number of neurons and encapsulating certain features in the input image as shown in figure 2.9.

CNN's are assembled by interconnecting three different types of layers one after another: Convolutional Layer, Pooling Layer and Fully-Connected Layer. Optionally, the output of each layer is passed through the ReLu function 2.5.

- **Convolutional layer** is the main part composing a CNN. It computes the dot product between parameters ω and a small area of the input image. Its functionality and output size are controlled by four hyperparameters: number of kernels or receptive fields K,

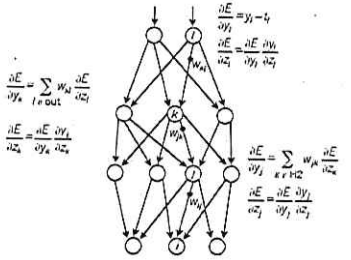

Figure 2.8 – After the error has been computed, it is backpropagated through the network updating ω parameters accordingly.

Algorithm 1 Stochastic Gradient Descen (SDG)

Require: Learning rate α
Require: $L(\omega)$: Loss function
Require: ω_0: Initial parameter values
1: $t \leftarrow 0$ (Initialize timestep)
2: **while** ω_t not converged **do**
3: $t \leftarrow t+1$
4: $g_t \leftarrow \nabla_\omega L_t(\omega_{t-1})$ (Gradient of loss function)
5: $\omega_t \leftarrow \omega_{t-1} - \alpha g_t$ (Update parameters)
6: **end while**
7: **return** ω_t (Resulting parameters)

size of kernel filters F, number of zero padding P and stride S. Figure 2.10 shows the mechanism that CNN's use, each channel of the input image is convolved with the filters generating an output.

In general convolution layers have the following characteristics:

- Input size: $W_1 \times H_1 \times D_1$
- Hyperparameters: Number of filters K, filter size F, stride S and zero padding P.
- Output size: $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$

Sometimes it is necessary to preserve the original sizes of height and width, like in our case, a common setting to achieve that is by using these settings: $K = 3$, $S = 3$ and $P = 3$.

- **Pooling layer** commonly used for classification purposes. Its main function is to reduce the spatial size, doing that allows to reduce the complexity of the network while maintaining information. In our experiments we do not make use of it.

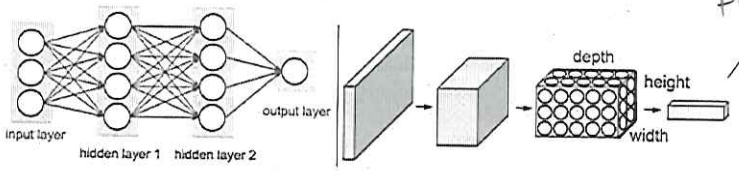

Figure 2.9 – **Left:** Fully connected network with 2 hidden layers. **Right** 3D (width, height, depth) arrangement of neurons. Each layer of the CNN carries out a 3D transformation on each channel (RGB in case of images).

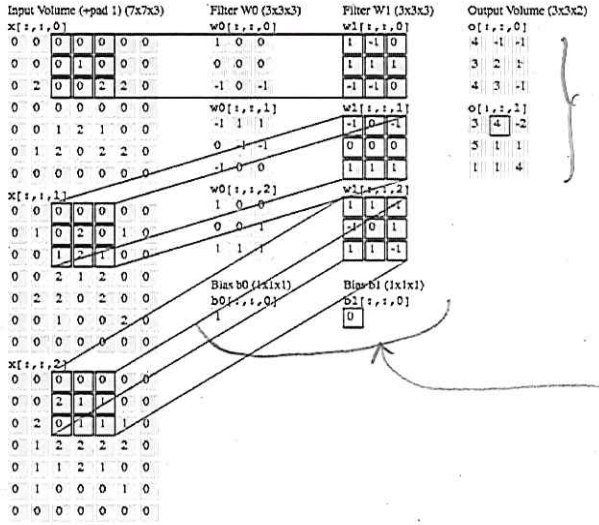

Figure 2.10 – Example of a convolution layer in a 3D input with parameters Initial input volume 3×3, $K = 2$, $F = 3$, $S = 2$ and $P = 0$

$D_2 = 2$ $F_H = 3, F_W = 3$ (kernels are of size 3×3, there are $D_1 \times D_2$ $3 \times 2 = 6$ kernels, output is 2 maps of size $W_2 = \frac{7 - 3 + 0 + 2}{2} = 3$

$D_1 = 3$
$W_1 = 7$
$H_1 = 7$

$H_2 = \frac{7 - 3 + 0 + 2}{2} = 3$

- **Fully connected layer** is completely connected to all neurons in the previous layer, as in a regular neural network. It also makes networks grow lager and when used in CNN's they are commonly the last layer computing the final result for a type of classification. As with pooling layers we do not make use of it.

- **Activation function for CNN's** normally refers to a linear rectifier function ReLu 2.5. It is explicitly given as another layer in the network and allows for efficient gradient propagation. It is very popular because not only is its computation easier than other activation functions like sigmoid 2.3 or tahnh 2.4 but, it also decreases the probability for observing vanishing gradients.

(NOT READ YET)

3 Implementation methodology

As stated before we want to evaluate the performance of CNN's to reconstruct images from compressed measurements. In this chapter we will describe the tools and software frameworks used in order to build the set-up for our experiments. First, we introduce some of the most widely used libraries to build and train neural networks and the specific software tools for this thesis. Second, we briefly describe Graphics Processing Units (GPU's) and give reasons why they are an important tool when working with deep learning. Third, we will describe the datasets for training the networks and briefly justify its utilization. Then, we explain how the data is pre and post-processed. This is important since it allows to efficiently use the hardware resources and makes the training process numerically stable. Finally, we propose CNN architectures for the reconstruction process.

3.1 Theano

Theano is a library written in python that permits to define, optimize and compute mathematical expressions that deal with high-dimensional arrays in an efficient way. It is widly used among the deep learning community because it is optmized to make use of GPU routines that make training faster, efficient symbolic differentiation, stability optimizations and therefore making it realiable since it is constantly tested and debugged [49]. Nevertheless, Theano is not intendet to be used only for neural natwork applications. Therefore, it is necessary to write routines or API's that specially handle the difficulties encountered for deep learning. Such applications are normally an extra abstraction layer that sits on top of Theano's implementation and specially allow to build and train neural networks. Examples publically avaible are Lasagne [22], Blocks [50] and Keras [19]. Other common frameworks in deep learning are Google's Tensorflow [4] that is written in C++, Torch [20] written in Lua and Caffe [36] also written in C++. Most of them offer the same characteristics and mostly differ in the language they are written in, speed and target application, for example Caffe is not suitable for audio and text applciations.

Figure 3.1 – NVIDIA graphics card GeForce GTX Titan X used in our experiments.

3.1.1 Sdeepy

Due to the fact that the previously mentioned implementations are constantly updated and changing, Sony decided to develop its own library to develop its projects. Sdeepy is an in-house Sony's deep learning library implementation that makes used of Theano and incorporates routines commonly used as building-blocks for training and testing neural networks. It has the advantage that many new features can be added at any time they become available while maintaining the stability. Furthermore, it is easier to adapt and modify according to one's needs. Because of all that, we will use Sdeepy for this thesis but, all implementations might be easily translated into any available framework.

3.2 Graphics Procesing Unit (GPU)

A Graphics Processing Unit (GPU) is a chip architecture mainly design for imaging and gaming. It differs from general-purpose CPU's in that they can handle larger amounts of data efficiently and faster in a parallel way, making them much more suitable for deep learning applications. It has extended its usage into machine learning area because companies like NVIDIA have develop GPU's specially optimized for neural networks. Not only that, but they have also written routines (NVIDIA Deep Learning SDK) that improve and speed up common operations like convolutions, better memory management and friendlier API's for easy learning. By distributing the workload of the training process the convergence happens much faster than by using conventional CPU's. See Figure 3.1 and Table 3.1 .

Table 3.1 – Technical details of the GPU

Architecture	Maxwell
CUDA cores	3072
Base clock speed	1000MHz
Boost clock speed	1075MHz
Memory type	GDDR5
Memory	12GB
Memory Bandwidth	7Gbps
Memory interface width	384-bit