Institute for Parallel and Distributed Systems
Machine Learning and Robotics Lab

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis Nr. MCS-0010

IMAGE RECOSNTRUCTION FROM COMPRESSIVE SENSING MEASUREMENTS USING DEEP LEARNING

Luis Manuel Bracamontes Hernandez

Course of Study: M.Sc. Computer Science

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

Advisor 1: MSc. Javier Alonso Garcia
Advisor 2: Ph.D. Fabien Cardinaux

Commenced: 1. April 2016

Completed: 30. September 2016

CR-Classification: C.1.3, I.2.6, I.4.5

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 machine learning method using Deep Learning (DL) in order 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 while maintaining a good quality level.

Contents

List of Figures

List of Tables

List of Algorithms

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, 52]. 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 conventional sensors and there is no extra compression phase after the measurements are taken, thereby saving computation workload. 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 measurements is a problem that has high complexity and computationally expensive to solve. These 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 under-determined 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 disadvantages 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 correspond 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 therefore not suitable for real-time applications, 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 measurements 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 of the images 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 - Theoretical 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. We also describe the CNN's we have devised in order to recover images from compressed measurements.
Chapter 4 - Evaluation shows the reconstructed images using our CNN and explains the training procedure. We also make a 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 Theoretical 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 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, like 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 to 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.

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 representaion. 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 $M <$ inner products, where $M < N$, 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

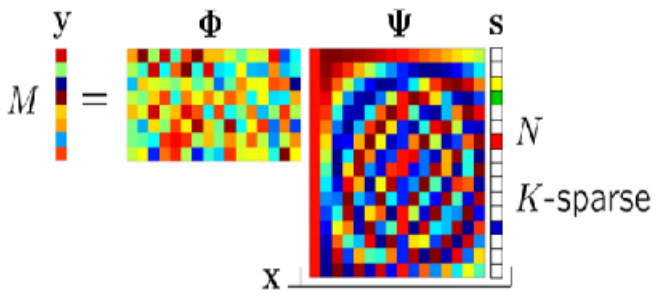


Figure 2.1 – Measurement process in compressed sensing.

a measure that asserts the level of correlation of Φ and Ψ and is computed as follows

$$\mu(\Phi,\Psi) = \sqrt{n} \max_{1\leq k,j\leq n} | < \phi_k, \psi_j > | \tag{2.3}$$

where n is the number of elements of the signal and $\mu(\Phi,\Psi) \in [1,\sqrt{n}]$. The lower μ the more incoherent the matrices are and therefore the successful reconstruction of the original signal is more probable.

2.1.3 Sensing Matrix and RIP

The sensing matrix Φ should be chosen so that the number M of its rows is larger 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. Moreover, since 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 a constant 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 can be entirely described with only M measurements, it is still necessary 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 $\hat{x} = \Psi\alpha$ for some transform basis Ψ. It has been proved [16, 25] that the optimal solution for that problem is $\hat{x}$ with the smallest l_0 norm from measurements y given by

$$\min_{\hat{x}\in R^N} \|\hat{x}\|_0 \qquad s.t. \quad y = \Phi\hat{x} \tag{2.5}$$

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

form

$$\min_{\hat{x} \in R^N} \|\hat{x}\|_1 \qquad s.t. \quad y = \Phi\hat{x} \tag{2.6}$$

This holds true because measurements **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, either supervised or unsupervised, learn multiple level representations. Such representation 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 without 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 **x** and an analogous label **y**. The main reason for this learning approach is to produce the

most accurate mapping $f : \mathbf{x} \mapsto \mathbf{y}$ 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 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, unsupervised learning 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 these measurements.

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

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

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

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

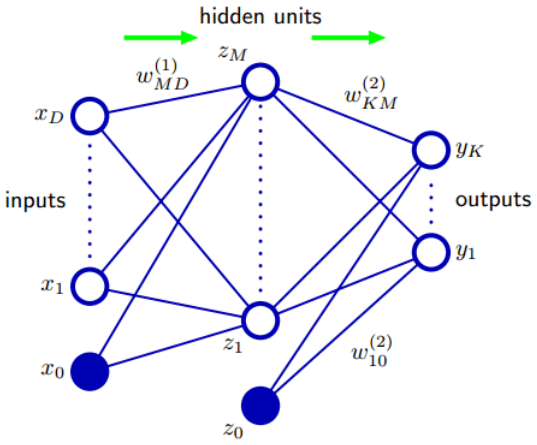


Figure 2.2 – A neural network takes x_D inputs and gives y_K outputs. Another important term is the number of hidden units which refers to the number of layers between the input and output layers. The term deep comes when the number of layers and its size grows larger. Most commonly $\omega^{(i)}_{MD}$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

- **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 sigmoid 2.3, 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

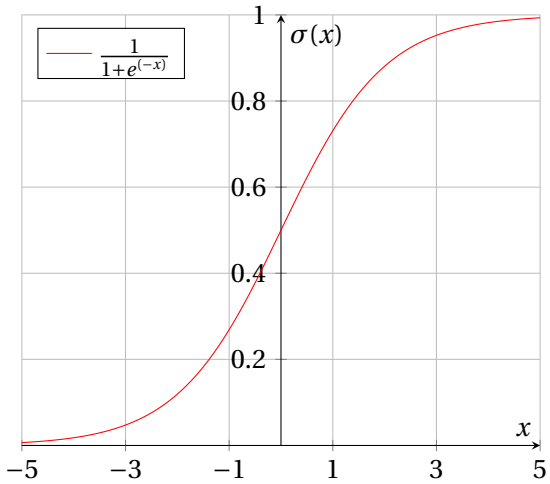


Figure 2.3 – Sigmoid function plot.

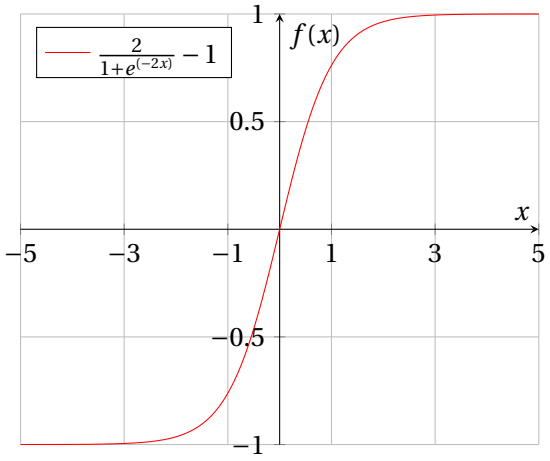


Figure 2.4 – Tanh function plot.

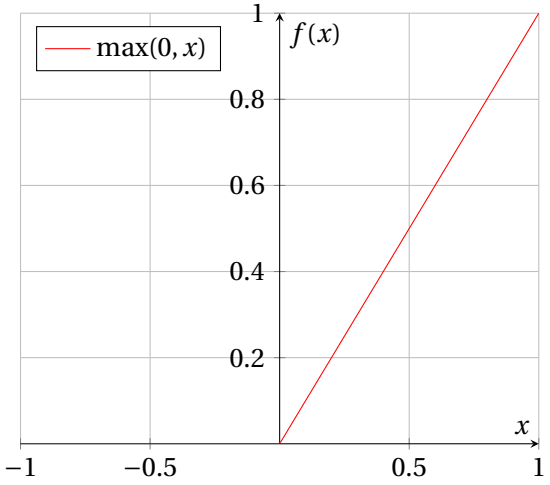


Figure 2.5 – ReLu function plot.

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 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.

- **Weight Initialiyation** in order to carry out the training process parameter values must have an initial value. Such value is, normally, assigned randomly and does not affect the final outcome.

- **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 desired input and transforming it according to the current network parameters. The output is then compared with a desired target to obtain the error. Figure 2.7 provides a graphical representation.
- **Backpropagation** is a technique that permits NN's to learn function approximations. Once the forward pass is carried out, the error is obtained evaluating 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 towards the minimum of the loss function. Figure 2.8 depicts the process. The action of completing a forwards pass, error computation, loss gradient and backpropagation over all training examples is called epoch.
- **Gradient Descent (GD)** is the numeric algorithm used to find the minimum of the loss function. The gradient of the loss function is evaluated using the training examples as input, then it is multiplied by a parameter called learning rate and substracted from

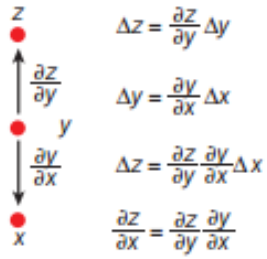


Figure 2.6 – Chain rule computation in the network.

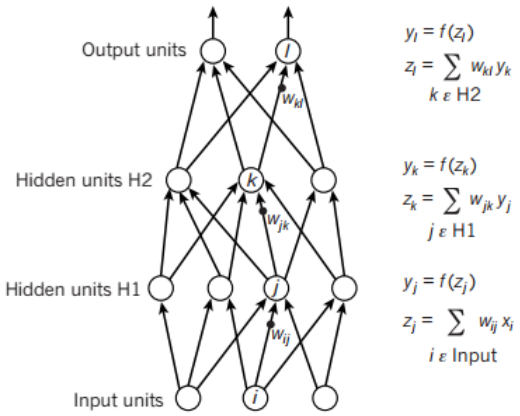


Figure 2.7 – Forward pass is used to compute the predictions using the current values of weight parameters.

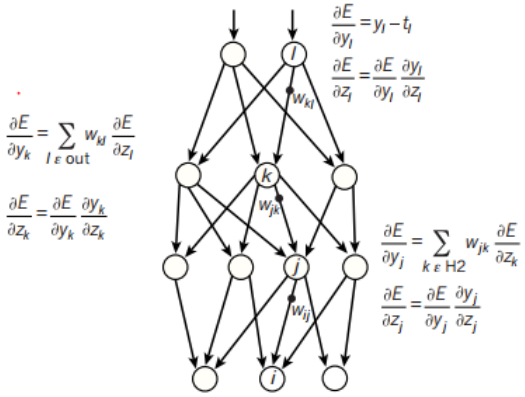


Figure 2.8 – After the error has been computed, it is backpropagated through the network updating ω parameters accordingly.

the current weight values. In practice, it is not implemented because it first needs to compute the gradient with respect to each training sample before doing a simple update for the weight parameters. For large datasets this approach would require agreat deal of computational effort and its speed convergence is slow. As a result, GD is not scalable and limits its application to small datasets. The process is decribed in Algorithm 1.

Algorithm 1 Gradient Descen (GD)

Require: Initial $\omega_0 \in R^n$, stepsize α, function $\nabla_\omega L_t(\omega)$, training samples M
1: $t \leftarrow 0$ (Initialize timestep)
2: **while** ω_t not converged **do**
3: $t \leftarrow t+1$
4: $g_t \leftarrow \frac{1}{M}\sum_{m=1}^{M} \nabla_\omega L_t(\omega_{t-1}^m)$ (Gradient of loss function for each sample)
5: $\omega_t \leftarrow \omega_{t-1} - \alpha g_t$ (Update parameters)
6: **end while**
7: **return** ω_t (Resulting parameters)

- **Stochastic gradient descent (SGD)** is a variation of GD that makes updates in a different way. Unlike GD, SGD makes an update of weight parameters for each sample making more efficient. That is, it moves towards the minimum with each gradient evaluation instead of doing a weighted average of gradients. Nevertheless, it requires a certain number of iteration, normally unknown, and since it makes updates with each sample it is very sensitive if the data has been scaled or transformed making it inaccurate. Algorithm 2 shows a generalization of how it is implemented.

- **Mini-batch Stochastic gradient descent (SGD)** is the most commonly used algorithm for large neural networks because it overcomes the limitations of GD and SGD for big datasets. Unlike GD, it starts moving towards the minimum by just evaluating a small number of samples and updating the weight parameters accordingly. That is, not all training examples are used before ω values are changed. Dissimilar to SGD, it does

Algorithm 2 Stochastic Gradient Descen (SGD)

Require: Initial $\omega_0 \in R^n$, stepsize α, function $\nabla_\omega L_t(\omega)$, training samples M

$t \leftarrow 0$ (Initialize timestep)
while ω_t not converged **do**
  $t \leftarrow t+1$
  for $m = 1$ `to` M **do**
    $g_t \leftarrow \alpha \nabla_\omega L_t(\omega_{t-1}^m)$ (Gradient of loss function for each sample)
    $\omega_t \leftarrow \omega_{t-1} - \alpha g_t$ (Update parameters)
  end for
end while
return ω_t (Resulting parameters)

not update the parameters with each training sample, rather it evaluates a smaller subset of training samples thereby increasing accuracy. That means, it introduces a new hyperparameter for the training specifying the number β of samples to be used during each iteration and it is referred to as batch size. Algorithm 3 shows this approach.
A practical difficulty, common to all algorithms following gradient descent approach, is to find a good learning rate α. Since, there is no proven optimum value in literature many algorithms are implemented trying to adapt the learning rate on the go to avoid overshooting and allow for faster convergence.

Algorithm 3 Mini Batch Stochastic Gradient Descent (SGD)

Require: Initial $\omega_0 \in R^n$, stepsize α, function $\nabla_\omega L_t(\omega)$, training samples M, match size β

$t \leftarrow 0$ (Initialize timestep)
while ω_t not converged **do**
  $t \leftarrow t+1$
  for $m = 1$ `to` $\frac{M}{\beta}$ **do**
    $g_t \leftarrow \sum_{i=1}^{\beta} \nabla_\omega L_t(\omega_{t-1}^i)$ (Gradient of loss function for each sample)
    $\omega_t \leftarrow \omega_{t-1} - \alpha g_t$ (Update parameters)
  end for
end while
return ω_t (Resulting parameters)

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 on certain regions of the input data and normally followed by one of the afore mention nonllinear functions. The main difference, however, is that CNN's assume that the input data exposes high correlation in at least one dimension, for example images, and exploits that nature. The input data is 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.

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

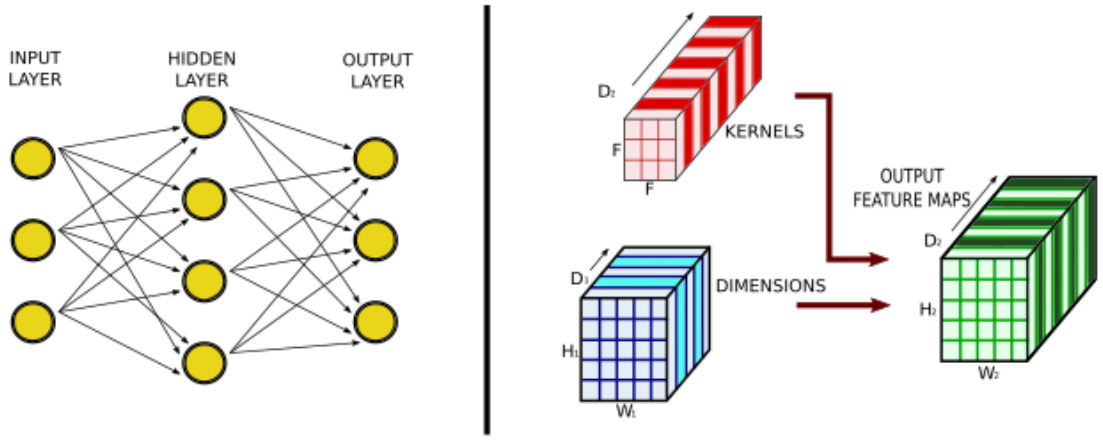


Figure 2.9 – **Left:** Fully connected network with one hidden layer. **Right** 3D (width, height, depth) CNN implementation. Each input dimension or channel is convolved with all kernels and each element in the kernel represent a neuron.

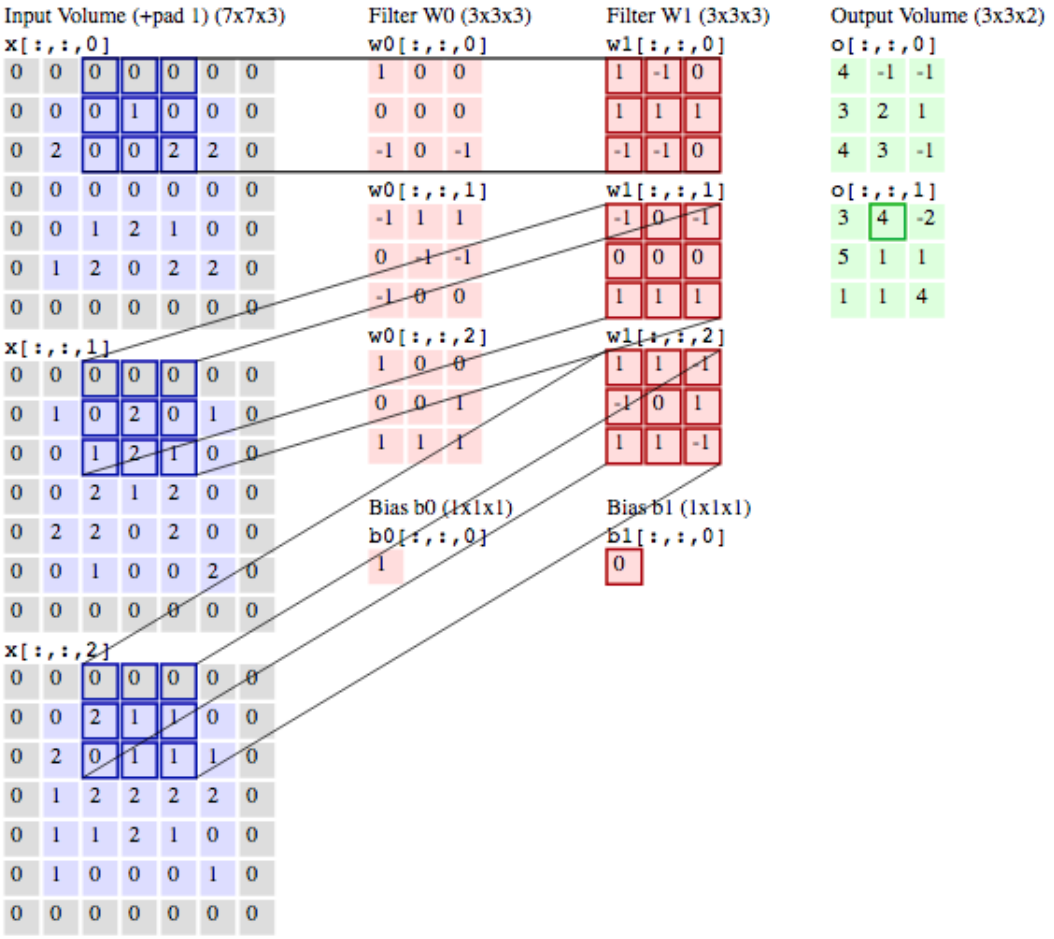


Figure 2.10 – Example of a convolution layer in a 3D input with parameters: initial input volume 5×5, $K = 2$, $F = 3$, $S = 2$ and $P = 1$.

  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$

  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** 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.

- **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.

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 widely 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 intended 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 [51] 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.

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 their projects. Sdeepy is an in-house Sony's deep learning library implementation that makes use 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 designed 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 occurs much faster than by using conventional CPU's. Table 3.1 lists the technical characteristics of the GPU used in our experiments.

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

3.3 Datasets

It is also important to mention that along with better hardware and more ingenious algorithms, deep learning has achieved major success because in recent years more and more data is becoming available through the internet. For most of machine learning methods data is the most important asset to obtain successful results. In fact, choosing the right dataset is the first

step in the deep learning work-flow cycle and it is, in general, more important than the chosen learning algorithm itself.
Since our goal is to reconstruct images, we make use of three different image datasets: BioID [29], Label faces in the wild (LFW) [33] and LabelMe [48]. The first two datasets served as an initial baseline for measuring the plausibility of our approach and the last one allows for a more robust justification as explained later. All of the datasets are known in the community for being applied to different computer vision tasks like face detection, emotion recognition and image classification. Here, we propose its usage for image reconstruction from compressed samples.

Table 3.2 – Original features of each dataset.

Dataset name	Number of images	training	testing	Original image size	Grayscale
BioID	1521	1371	150	384x286	Yes
LFW	13233	11933	1300	250x250	No
LabelMe	50000	40000	10000	256x256	No

All datasets differ in terms of image size, color model (RGB or Grayscale) and file format (jpg, png, bmp, etc.). In order to facilitate implementation, all datasets had to be transformed into a more generic form that could meet our constraints. Namely, it is more convenient to work with grayscale images and once promising results are obtained, it can be easily extended to RGB images or even video. For that reason, LFW and LabelMe datasets were converted into grayscale images using rgb2gray MATLAB function. Furtheremore, having an image size that was divisible by 16 was an initial requirement and so each image was resized to $256x256$ using imresize. The actual data being used is described in table 3.3.

Table 3.3 – Transformed features of each dataset.

Dataset name	Number of images	training	validation	Original image size	Grayscale
BioID	1521	1371	150	256x256	Yes
LFW	13233	11933	1300	256x256	Yes
LabelMe	50000	40000	10000	256x256	Yes

Finally, in order to evaluate the performance of our trained network we introduce a set of 18 images. The images, we believe, are the most popular ones in the image processing community and serve as a baseline for many reconstruction algorithms and image processing tasks. They also help compare our results against the ones reported by other researchers.

3.3.1 Training set

Consist of the largest set of images and is the one used for learning the optimum values of ω parameters during training. Because one can argue that fitting ω values to a particular dataset may lead to overfitting and therefore the network could perform poorly with unseen data we have to test our model with different data. Column training in table 3.3 refers to the number of training images for each dataset.

3.3.2 Validation set

Is a smaller part of the dataset that is not used for training. Since, the training images also come from the same distribution, correlations about the performance in in this data help decide whether the CNN is overfitting the data or how good the generalization is. Column validation in table 3.3 gives the number of validation images for each dataset.

3.3.3 Testing set

This a set of images we introduce in order to further test the performance of our CNN. It's not used or linked in anyway to the training process and serves more as a proof of the CNN's performance and overfitting avoidance. We divide the set from the tradition of using certain famous images for showing results of an image processing algorithm. The testing set is shown in figure 3.1

Figure 3.1 – From Top Left to Bottom Right: 'fingerprint', 'couple', 'boats', 'flintstones', 'peppers', 'woman', 'butterfly', 'bridge', 'houses', 'house', 'mandril', 'parrot', 'montage', 'foreman', 'man', 'barbara', 'lena', 'cameraman'

3.4 Preprocessing of the images for supervised learning

Even though grayscale images can be easily interpreted as a 2D matrix, CNN's expect as an input a 3D tensor, chapter 2.2.2, complying with the convention depth, height and width. Consequently, extra preprocessing is carried out to generate the data set that is used as the final input of the network for training. Moreover, we make use of a technique called Block Compressed Sensing (BCS). BCS allows to divide the image into small blocks of size $B \times B$ and then compress them by using the same sensing matrix Φ. It has the main advantage of having

a small sensing matrix that can be easily extended to process high-dimensional images and making the overall process faster [30, 28]. Another important parameter is the compression rate C. That is, how much information we would like to sense. For most of our experiments we choose $C = 16$ (we also show results with different C values).

To demonstrate the process, consider an image randomly taken from one of the datasets. It has a size of $M \times N$. First, it is converted into blocks of size $B \times B$, where $B = 16$, by using MATLAB command im2col. Second, each block is multiplied with the sensing matrix Φ to obtain the measurements. That is, blocks of size $16 \times 16 = 256$ samples, with $C = 16$ we only take $\frac{256}{16} = 16$ measurements per block. blocks are 16×16, we take thus $16 \times 16 \times 16$ measurements in total for an image of size $M = 256 \times N = 256$. A pictorial representation of the process can be seen in Figure 3.2. Φ is the sensing matrix of size $C \times (pixels\ per\ block) = C \times B^2$ and each block has $B \times B pixels$. Then, the blocks are vectorized so that they have a shape $B^2 \times 1$ and consequently multiplied with $\Phi(C \times B^2)$ yielding a final measurement shape $C \times 1$. Finally, after preprocessing each image individually, the data used for training and testing is a 4D tensor and each dimension is interpreted as follows: $(Number\ of\ images,\ size\ of\ C,\ size\ of\ \frac{M}{B},\ size\ of\ \frac{N}{B}$.

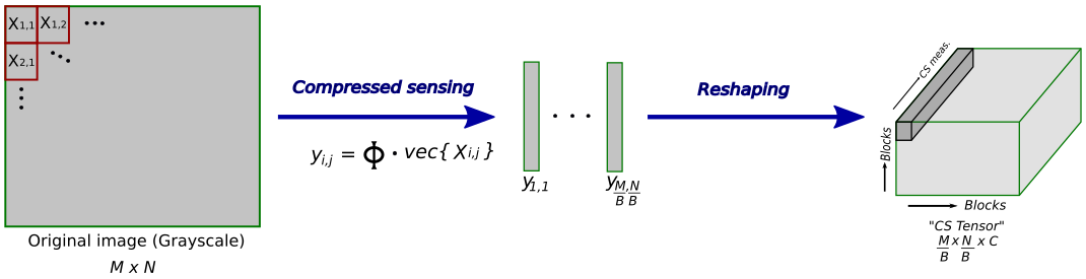


Figure 3.2 – Example of image preprocessing with $B = 16$, $M = 16$ and $N = 256$.

3.5 Preprocessing of the images for unsupervised learning

The preprocessing of the data for unsupervised learning is very similar to the approach explained in section 3.4. The main difference is that there is no compressing phase. That is, we directly reshape the image from $M \times N$ to $M \times \frac{N}{B} \times \frac{N}{B}$. In our case, the values are still the same as with the previous section $M = 256$, $N = 256$ and $B = 16$. This step is necessary because we would like to test if learning the sensing matrix Φ can produce better results than using the approach of compressing samples using i.i.d. Gaussian matrices.

3.6 Network architectures for CS recovery

The architecture of the network refers to the order in which different layers are arranged and interconnected to achieve a specific purpose. Throughout the development of the thesis

different architectures were heuristically tested and in the following we present the ones that produced better performance.

3.6.1 Small supervised CNN network

This the first network we propose to recover images from compressed samples is in Figure 3.3. It is composed of an input layer and an output layer. The input layer receives measurements of size $C \times \frac{M}{B} \times \frac{N}{B}$, with a number of filters $K = 128$ and followed by ReLu. The output layer receives as input the output of the previous layer, that is $K \times \frac{M}{B} \times \frac{N}{B}$ and number of filters $K = 256$. The output layer is designed so that it matches the size of the original image.

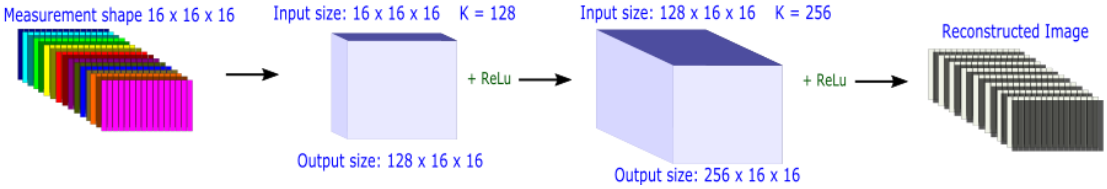


Figure 3.3 – Small supervised CNN architecture.

3.6.2 Small usupervised CNN network

Unlike the supervised network, the unsupervised approach does not take as input compressed measurements using iid Gaussian matrices. Instead, it learns the sensing matrix Φ in the input layer and the subsequent layers reconstruct the original image. Figure 3.4 shows this network.

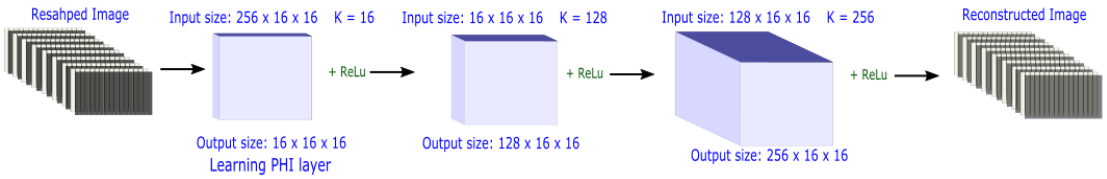


Figure 3.4 – Small unsupervised CNN architecture. Notice the first layer is learning the sensing matrix Φ.

3.6.3 Large supervised CNN network

During the development of this thesis several attemps for reconstructing images using deep learning were published [39, 44, 34, 35, 5]. Interestingly, Kulkarni *et al.* [39] also addressed the problem of CS recovery using a CNN. The other approaches are very similar among themselves because they are based on an auto-encoder. The most interesting findings, however, were made by Adler *et al.* [5]. They introduced a new parameter that controls the redundancy of the network. Redundancy refers to how large the network will be in relation with the size of the

original image. For example, by taking into account the redundancy factor a reconstruction layer will have a size $redundancy \times N$. The value 8 turned out to yield the best reconstruction performance according to their tests. In our case the image size is $N = 256$ giving 256×8 feature maps $K = 2048$ followed by another twin layer.
Based on the previous concepts and proved capabilities of the small network we proposed another network sketched in Figure 3.5. Unlike small network, large network adds two extra reconstruction layer, making deeper.

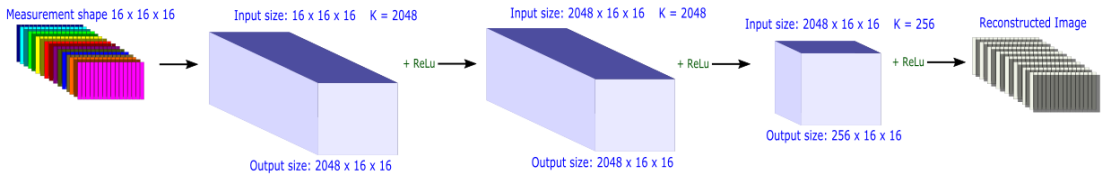


Figure 3.5 – Large supervised CNN architecture.

3.6.4 Large unsupervised CNN network

The unsupervised approach for the large network is the same as with the small network. That is, the first layer takes the original image (interpreted as blocks) which was also reshaped into a 3D tensor. Doing that, allows the network to learn the sensing matrix, in fact the first layer is interpreted as the compression step. The second layer and third layers are called reconstruction layers. The output layer reshapes the data back to the original image size and is the last step in the recovery process.

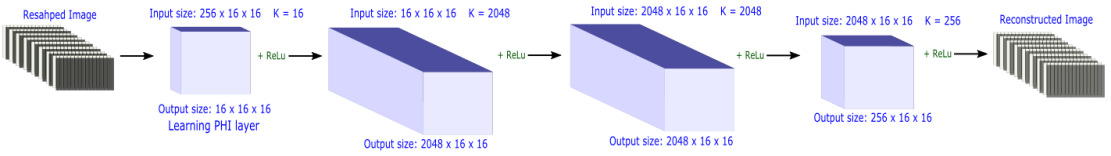


Figure 3.6 – Large unsupervised CNN architecture. First layer is learning the sensing matrix Φ.

3.7 Training CNN's

Training is the process of learning the values of the parameters of the networks. During this step one is concerned with choosing an appropriate weight initialization, loss function, parameter update rule and batch size.

3.7.1 Weight initialization

Choosing the right initial values of the network is needed to secure a sucessful convergence. Furthermore, it can avoid problems like vanishing gradients or ending up being stuck in local minima. Using independent Gaussian random numbers is a very common practice for simple networks but, as complexity increases, smarter ways had been investigated [31]. Glorot and Bengio empirically validated a method to initialize the weights called normalized initialization

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}] \tag{3.1}$$

were n_j is the layer size of layer j. In this thesis we follow the same approach.

3.7.2 Loss function

Since we need a measure of how good the reconstruction from compressed measurements is, we need to define a way to compare the difference (error) against the original image. Not only that, the loss function also serves as an objective that we would like to minimize as much as possible. The choice of the loss functions depends on the type of problem that one is facing, regression or classification, in order to achieve better results. We have chosen to use the mean square error as the loss function (MSE) because it is highly related to an evaluation metric (PSNR) that we will explain later. The MSE is mathematically defined as

$$L(w) = \frac{1}{N} \sum_{n=1}^{N} \| f(x_n, w) - y_n \|_2^2 \tag{3.2}$$

were N is the number of images in the dataset, $\hat{x} = f(x_n, w)$ represents the nonlinear mapping computed by the CNN (reconstructed image) and y_n is the original image. Using backpropagation, we train the CNN by minimizing the loss function defined in 3.2.

3.7.3 Parameter update rule

The easiest way to update the weights ω is by following the negative direction of the loss function using mini batch SGD as we explanied previously 2.2.3. As with initialization methods, the algorithms to find a minimum abound. For our training, we use Adam (Adaptive moment estimation) [37] as it converges faster compared to other algorithms.

It works by storing the weighted decay of the previous square gradient v_t and previous gradient m_t. Subsequently, v_t and m_t are corrected in order to eliminate initial bias. Those estimates are used along with SGD as a way to adapt learning rates for parameter ω. The full algorithm as we use it goes

Algorithm 4 Adam update

Require: $\alpha = 0.0005$: Stepsize, $\epsilon = 10^{-8}$
Require: $\beta_1 = 0.9$, $\beta_2 = 0.999$: Exponential decay rates for moment estimates
Require: $L(\omega)$: Loss function
Require: ω_0: Initial parameter values
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while ω_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\omega L_t(\omega_{t-1})$ (Gradient of loss function at time step t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second moment estimate)
$\hat{m}_t \leftarrow \frac{m_t}{(1-\beta_1^t)}$ (Compute bias-corrected first moment estimate)
$\hat{v}_t \leftarrow \frac{m_t}{(1-\beta_2^t)}$ (Compute bias-corrected second moment estimate)
$\omega_t \leftarrow \omega_{t-1} - \alpha \cdot \frac{\hat{m}_t}{(\sqrt{\hat{v}_t}+\epsilon)}$ (Update parameters)
end while
return ω_t (Resulting parameters)

3.7.4 Batch size training

For training our CNN's and results presented later we have used a batch size of 20.

3.8 Postprocessing of reconstructed images

As it can be inferred from Figure 3.6 that the output of the CNN is not an image. Rather, it is a 3D tensor from which, with minor extra postprocessing, we recover the image. This process aims to invert the prepocessing step explained in section 3.4.
First the image is reshaped back to ist original size $M \times N$. Then using col2im MATLAB commnad we get the reconstructed image ready to be visualized. Image 3.7 shows the procedure. Optionally, we also make use of the very common image denoiser BM3D [21]. That is because we would like to get rid of the block nature previously explained.

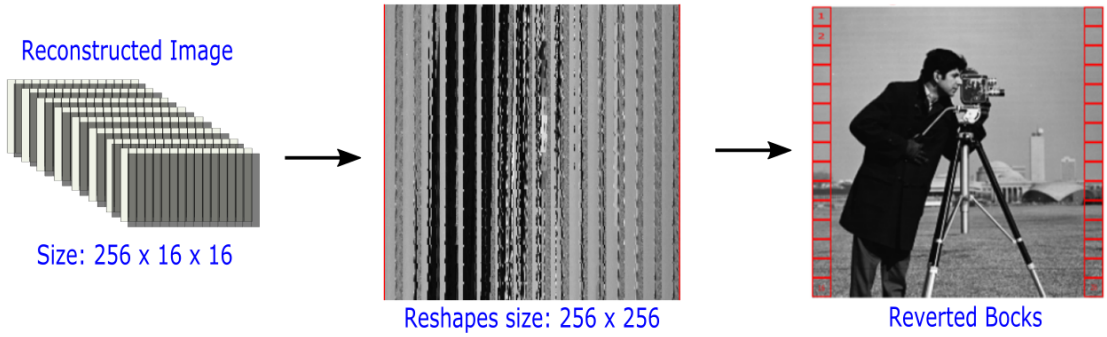


Figure 3.7 – Postprocessing illustration to visualize the reconstructed image.

3.9 Evaluation metrics

Whenever images are undergoing any type of processing, they are bound to suffer a deprecation of the visual quality. Namely, for applications that are ultimately perceived by human eye, there is a need to precisely evaluate the degradation rate (noise) in the image that a certain algorithm adds. Even though agreeing on a metric that is capable of fairly measureing the visual impact is somewhat subjective, there are two methods widely used for that purpose Peak signal-noise-to ratio (PSNR) and Structural similariry index (SSIM) [53].

3.9.1 PSNR

The most common and straightforward way to compute measurement metric is the mean square error (MSE) 3.2. It is obtained by averaging the difference between the original pixel values of an image and the pixel values of a reconstructed image. By using the MSE of the original image I and reconstructed image RI, the PSNR expressed in decibels (dB) is calculated as

$$MSE = \frac{1}{NN}\sum_{i=1}^{N}\sum_{j=1}^{N}[I(i,j) - RI(i,j)]^2 \tag{3.3}$$

$$PSNR = 10 \cdot log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{3.4}$$

Here, MAX_I^2 is the maximum pixel value of the image. For grayscale images such value is $(2^8 - 1)$ using 8 bits per pixel. PSNR has the advantages of being simple and mathematically suited from the optimization point of view but, it also has two major disadvantages: it fails to capture the effects on the structure of the image which means that two images with the same MSE may have a considerable visual impact difference and it does not consider the human perception system. Desired values, considered good, for PSNR should be close or above 30dB.

3.9.2 SSIM

SSIM is a method for measuring the similarity between two images. Unlike PSRN, it uses a more complex approach to computing the difference in accordance with the human visual perception. It takes into consideration three features of an image: luminance (l), contrast (c) and structure (s). For two images I and RI, each term is evaluated as

$$l(I,RI) = \frac{2\mu_I\mu_{RI} + C_1}{\mu_I^2 + \mu_{RI}^2 + C_1} \tag{3.5}$$

$$c(I,RI) = \frac{2\sigma_I\sigma_{RI} + C_2}{\sigma_I^2 + \sigma_{RI}^2 + C_2} \tag{3.6}$$

$$s(I,RI) = \frac{\sigma_{IRI} + C_3}{\sigma_I\sigma_{RI} + C_3} \tag{3.7}$$

variables μ_I, μ_{RI}, σ_I, σ_{RI} and σ_{IRI} represent local means, standard deviations and cross-covariance of images I and RI. Arguments C_1, C_2 and C_3 are predefined constants. The final index is weighted a multiplication of the terms

$$SSIM(I,RI) = \left[l(I,RI)^\alpha \cdot c(I,RI)^\beta \cdot s(I,RI)^\gamma\right] \tag{3.8}$$

In practice $\alpha = \beta = \gamma = 1$, and $C_3 = \frac{C_2}{2}$ simplifying to

$$SSIM(I,RI) = \frac{(2\mu_I\mu_{RI} + C_1)(2\sigma_{IRI} + C_2)}{(\mu_I^2 + \mu_{RI}^2 + C_1)(\sigma_I^2\sigma_{RI}^2 + C_2)} \tag{3.9}$$

This index can have values ssim $\in [0,1]$ with 0 meaning that the images are not similar at all and with 1 meaning that images are exactly the same. Ssim values above 0.7 are considered sufficiently good for any reconstruction algorithm.

4 Results

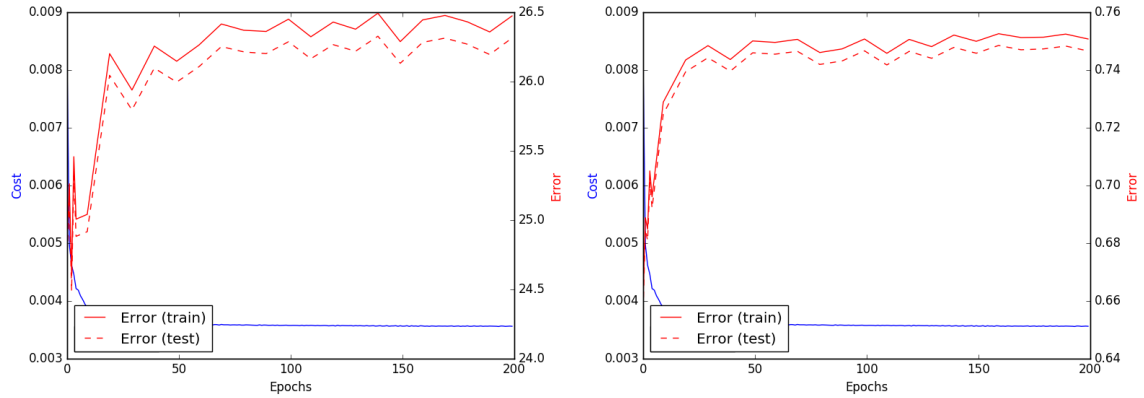
In this chapter we present the outcome of using CNN's for recovering images from compressed measurements. First, we present the small network and its performance using both learning approaches: supervised and unsupervised. We present the training, test and validation error for LabelMe dataset [33]. Afterwards, we also show the images from the test dataset and compare them against the ground truth in terms of PSNR and SSIM. The same data can be expected for the large network. Finally, we present the time measurement that our approach needs for fully reconstructing an image.

4.1 Small network

The small network was describe in section 3.6.2. It is not as deep as the large network and in the following we show its performance.

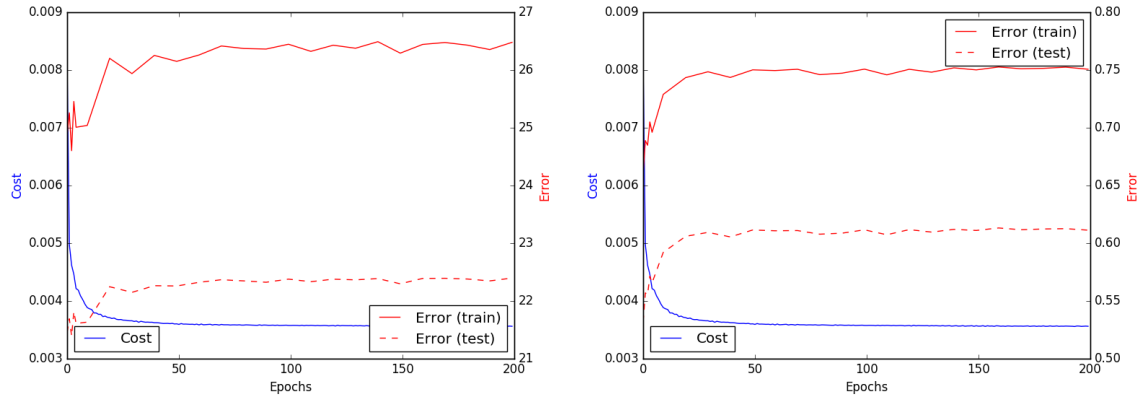
4.1.1 Supervised training

For supervised learning we train the network using compressed measurements generated with the constant matrix Φ. The left figure in 4.1 shows the training of the network for 200 epochs. The blue line is the loss function MSE. The red solid line represents the PSNR on the whole training dataset whereas the red doted line represents the PSNR of the validation dataset. As it can been seen the PSNR on the validation set is not as high as in the training dataset but it is close. That means our network is not suffering from overfitting. On the right figure in 4.1 is the SSIM progress on the validation dataset over each epoch. Figure 4.2 shows the progress of PSNR and SSIM during training for the testing dataset. One can see that even though the difference is higher, approximately 4 dB, the reconstruction is considerably good. Higher values of PSNR and SSIM indicate better recovery quality.



(a) (b)

Figure 4.1 – Error=PSNR (left) and Error=SSIM (right) progress over each epoch of supervised small network on validation dataset.

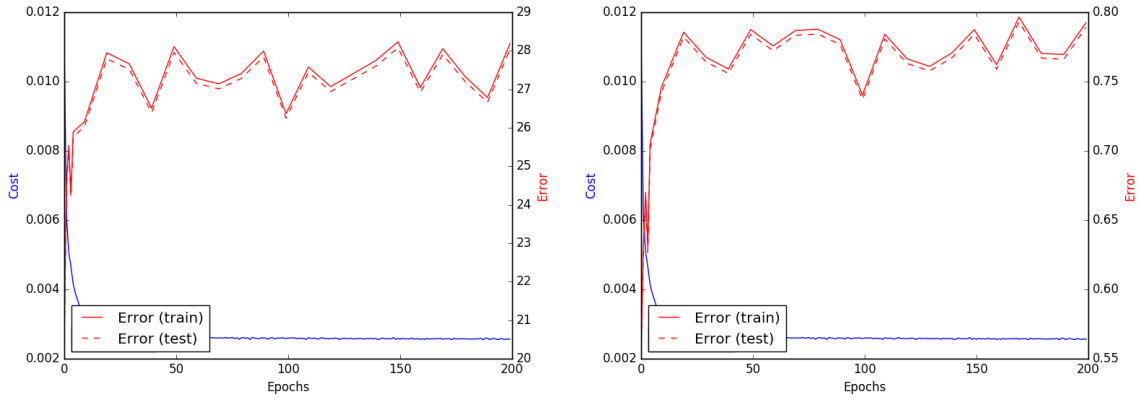


(a) (b)

Figure 4.2 – Error=PSNR (left) and Error=SSIM (right) progress over each epoch of supervised small network on testing dataset.

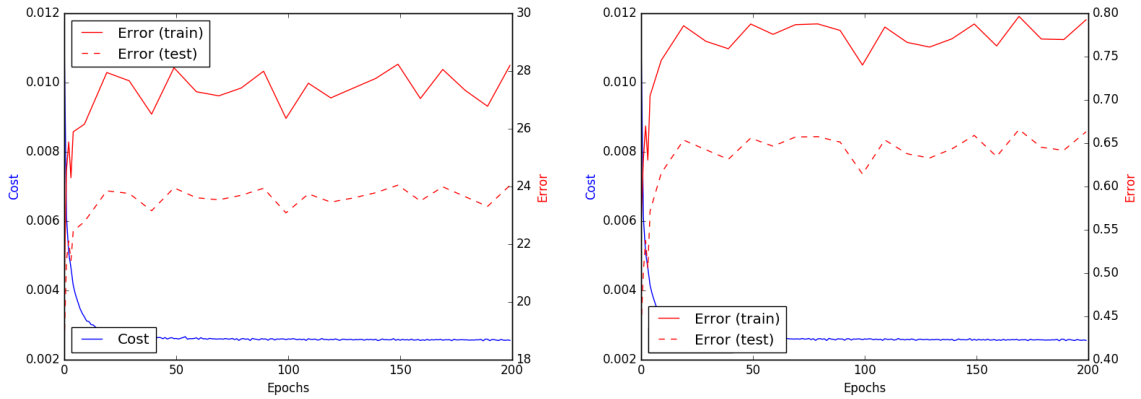
4.1.2 Unsupervised training

For unsupervised learning the training data was not compressed beforehand. That is, the matrix Φ is intrinsically learned as the first layer of the network and the subsequent layers aim to reconstruct the image. Figure 4.3 shows the progress of PSNR and SSIM over 200 epochs on validation datasets. Figure 4.4 depicts the same values for testing dataset using the same network. As it was expected learning the matrix Φ yielded better results. Namely, there was an increase of approximately 1.2 dB for PSNR and 0.02 for SSIM.



(a) (b)

Figure 4.3 – Error=PSNR (left) and Error=SSIM (right) progress over each epoch of unsupervised small network on validation dataset.



(a) (b)

Figure 4.4 – Error=PSNR (left) and Error=SSIM (right) progress over each epoch of unsupervised small network on testing dataset.

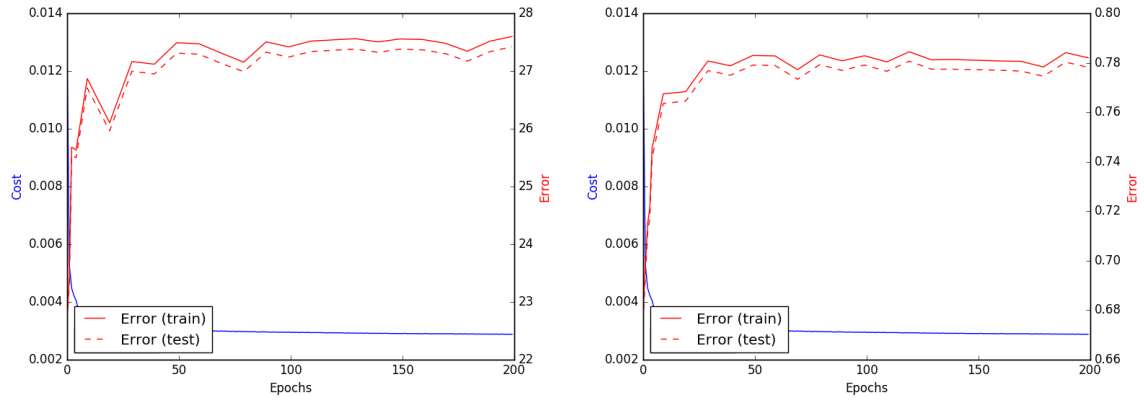
4.2 Large Network

Large network is much deeper compared to small network. It was described in section 3.6.2 and in the following we present the results obtained during training.

4.2.1 Supervised training

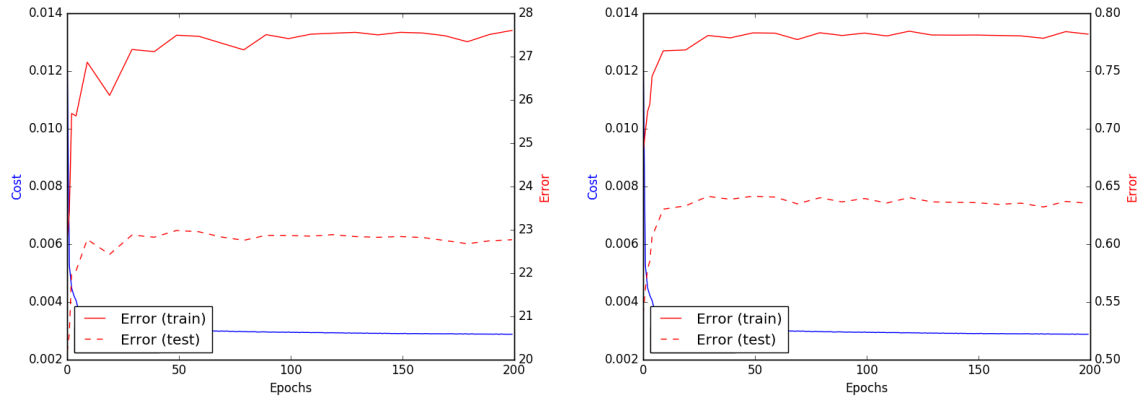
This approach reconstructs images from compressed measurements. Like with small networks, we show the results after 200 training epochs. Figure 4.5 shows the progress of PSNR and SSIM on the validation dataset. Large network, improves PSNR in approximately 1.1 dB and SSIM

0.03 with respect to small network. Figure 4.6 shows the same information explained before on the testing datatset.



(a) (b)

Figure 4.5 – Error=PSNR (left) and Error=SSIM (right) progress over each epoch of supervised large network on validation dataset.



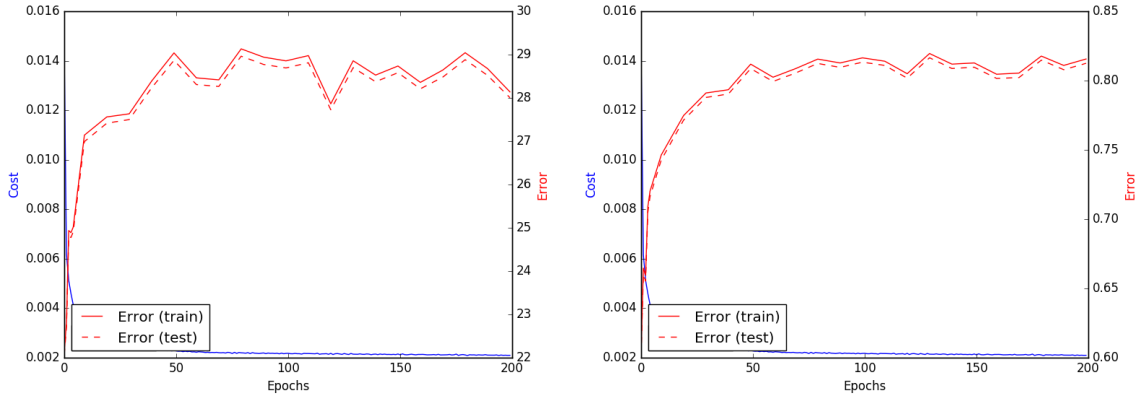
(a) (b)

Figure 4.6 – Error=PSNR (left) and Error=SSIM (right) progress over each epoch of supervised large network on testing dataset.

4.2.2 Unsupervised training

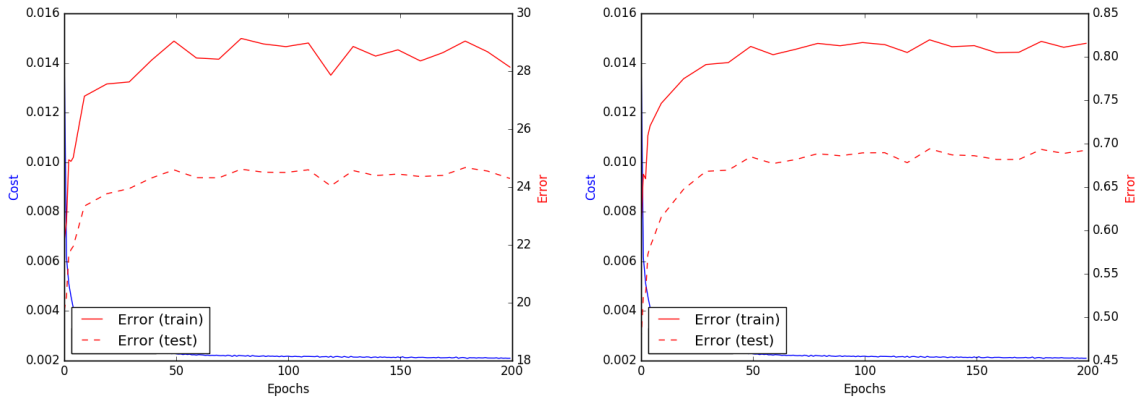
The training for unsupervised learning uses the reshaped original images as its input. The compression phase is part of the network architecture and in the following we present the results of the training. Figure 4.7 shows the progress in PSNR and SSIM over each epoch in

the validation dataset and figure 4.8 on the testing dataset. Compared to the small network it improves PSNR by 1 dB and SSIM by 0.04.



(a) (b)

Figure 4.7 – Error=PSNR (left) and Error=SSIM (right) progress over each epoch of unsupervised large network on validation dataset.



(a) (b)

Figure 4.8 – Error=PSNR (left) and Error=SSIM (right) progress over each epoch of unsupervised large network on testing dataset.

4.3 Results summary

In this section we present a compendium of training and outcome of all proposed architecture networks for recovering the images. Table 4.1 shows PSNR for each dataset and Table 4.2 does the same for SSIM. Additionaly, we add a column representing the denoised testing images using BM3D[21]. Using BM3D does not increase quality considerably but, it helps remove the block representation introduced during preprocessing. It is clear that large network's performance is better.

Table 4.1 – Comparison of PSNR performance for each proposed network.

Network	PSNR TRAINING	PSNR VALIDATION	PSNR TESTING	PSNR TESTING DENOISED
Small Supervised	26.47 dB	26.31 dB	22.37 dB	22.54 dB
Small Unsupervised	28.19 dB	28.03 dB	24.02 dB	24.24 dB
Large Supervised	27.60 dB	27.42 dB	22.73 dB	22.91 dB
Large Unsupervised	29.14 dB	28.96 dB	24.63 dB	24.77 dB

Table 4.2 – Comparison of SSIM performance for each proposed network.

Network	SSIM TRAINING	SSIM VALIDATION	SSIM TESTING	PSNR TESTING DENOISED
Small Supervised	0.7507	0.7466	0.6062	0.6195
Small Unsupervised	0.7925	0.7891	0.6533	0.6715
Large Supervised	0.7819	0.7781	0.6306	0.6381
Large Unsupervised	0.8155	0.8124	0.6873	0.6947

4.4 Reconstruction of testing dataset images

In this section we show all testing images and the difference in reconstruction capabilities for each network. Each figure will provide a visual insight of the perfomance of each network. See figures 4.9, 4.10, 4.11 and 4.12.

Figure 4.9 – Reconstructed barbara, boats, bridge and cameraman compared against the ground truth, PSNR and SSIM.

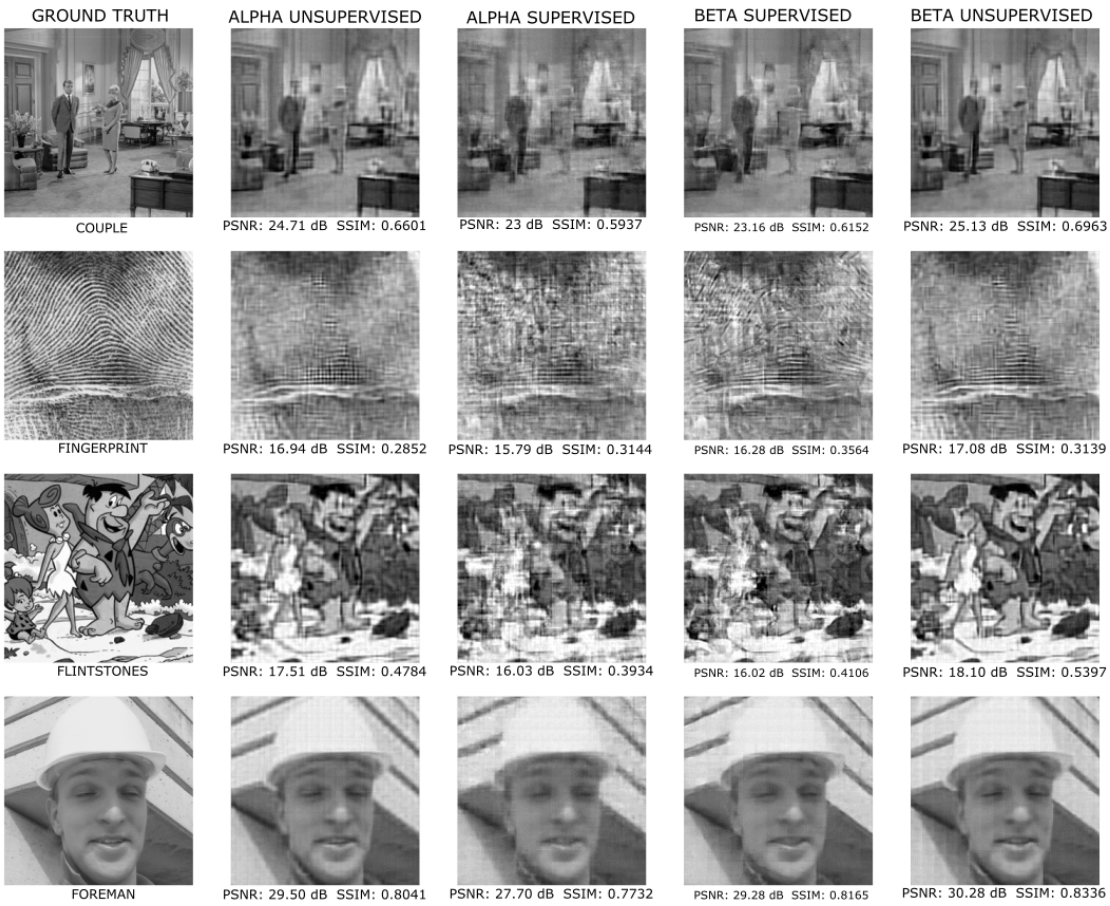


Figure 4.10 – Reconstructed couple, fingerprint, flintstones and foreman images compared against the ground truth, PSNR and SSIM.

Figure 4.11 – Reconstructed house, lena, man and mandrill images compared against the ground truth, PSNR and SSIM.

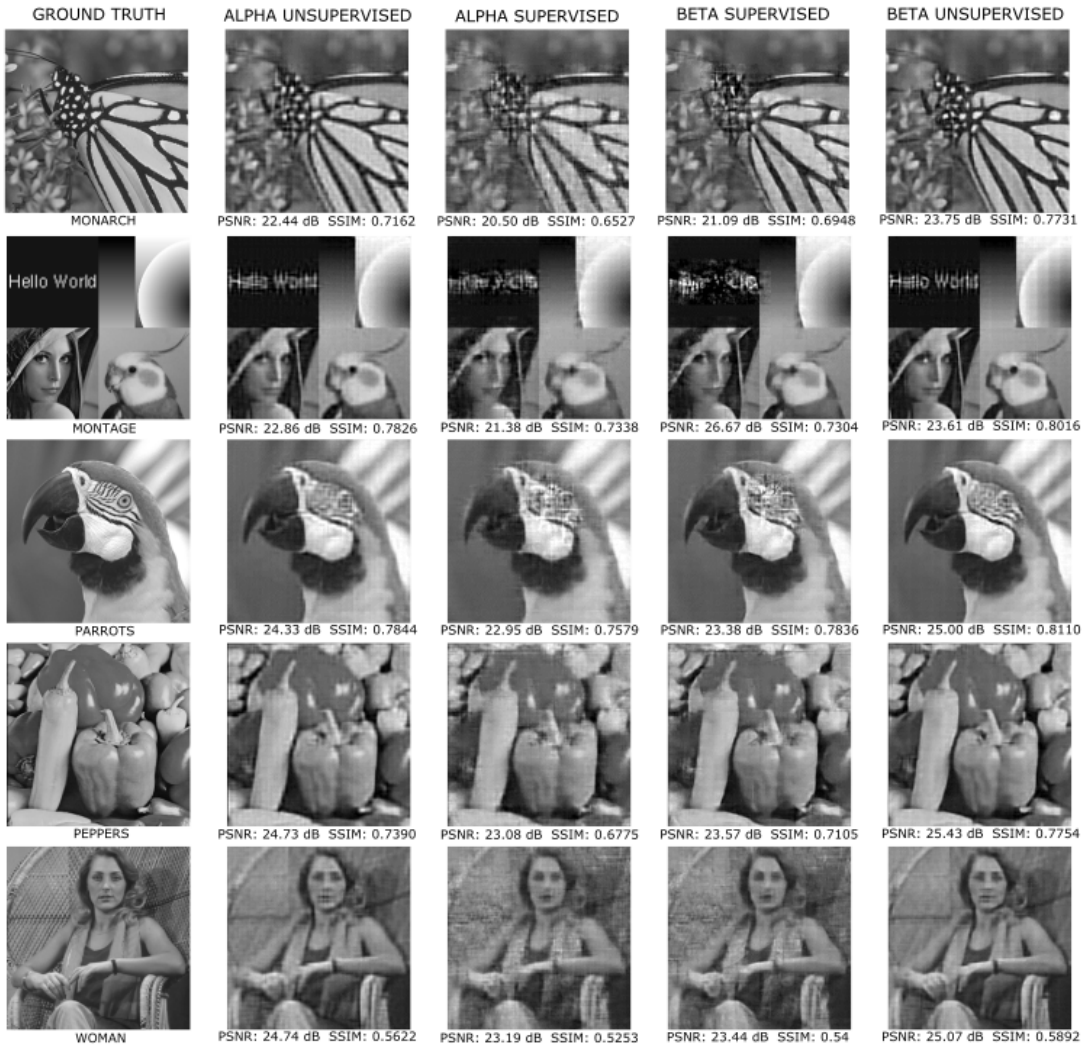


Figure 4.12 – Reconstructed monarch, montage, parrot, peppers and woman images compared against the ground truth, PSNR and SSIM.

4.5 Computational reconstruction time and quality comparison against traditional methods

Table 4.3 compares our approach against some of the most recognized state-of-the-art algorithms for reconstruction. The table shows the average computational time and quality for the testing dataset. It is clear that using CNN's performs better in terms of quality and speed up. Particularly, using large network yields the best visual impact quality and speeds the recovery process more than 100x for NLR-CS, 90x for TVAL3 and 188x for D-AMP. It is important, however, to mention that the reported time for our approach is obtained using an Intel Xeon Core E5-2690 2.9GHz and the MATLAB code provided by their authors. Their code was adapted to have a fair comparison with our settings. That is using the same image size, compression rate and block size. The code implementing their proposed recovery algorithm was not touched. Our CNN's did not use the GPU for this test, we use the same CPU. Using GPU will considerably increase the speed-up even more.

As an example, figure 4.13 shows the the reconstruction of cameraman using several algorithms and our large network. While PSNR is similar among different approaches, the visual impact and SSIM of our network performs better.

Table 4.3 – Comparison of reconstructing testing dataset with several algorithms with subrate $\frac{1}{16}$.

Network	TIME(sec)	PSNR	SSIM
TVAL3	255.50	23.04 dB	0.5863
NLR-CS	289.31	20.59 dB	0.5575
D-AMP	531.61	20.05 dB	0.3628
Small Supervised	0.1654	22.39 dB	0.6119
Small Unsupervised	0.3856	24.04 dB	0.6595
Large Supervised	0.6099	22.77 dB	0.6360
Large Unsupervised	0.7612	25.18	0.6937

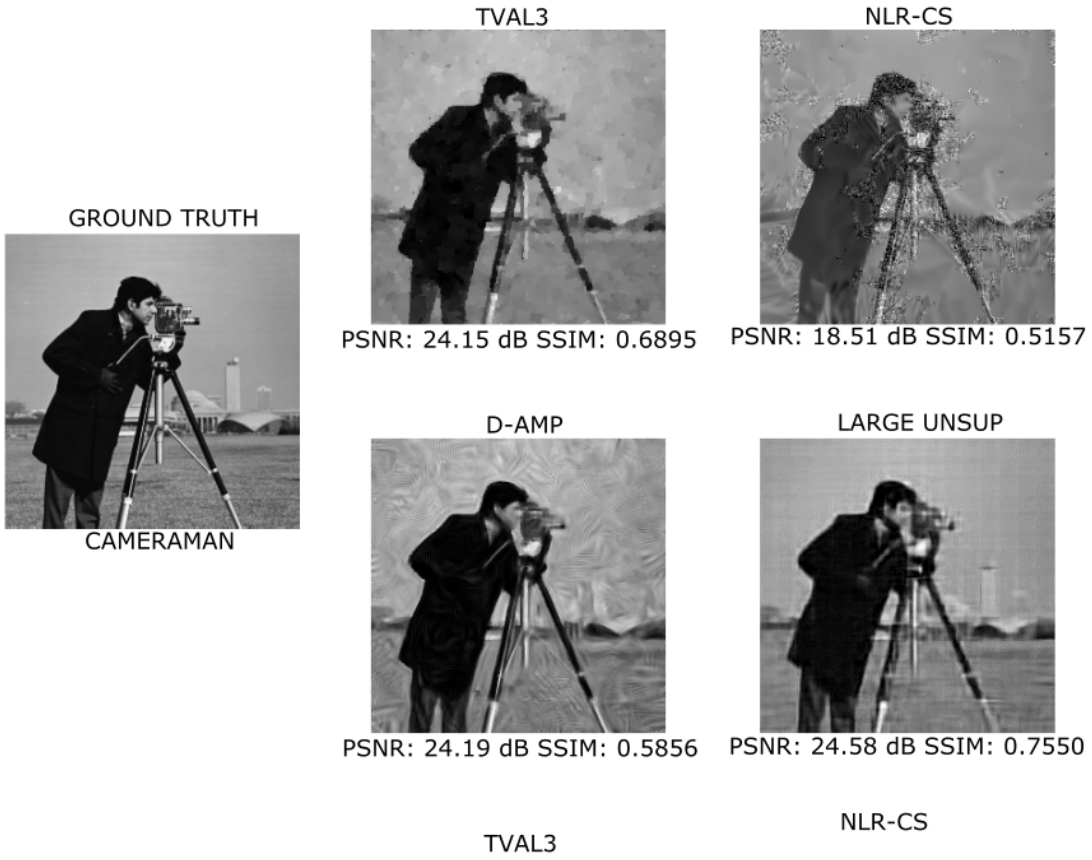


Figure 4.13 – Reconstructed cameraman using traditional CS algorithms and large network.

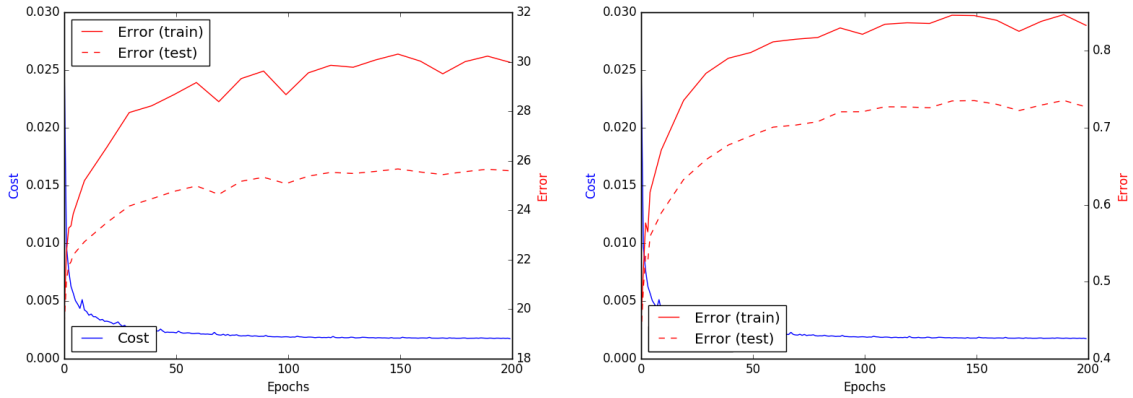
4.6 Reconstruction with alternate compression rates

In this section we present the outcome when using subrates $\frac{1}{10}$, $C = 10$ and $\frac{1}{8}$, $C = 8$. We are only using large network because it produces the best results. As it will be seen, the gain in information is about 2 dB for PSNR and 0.05 for SSIM.

4.6.1 Compression rate 1/10

Here we present the performance of our approach when reconstructing lena, barbara and woman. We also present the performance of the network during training on the testing dataset. See igure 4.14.

Figure 4.16 shows the reconstructed images with subrate $\frac{1}{10}$. Comparing against subrate $\frac{1}{16}$ there is an expected improvement and the visual impact was also well preserved. We show this result so is easy to evaluate the trade-off between quality and compression ratio.

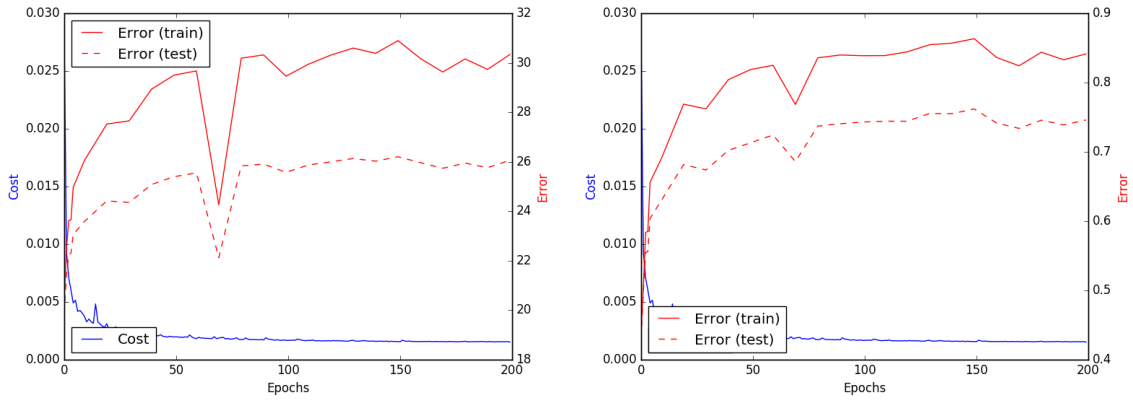


(a) (b)

Figure 4.14 – Error=PSNR (left) and Error=SSIM (right) development of testing dataset during training for compression ratio 1/10.

4.6.2 Compression rate 1/8

In this section we introduce the same results as before for a compression ration of $\frac{1}{8}$. Figure 4.15 shows training phase.



(a) (b)

Figure 4.15 – Error=PSNR (left) and Error=SSIM (right) development of testing dataset during training for compression ratio 1/8.

Figure 4.16 shows the reconstructed images with both subrates. It is also evident that decreasing the compression ratio also helps getting rid off the block effect which would not require any denoising. This is also another feature of our network proposal.

Figure 4.16 – Reconstructed barbara, lena and woman using subrates 1/8 and 1/10.

5 Conclusion

In this thesis we have proposed a method that, after evaluation of our experiments, successfully recovers original images from compressed measurements. We have found out that this method is able to perform much faster and improves quality and visual impact when compared to traditional methods. Namely, we have introduced an approach that uses deep learning with CNN's. Two different network architectures have been proposed: small network and large network. Small network has a performance that is not as good as the large network but, it has the advantage of being lighter and thereby performing twice as fast. Besides, the loss in reconstruction quality is not large. Large network is deeper and has more parameters, but its performance is better. In fact, it produces images that are 2 to 4 dB's higher in terms of PSNR when compared to iterative state-of-the-art methods publicly available. Not only that but, our method seems to be more capable of encapsulating and reconstructing the original structure of the image which is supported by the higher values of SSIM. Having high values of SSIM provides a better visual impact. Besides, if we decrease the compression ratio our network gives images that without further processing seem appealing to the human eye. That is, no extra denoising phase is needed.

Based on our experimentation we have also demonstrated that while learning the sensing matrix Φ imposes extra design constraints, it also improves considerably reconstructed images as compared to i.i.d. Gaussian matrices. Furthermore, learning the sensing matrix Φ may avoid generating random matrices in hardware, which is a very expensive process.

Even though our approach is faster because it makes used of optimized GPU's, it may still be a good choice for many real time applications due to the fact that iterative methods are very slow. With this approach we can give a speed up of up to 188x using the heaviest large network. Using the other networks would increase that ratio even more.

5.1 Future work

There are interesting ideas still remaining to be tested. First, we believe that increasing and extending the number of training images will ultimately generate better results. Therefore, changing the training dataset for a much bigger one seems to be promising. Second, trying to reconstruct images with higher resolution like 4K is also good way to asses our implementation. We think that our approach is not sensible to image size but having real proof is also necessary to support this claim. In fact, there is some evidence that scaling the image size produces even better results. Third, having a fixed sensing matrix Φ is also a good improvement but another thing to consider is binarizing it. That is, the elements of the sensing matrix only have values 0 or 1. That would allow even cheaper hardware implementation for CS sensors. Recently, there has been development on this topic. Integrate batch normalization during the training phase also may help improve the reconstruction of the images but it still needs to be further investigated.

A Learned sensing matrices and network parameters

A.0.1 Unsupervised small network

Figure A.2 shows the graphical representation of the sensing matrix learned during training. Each small square is interpreted as the subsampling matrix for each input dimension.

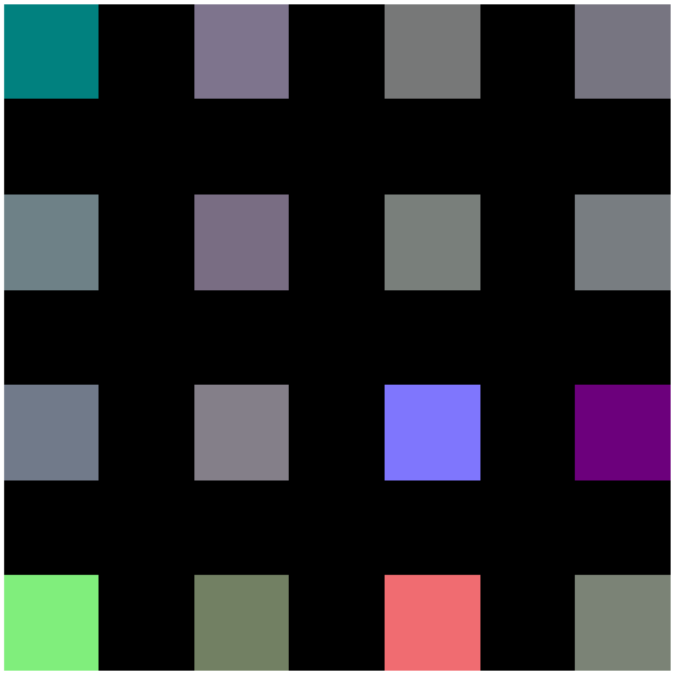


Figure A.1 – Sensing matrix for small network.

A.0.2 Unsupervised large network

Figure A.2 shows the graphical representation of the sensing matrix learned during training. Each small square is interpreted as the subsampling matrix for each input dimension.

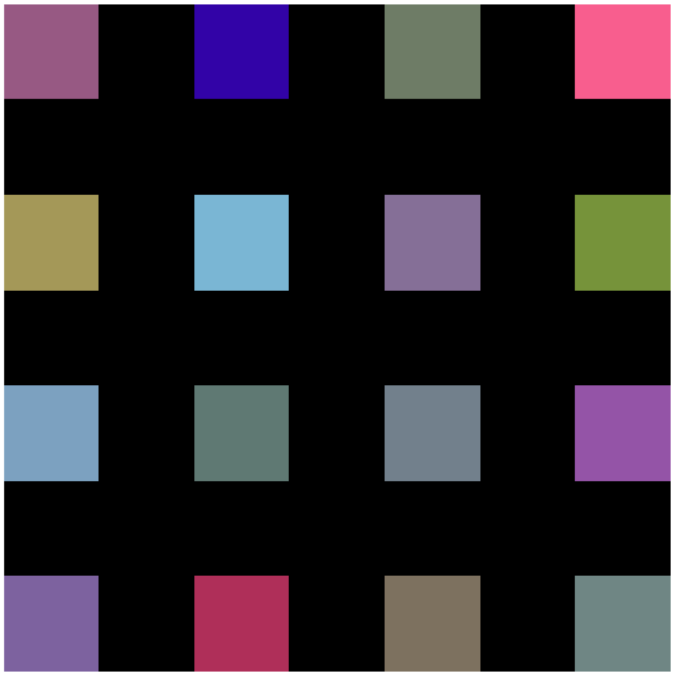


Figure A.2 – Sensing matrix for large network.

A.0.3 Network implementation settings

Table A.1 lists the details of each of our networks for reconstruction images.

Table A.1 – Implementation details of each proposed network.

Network	# of layers	# hidden layers	# parameters ω	appprox. train time
Small Supervised	2	0	313728	3.3 hrs
Small Unsupervised	3	1	350608	4.85 hrs
Large Supervised	3	1	42766592	18 hrs
Large Unsupervised	4	2	42803472	40 hrs

Bibliography

[1] Artificial neural network. https://en.wikipedia.org/wiki/Artificial_neural_network. [Online; accessed 26-08-2016].

[2] CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.stanford.edu/. [Online; accessed 02-09-2016].

[3] Deep Learning Compared. https://http://deeplearning4j.org/compare-dl4j-torch7-pylearn.html. [Online; accessed 29-08-2016].

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] A. Adler, D. Boublil, M. Elad, and M. Zibulevsky. A deep learning approach to block-based compressed sensing of images. *http://arxiv.org/abs/1606.01519*, 2016.

[6] R. G. Baraniuk. Compressive sensing. *IEEE signal processing magazine*, 24(4), 2007.

[7] C. M. Bishop. Pattern recognition and machine learning, 2006.

[8] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with bm3d? In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2392–2399. IEEE, 2012.

[9] E. Candes and J. Romberg. Sparsity and incoherence in compressive sampling. *Inverse problems*, 23(3):969, 2007.

[10] E. J. Candes. The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Mathematique*, 346(9):589–592, 2008.

[11] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006.

[12] E. J. Candes and J. K. Romberg. Signal recovery from random projections. In *Electronic Imaging 2005*, pages 76–86. International Society for Optics and Photonics, 2005.

[13] E. J. Candès and J. K. Romberg. Errata for quantitative robust uncertainty principles and optimally sparse decompositions. *Foundations of Computational Mathematics*, 7(4):529–531, 2007.

[14] E. J. Candes, J. K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, 59(8):1207–1223, 2006.

[15] E. J. Candes and T. Tao. Decoding by linear programming. *IEEE transactions on information theory*, 51(12):4203–4215, 2005.

[16] E. J. Candes and T. Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory*, 52(12):5406–5425, 2006.

[17] E. J. Candès and M. B. Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30, 2008.

[18] C. Chen, E. W. Tramel, and J. E. Fowler. Compressed-sensing recovery of images and video using multihypothesis predictions. In *Asilomar Conference on Signals, Systems, and Computers*, 2011.

[19] F. Chollet. Keras. https://github.com/fchollet/keras, 2015.

[20] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[21] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on image processing*, 16(8):2080–2095, 2007.

[22] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, diogo149, B. McFee, H. Weideman, takacsg84, peterderivaz, Jon, instagibbs, D. K. Rasul, CongLiu, Britefury, and J. Degrave. Lasagne: First release., Aug. 2015.

[23] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *Computer Vision–ECCV 2014*. Springer International Publishing, 2014.

[24] W. Dong, G. Shi, X. Li, Y. Ma, and F. Huang. Compressive sensing via nonlocal low-rank regularization. *IEEE Transactions on Image Processing*, 23(8):3618–3632, 2014.

[25] D. L. Donoho. Compressed sensing. *IEEE Trans. Information Theory*, 52(4):1289–1306, 2006.

[26] M. F. Duarte, M. A. Davenport, D. Takhar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, 25(2):83–91, 2008.

[27] J. E. Fowler, S. Mun, and E. W. Tramel. Multiscale block compressed sensing with smoothed projected landweber reconstruction. In *European Signal Processing Conference (EUSIPCO)*, pages 564–568, 2011.

[28] J. E. Fowler, S. Mun, and E. W. Tramel. Block-based compressed sensing of images and video. *Foundations and Trends in Signal Processing*, 4(4):297–416, 2012.

[29] R. Frischholz and U. Dieckmann. Bioid. *HumanScan Inc.,"http://www. human scan. de/products/bioid/bioid31. php*, 2003.

[30] L. Gan. Block compressed sensing of natural images. In *2007 15th International conference on digital signal processing*, pages 403–406. IEEE, 2007.

[31] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[32] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.

[33] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[34] M. Iliadis, L. Spinoulas, and A. K. Katsaggelos. Deep fully-connected networks for video compressive sensing. *arXiv preprint arXiv:1603.04930*, 2016.

[35] M. Iliadis, L. Spinoulas, and A. K. Katsaggelos. Deepbinarymask: Learning a binary mask for video compressive sensing. *arXiv preprint arXiv:1607.03343*, 2016.

[36] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[37] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[39] K. Kulkarni, S. Lohit, P. Turaga, R. Kerviche, and A. Ashok. Reconnet: Non-iterative reconstruction of images from compressively sensed random measurements. *arXiv preprint arXiv:1601.06892*, 2016.

[40] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[41] C. Li, W. Yin, H. Jiang, and Y. Zhang. An efficient augmented lagrangian method with applications to total variation minimization. *Computational Optimization and Applications*, 56(3):507–530, 2013.

[42] S. Mallat. *A wavelet tour of signal processing*. Academic press, 1999.

[43] C. A. Metzler, A. Maleki, and R. G. Baraniuk. From denoising to compressed sensing. 2014.

[44] A. Mousavi, A. B. Patel, and R. G. Baraniuk. A deep learning approach to structured signal recovery. *arXiv preprint arXiv:1508.04065*, 2015.

[45] S. Mun and J. E. Fowler. Block compressed sensing of images using directional transforms. In *2009 16th IEEE international conference on image processing (ICIP)*, pages 3021–3024. IEEE, 2009.

[46] N. Ratlif. Lecture notes mathematics for intelligent systems, October 2014.

[47] S. Ruder. An overview of gradient descent algorithms. http://sebastianruder.com/about/. [Online; accessed 05-09-2016].

[48] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1):157–173, 2008.

[49] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[50] M. Toussaint. Lecture notes machine learning and optimization, April 2015.

[51] B. Van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio. Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619*, 2015.

[52] M. B. Wakin, J. N. Laska, M. F. Duarte, D. Baron, S. Sarvotham, D. Takhar, K. F. Kelly, and R. G. Baraniuk. An architecture for compressive imaging. In *2006 International Conference on Image Processing*, pages 1273–1276. IEEE, 2006.

[53] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[54] Y. Weiss, H. S. Chang, and W. T. Freeman. Learning compressed sensing. In *Snowbird Learning Workshop, Allerton, CA*, 2007.

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, Germany Luis Manuel Bracamontes Hernandez