


UNIVERSITAT POLITÈCNICA DE CATALUNYA
UNIVERSITAT DE BARCELONA
UNIVERSITAT ROVIRA I VIRGILI

Master in Artificial Intelligence

Master of Science Thesis

Synthetic Data Generation for Deep Learning in Small Datasets

Hadi Keivan Ekbatani

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
FACULTAT DE MATEMÀTIQUES (UB)
ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA (URV)

Supervisor:

Oriol Pujol Vila

Department of analysis
and applied Mathematics,
Universitat de Barcelona (UB)

Co-supervisor:

Santiago Segui Mesquida

Department of analysis
and applied Mathematics,
Universitat de Barcelona (UB)

April 10, 2016

Acknowledgments

I would like to sincerely thank my supervisors Oriol Pujol and Santi Segui for their support, guidance and mentorship. I greatly appreciate their demanding and inquisitive scientific attitude, while keeping always a calm and positive mindset. They are, in my mind, an inspiring example of what university professors should be.

Furthermore, I would also like to dedicate this master thesis to my beloved parents and sister for their unsparing supports. My gratitude knows no bounds.

Another special acknowledgment must be made to my friends in the master: Denis, Jeroni, Philipp, Pablo, Lorenzo, Iosu, Ferran and many others for the endless studying hours, morning coffees after crunching the brutal assignments all night long, valuable and constructive discussions which allowed me carry out this program.

Abstract

This master's thesis introduces an analysis on the generation of synthetic datasets as a pertinent substitute for missing large training sets in learning algorithms. Furthermore, it endeavors to enhance feature detection in learning to count problems by application of Deep Learning (DL) algorithms.

In the course of our research, we introduce two highly realistic synthetically created sets of images to train proposed Deep Convolutional Neural Networks (DCNN) to approach two tasks: learning to count even-digits and counting the number of pedestrians in a walkway. In the former case, we analyze the applicability of deep features as a surrogate to hand-crafted features for understanding the underlying representations. Subsequently, in the latter task, the features learned by training the DCNN on synthetic images, are tested on a small real-world dataset of pedestrians to overcome the inefficiency of deep learning algorithms when facing lack of data.

To evaluate our methodology, we present an empirical experiment followed by comparison to the state-of-the-art methods. Improving the cutting-edge solutions, our results demonstrate the suitability of deep models for learning objects features for counting tasks in Computer Vision (CV). Furthermore, obtaining notable results on real-word problems compared to previous highly specialized approaches, suggests synthetic data as a workaround for application of deep learning to problems with small datasets.

Contents

1 Introduction **2**
- 1.1 Motivation . . . 2
- 1.2 Objectives . . . 2
- 1.3 Contributions . . . 3
- 1.4 Organization . . . 4

2 Background and Definitions **5**
- 2.1 Deep Learning . . . 5
- 2.2 Artificial Neural Networks . . . 6
  - 2.2.1 Activation Functions . . . 7
  - 2.2.2 Multi-layer Neural Network . . . 7
    - 2.2.2.1 Training Objective . . . 8
    - 2.2.2.2 Back Propagation . . . 9
  - 2.2.3 Training Algorithm . . . 9
- 2.3 Deep Neural Networks . . . 10
- 2.4 Convolutional Neural Networks . . . 12
  - 2.4.1 Convolutional Layer . . . 12
    - 2.4.1.1 Weight Sharing . . . 13
  - 2.4.2 Pooling/Sub-sampling Layer . . . 13
  - 2.4.3 Local Response Normalization . . . 14
  - 2.4.4 Fully Connected/Inner Product Layer . . . 15

3 State of the Art Review **16**
- 3.1 Synthetic Data Generation . . . 16
- 3.2 Feature Detection Algorithms . . . 18

4 Proposal **24**
- 4.1 Datasets . . . 24
  - 4.1.1 Learning to Count Even-digits Dataset . . . 25
  - 4.1.2 Synthetic Crowd Counting Dataset . . . 26
  - 4.1.3 UCSD Crowd Counting Dataset . . . 26
- 4.2 Method Selection . . . 26
- 4.3 Architecture . . . 27

5 Implementation **29**
5.1 The Datasets . . . 29
5.1.1 Even-odd Digits Dataset . . . 29
5.1.2 Synthetic Pedestrians Dataset . . . 30
5.1.2.1 Data Generation . . . 31
5.1.2.2 Data Improvement . . . 33
5.1.3 UCSD Crowd counting Dataset . . . 35
5.2 Caffe Deep Learning Platform . . . 36
5.2.1 Model Implementation and Design . . . 37
5.2.2 Model Optimization . . . 39
5.2.2.1 Batch Size . . . 39
5.2.2.2 Learning Rate . . . 39
5.2.2.3 Weight Decay . . . 40
5.2.2.4 Momentum . . . 40
5.2.2.5 Number of Iterations . . . 41
5.3 The Architecture . . . 41
5.3.1 Even-digit Counting . . . 41
5.3.2 Crowd Counting . . . 44

6 Experiments and Results **47**
6.1 Learning to Count Even-odd Handwritten Digits . . . 47
6.1.1 Dataset . . . 47
6.1.2 Learning process . . . 48
6.1.3 Experimental Results . . . 50
6.1.4 Conclusion . . . 52
6.2 Counting Pedestrians in a Walkway . . . 52
6.2.1 Datasets . . . 52
6.2.2 The Learning Process . . . 53
6.2.3 Experimental Results . . . 55
6.2.4 Conclusion . . . 57

7 Conclusions and Future Work **58**
7.1 Conclusions . . . 58
7.2 Future Work . . . 59

References **60**

List of Figures

2.1 A single artificial neuron 6
2.2 A Rectified Linear Unit (ReLU) 7
2.3 A neural network composed of three layers 8
2.4 A deep neural network with three hidden layers 11
2.5 The forward propagation phase of a convolutional layer 12
2.6 The forward propagation phase of a pooling layer with stride equal to 1. 14
2.7 Local response normalization in a CNN after the convolution layer. 15

3.1 The basic idea of the synthetic fingerprint generation approach [9]. 17
3.2 An example of synthetically generated images in [24]. Starting from a base image, a random perspective transform is applied, followed by gaussian blur (not shown), color augmentation (exaggerated for viewing purposes) and background replacement. 17
3.3 An example of synthetically generated images by Seguí et al. [94]. 18
3.4 Crowd counting system: the scene is segmented into crowds with different motions. Normalized features that account for perspective are extracted from each segment, and the crowd count for each segment is estimated with a Gaussian process [11]. 19
3.5 Crowd counting results: The red and green segments are the "away" and "towards" crowds. The estimated crowd count for each segment is in the top-left, with the (rounded standard-deviation of the GP) and the ground-truth. The Region Of Interest (the area in the walkway in which the pedestrians are counted and labeled) is also highlighted [11]. 20
3.6 Learning to count hand-written digits problem in which the features of a CNN that has been trained to count digits can be readily used for more specific classification problems and even to localize digits in an image [94]. 22

4.1 A selection of MNIST images of hand-written digits. 25
4.2 Deep CNN proposed by LeCun et al. [62] for MNIST hand-written digits recognition problem. 27
4.3 The basic design for the proposed DCNNs. 28

5.1 An example of even-odd digits images. Form left to right, images contain 0 to 15 even digits. 30
5.2 Sample images of UCSD Anomaly detection dataset. 30
5.3 One background image extracted from UCSD dataset. 31
5.4 Images after background subtraction. 31
5.5 Pedestrians and their corresponding masks. 32

5.6 A synthetically generated background image 32
5.7 Applying the mask of region of interest on the background image. 32
5.8 created synthetic images for counting pedestrians problem. 33
5.9 A selection of non-human or incorrectly labeled objects. As you may see, there are some images with "half a person" and some others with more than just one person in the image. 33
5.10 A few synthetic images with halos around the pedestrians in the walkway. 34
5.11 Some examples of images with no or less halo around the people in the images. . 34
5.12 created images before considering image perspective. 35
5.13 Synthetic images considering image perspective based on the real distribution of people's height. As you may see, after applying perspective, the images look more realistic. 35
5.14 Normal UCSD crowd counting dataset images. 36
5.15 Real UCSD images after being masked and resized. 36
5.16 Input (data) and output (conv1) blobs of a convolution layer in a CNN implemented in Caffe. 37
5.17 From bottom-left to top-right, Lenet architecture for MNIST digit classification example of a Caffe network, where boxes represent layers and octagons represent data blobs produced by or fed into the layers [55]. 38
5.18 From top-left to bottom-right, the network proposed in [94] for Even digits recognition task. 41
5.19 Proposed network architecture for Even digits counting task 43
5.20 From top-left to bottom-right, the DCNN proposed by [94] for learning to count the number of pedestrians in a walkway. 44
5.21 Proposed network architecture for learning the number of pedestrians in the image. 45

6.1 49
6.2 Filters and output of the first convolutional layer in the network. 49
6.3 Filters and output of the second convolutional layer in the network. 50
6.4 51
6.5 54
6.6 54
6.7 55
6.8 56

List of Tables

5.1 The DCNN proposed by [94] for learning the number of even digits present in the images. . . . 42
5.2 Proposed settings for network's hyper-parameters for counting number of even digits task. . . . 43
5.3 Proposed settings for network's hyper-parameters for counting pedestrians. . . . 44
5.4 The deep CNN proposed by Seguí et al. 94 for learning the number of pedestrians in a walkway. . . . 45

6.1 Proposed settings for network's hyper-parameters. . . . 48
6.2 Proposed architecture's settings. . . . 48
6.3 . . . 51
6.4 . . . 52
6.5 Proposed architecture's settings. . . . 53
6.6 Proposed settings for network's hyper-parameters. . . . 53
6.7 . . . 55
6.8 Table caption font is different from the normal text font in order to get a better differentiation – I like it that way. . . . 56
6.9 Table caption font is different from the normal text font in order to get a better differentiation – I like it that way. . . . 57
6.10 Table caption font is different from the normal text font in order to get a better differentiation – I like it that way. . . . 57

1 Introduction

1.1 Motivation

Learning to count is a central concept which constitutes the most fundamental idea of mathematics. In computer vision, the counting problem is the estimation of number of objects in a still image or video frame. Learning to count visual objects is a new approach towards dealing with detecting object in the images and video, which has been recently proffered in the literature [89, 57, 11, 94]. It arises in many real-world applications, including cell counting in microscopic images [28], monitoring crowds in surveillance systems [90], and performing wildlife census or counting the number of trees in an aerial image of a forest [86, 69].

Artificial Intelligence (AI) and computer vision share topics such as pattern recognition and machine learning techniques [76]. Consequently, computer vision is sometimes seen as a part of the artificial intelligence field or the computer science field in general. Recent machine learning methods applied for computer vision tasks, require large number of data for the learning process. In order to learn to count the object of interest in an image or video, various object features need to be designed, extracted or detected during the learning phase. The complexity of feature detection process in vision tasks, restricts their usage in large-scale computer vision applications thus demanding more efficient solutions to alleviate, expedite and improve this process.

One of the recent and commonly used methods to facilitate feature detection process is application of deep convolutional neural networks [100, 58, 60]. One of the promises of DCNN is replacing handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction [97]. DCNNs have been claimed and practically proven to achieve the most assuring performance in different vision benchmark problems concerning feature detection and classification [100, 17].

Although access to fast computers and vast amounts of data has enabled the advances of deep learning algorithms such as DCNN in solving many problems that were not solvable using classic AI[1], they have limitations. For instance, they don't perform well when there is limited data [40]. This constrain restricts the application of DL methods in various areas including computer vision where they have shown promising performances.

1.2 Objectives

This work puts forward several objectives:

[1]Computer programs designed to solve problems that human brains performed easily, such as understanding text or recognizing objects in an image. Due to their poor performance, they introduced *Expert Systems* (computer programs combined with rules provided by domain experts) which were highly tuned for a specific problem.

1. To prevail over the main limitation of deep learning regarding problems with small datasets, by introducing the benefit of synthetic data generation as a surrogate for needed large training sets in fully supervised learning to count tasks in computer vision (when using DL algorithms) where DL methods haven't been applied yet due to the lack of annotated data.

2. To abate extensive highly-specialized feature detection efforts, by substituting deep CNN for learning features in problems concerning counting object of interest, where no explicit information about what we are counting is given to the system, except for the object's multiplicity in the image.

3. To study and analyze the process of generating synthetic images by exploring the behavior of proposed algorithms on synthetic datasets of different types and complexity, and compare the performance with state-of-the-art outcomes [94].

4. To analyze the performance of designed system (trained with synthetic dataset) on real-world crowd counting problems and compare the results with previously developed state-of-the-art systems [11].

1.3 Contributions

Following the course of achieving aforementioned objectives, we created and designed different synthetic datasets and algorithms to provide an exhaustive analysis. Hence, this dissertation contains the following contributions:

1. It provides a thorough study on the procedure of generating synthetic datasets for learning to count the number of objects in images by the means of deep convolutional neural networks. In this course, we present two synthetically created datasets. The first set of images contains hand-written digits for counting the number of even digits in the images. The second dataset consists of pedestrians in a walkway, to train a counting deep convolutional neural network which is adequate for apprehending the underlying representations.

2. It proposes the problem of object representation as an indirect learning problem casted as learning to count strategy. The devised algorithm is capable of counting the number of even hand-written digits in images. Moreover, the model is able to be applied for different but related tasks such as even-odd digit recognition, to demonstrate the capability of the learned features of the network for classifying digits with no direct supervision while training.

3. It suggests a deep convolutional neural network for counting the number of pedestrians in a walkway that does not depend on object detection or feature tracking. The model is privacy-preserving in a sense that instead of tracking people, it learns the individual's features.

4. The proposed model is able to count the number of people in the real and unseen dataset using the features learned by training the network on synthetic dataset. To our knowledge, this is the first crowd counting system trained by synthetic data that successfully operates on real data.

5. Finally, we evaluate our proposal in different aspects regarding our objectives. In addition, fair comparisons between our methodology and different state-of-the-arts will demonstrate our improvements. In general, the validation of our proposal is done in the following steps:

   - First, we learn to count even hand-written digits in images. Then, we evaluate the learned features of the proposed model on an even-odd digits recognition task.
   - Second, we validate a more complex model quantitatively on a large synthetic dataset of pedestrians, containing maximum 29 people in each image.
   - Finally, we test our model's performance by counting the number of crowd in a real-world manually labeled dataset of people present in a walkway provided by Chan and Vasconcelos in [10].

1.4 Organization

This report takes off with a review of deep learning (chapter 2.1) as a branch of artificial intelligence which deep convolutional neural networks belong to, and moves on to introduce a DCNN's basic architecture and components in details (chapter 2.3).

Chapter 3 reviews state-of-the-art of synthetic data generation in CV along with feature detection and learning to count problems.

In Chapter 4, we introduce our created synthetic datasets to be applied in our proposal for constructing a deep neural network to tackle feature detection issue learning to count problems.

Chapter 5 introduces the applied platform to implement our methodology along with the peculiarities of proposed data creation process and network modeling.

Chapter 6 deals with the empirical experiments and analysis. Obtained results and comparisons with state-of-the-art solutions are expressed in this section.

Lastly, in Chapter 7, we conclude the report with a short summary of the scope of work conducted and the new areas of research that this master thesis has opened.

2 Background and Definitions

In this section, we go over the preliminarily concepts that help understand the contributions of this work. We start by looking at the family of methods to which deep convolutional neural networks belong, followed by a more detailed look at DCNNs in question by describing the main components of the method.

2.1 Deep Learning

One of the central challenges of artificial intelligence is solving the tasks that are easy for people to perform but hard for them to describe formally – problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images. One approach to that challenge is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, where each concept is defined in terms of its relation to simpler concepts. This hierarchy of concepts allows the computer to learn complex notions by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, it would be a deep graph with many layers. For this reason, we call this approach *Deep Learning* [38].

Modern deep learning provides a very powerful framework specially for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity. Most tasks that consist of mapping an input vector to an output vector, and that are easy for a human-being to perform quickly, can be accomplished via deep learning, given sufficiently large models and datasets of labeled training examples. Other tasks that cannot be described as associating one vector to another, or that are difficult enough such that a person would require time to think and reflect in order to accomplish the task, remain beyond the scope of deep learning for now [38].

In other words, deep learning is a new area of Machine Learning (ML) research, which has been introduced with the objective of moving ML closer to one of its original goals: artificial intelligence. DL is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound and text.

Various deep learning architectures involves Artificial Neural Networks (ANN). Networks such as *deep neural networks*, *deep convolutional neural networks*, *deep belief networks* and *recurrent neural networks* have been applied to fields like computer vision, automatic speech recognition or natural language processing where they have set the state-of-the-art in recent years, as reviewed by Bengio in [3, 4].

Besides improving the accuracy on different pattern recognition problems, one of the fundamental goals of DL is to move machine learning towards the automatic discovery of multiple levels of representation, reducing the need for feature extractors developed by domain experts

[4]. This is especially important, as noted by Bengio in [3], for domains where the features are hard to formalize, such as for object recognition and detection tasks.

In the task of object detection, deep architectures have been widely used to achieve state-of-the-art results, as in [58], where the top published results use Convolutional Neural Networks (CNN) [17]. Among all deep learning techniques, we will apply deep convolutional neural networks to tackle a specific vision problem. However, prior to delving into deep CNN, we intend to allude the basic concept of neural networks and deep NN briefly, in order to help understand the proposed method in this Master thesis.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are mathematical models that use a collection of simple computational units, called Neurons, interlinked in a network. These models are used in a variety of pattern recognition tasks, such as speech recognition, object detection, identification of cancerous cells, among others [46]. ANN's date back to 1943, with work by McCulloch [74]. The motivation for studying neural networks was the fact that the human brain was superior to a computer at many tasks, a statement that holds true even today for tasks such as recognizing objects and faces, in spite of the huge advances in the processing speed in modern computers.

The neuron is the basic unit on Artificial Neural Networks, and it is used to construct more powerful models. A typical set of equations to describe Neural Networks is provided in [104], and is also listed below for completeness. A single neuron implements a mathematical function given its inputs, to provide an output, as described in equation 2.1 and illustrated in figure 2.1.

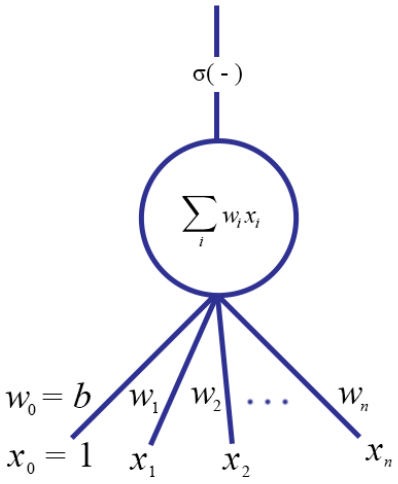


Figure 2.1: A single artificial neuron

$$f(x) = \sigma(\sum_{i=1}^{n} x_i w_i + b) \tag{2.1}$$

In this equation, x_i is the input i, w_i is the weight associated with input i, b is a bias term and σ is a non-linear function. Common non-linear functions are *Rectified Linear Unit* non-linearity, and *Sigmoid* function.

2.2.1 Activation Functions

To go from one layer to the next in a NN, units compute a weighted sum of their inputs from the previous layer and pass the result through a non-linear activation function [61]. There are many possible choices for the non-linear activation functions in a multi-layered network, and the choice of activation functions for the hidden units may often be different from that for the output units. This is a consequence of the fact the hidden and output units perform different roles [7].

At present, the most popular non-linear function is the Rectified Linear Units (ReLU), which is simply the half-wave rectifier $f(z) = max(z, 0)$. In the past decades, neural nets used smoother non-linearities, such as $tanh(z)$ or $1/(1 + exp(-z))$, but ReLU typically learns much faster in networks with many layers, allowing training of a deep supervised network without unsupervised pre-training [61]. A rectified linear unit has been illustrated in figure 2.2

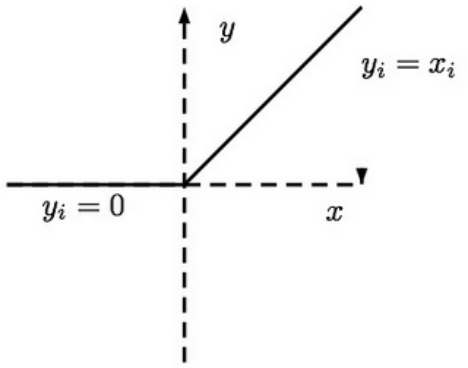


Figure 2.2: A Rectified Linear Unit (ReLU)

The rectifier activation function allows a network to easily obtain sparse representations. For example, after uniform initialization of the weights, around 50% of hidden units continuous output values are real zeros, and this fraction can easily increase with sparsity-including regularization. Apart from being more biologically plausible, sparsity also leads to mathematical advantages. On the other hand, one may hypothesize that the hard saturation at 0 may hurt optimization by blocking gradient back-propagation. However, experimental results done by Glorot et al. suggest that hard zeros can actually help supervised training [36].

2.2.2 Multi-layer Neural Network

Models based on a single neuron, also called Perceptron, have severe limitations. As noted by Preparata and Shamos in [88], a perceptron cannot model data that is not linearly separable, such as modeling a simple XOR operator. On the other hand, as shown by Hornik et al. [50], Multi-layer Neural Networks are universal approximators, that is: they can approximate any measurable function to any desired degree of accuracy.

A neural network of three or above layers of neurons shapes a *Multi-layer Neural network* where the first layer is composed of the inputs to the neural network, it is followed by one or more *hidden* layers, up to a last layer that contains the outputs of the network. In the simplest configuration, each layer l is fully connected with the adjacent layers $(l-1,\ l+1)$, and produces an output vector y^l given the output vector of the previous layer y $^{l-1}$. The output of a layer is calculated by applying the neuron activation function for all neurons on the layer, as noted in

equation 2.2, where W^l is a matrix of weights assigned to each pair of neurons from layer l and $l-1$, and b^l is a vector of bias terms for each neuron in layer l.

$$y^l = \sigma(W^l y^{l-1} + b^l) \tag{2.2}$$

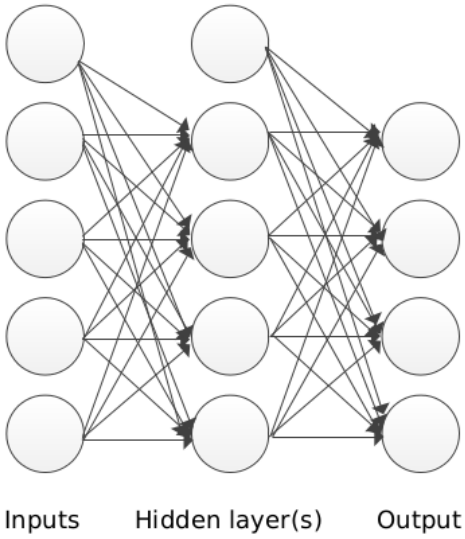


Figure 2.3: A neural network composed of three layers

2.2.2.1 Training Objective

In order to train the model, an error function, also called *loss function*, is defined. This function calculates the error of the model predictions with respect to a dataset. The objective of the training is to minimize the sum (or, equivalently, minimize the mean) of this error function applied to all examples in the dataset.

Commonly used loss functions are the Squared Error function (SE), described in equation 2.3, and the Cross-Entropy error function (CE), described in equation 2.4 (both equations describe the error for a single example in the dataset). As analyzed by Golik et al. [37], it can be shown that the true posterior probability is a global minimum for both functions, and therefore a Neural Network can be trained by minimizing either.

$$E = \frac{1}{2}\sum_n (\hat{y_n}^l - y_n)^2 \tag{2.3}$$

$$E = -\sum_n (y_n \log \hat{y_c}^l)^2 \tag{2.4}$$

In these equations, $\hat{y_n}^l$ is the prediction of the model on the last layer for the unit c, and y_n is the corresponding true label.

For regression strategy, *Euclidean loss* (Sum-of-Squares) function is commonly applied to the top of the outputs to compute Euclidean loss (L2 norm) for real-valued regression tasks.

2.2.2.2 Back Propagation

The error function can be minimized using Gradient-Based Learning. This strategy consists in taking partial derivatives of the error function with respect to the model parameters, and using these derivatives to iteratively update the parameters [63]. Efficient learning algorithms can be used for this purpose if these gradients (partial derivatives) can be computed analytically.

Backward Propagation of errors (BP) was the main advance in the 1980's that led to an explosion of interest in NNs. BP is one of the most commonly used methods for training NNs. The idea behind BP is that it repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the network and the desired one. As a result of the weight adjustments, internal (*hidden*) units come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units [104].

Specifically, BP computes how fast the error changes as we adjust a hidden activity by using error derivatives with respect to hidden activities. Since each hidden activity can have a notable effect on many output units and consequently on the error, a combination of these effects must be considered. This aggregation is done efficiently which allows us to compute error derivatives for all the hidden units quickly at the same time.

Computing the error derivatives for the hidden activities, it would be easy to get the error derivatives for the weights going into a hidden unit which is the key to be able to learn efficiently.

2.2.3 Training Algorithm

Training the network consists in minimizing the error function (by updating the weights and biases), often using the *Stochastic Gradient Descent* (SGD) algorithm.

Stochastic gradient descent has often been proposed to minimize the *empirical risk*[1] (training set performance measure) using *gradient descent* (GD). The standard gradient descent algorithm updates the parameter θ of the objective *J(θ)* with *learning rate* α[2] as:

$$\theta = \theta - \alpha \nabla_\theta E[(J(\theta)] \tag{2.5}$$

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set (Empirical Risk Minimization (ERM)). Stochastic gradient descent simply does away the expectation in the update and computes the gradient of the parameters using only a single or a few training examples. The new update is given by:

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \tag{2.6}$$

with a pair $(x^{(i)}, y^{(i)})$ from the training set [80].

The SGD algorithm, as defined in [23], is described below at a high-level. The inputs are: W –the weights and biases of the network, (X, y) – the dataset, batch size – the number of training examples that are used in each iteration, and α, the learning rate. For notation simplicity, all the model parameters are represented in W. In practice, each layer usually defines a 2-dimensional weight matrix and a 1-dimensional bias vector.

In summary, SGD iterates over mini-batches of the dataset, performing forward-propagation followed by a back-propagation to calculate the error derivatives with respect to the parameters.

[1] ERM as a principle in statistical learning theory sets theoretical boundaries on the learning algorithms.

[2] The learning rate α is the weight of the negative gradient (will be described fully in section 5.2.2.2).

Algorithm 1 Stochastic Gradient Descent

Require: W, X, y, $batch\_size$, α

$W \longleftarrow$ random values

repeat

x_batch, $y\_batch \longleftarrow$ next $batch\_size$ examples in (X, y)

$network\_state \longleftarrow$ **ForwardProp**(W, $batch\_size$)

$W_{grad} \longleftarrow$ **BackProp**($network\_state$, y_batch)

$\Delta W \longleftarrow -\alpha W_{grad}$

$W \longleftarrow W + \Delta W$

until Convergence_Criteria()

The weights are updated using these derivatives and a new mini-batch is used. This procedure is repeated until a convergence criterion is reached. Common convergence criteria are: a maximum number of epochs (number of times that the whole training set was used); a desired value for the cost function is reached; or training until the cost function shows no improvement in a number of iterations.

Generally, each parameter update in SGD is computed with respect to a few training examples or a mini-batch as opposed to a single example. The reasons for this are threefold [80]:

1. The variance in the parameter update is reduced, potentially leading to a more stable convergence.

2. It allows the computation to take advantage of highly optimized matrix operations that should be used in a well vectorized computation of the cost and gradient. A typical mini-batch size is 256, although the optimal size of the mini-batch can vary for different applications and architectures.

3. One final but important point regarding SGD is the order in which we present the data to the algorithm. If the data is given in some meaningful order, this can bias the gradient and lead to poor convergence. Generally, a good method to avoid this is to randomly shuffle the data prior to each epoch of training.

2.3 Deep Neural Networks

As mentioned, standard neural network (NN) consists of many simple neurons, each producing a sequence of real-valued activations. The depth of an architecture refers to the number of concatenated non-linear operations that are composed on the network. While many of the early successful applications of neural networks used shallow architectures (up to 3 levels), the mammal brain is organized in a deep architecture. The brain appears to process information through multiple stages, which is particularly clear in the primate visual system [3].

Moreover, theoretical results strongly suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one needs deep architectures. Deep Neural Networks (DNN) are composed of multiple levels of non-linear operations, such as those present in neural nets with many hidden layers [3]. Figure 2.4 demonstrates a deep neural network.

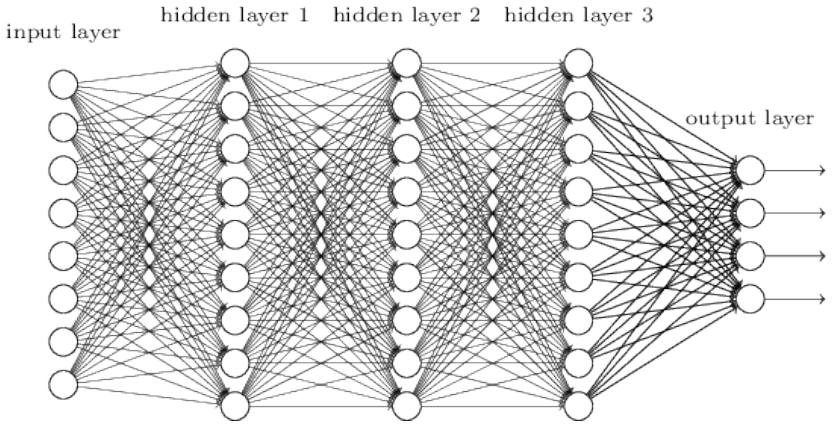


Figure 2.4: A deep neural network with three hidden layers

Deep Neural Networks have been investigated for decades, but training deep networks consistently yielded poor results, until very recently. It was observed in many experiments that deep networks are harder to train than shallow networks, and that training deep networks often get stuck in apparent local minima (or plateaus) when starting with random initialization of the network parameters.

It was discovered, however, that better results could be achieved by pre-training each layer with an unsupervised learning algorithm [47]. In 2006, Hinton et al. obtained good results using Restricted Boltzmann Machines (RBM) (a generative model) to perform unsupervised training of the layers [47]. The goal of this training was to obtain a model that could capture patterns in the data, similarly to a feature descriptor, not using the dataset labels. The weights learned by this unsupervised training were then used to initialize neural networks.

Similar results were reported using auto-encoders for training each layer in [6]. These experiments identified that layers could be pre-trained one at a time, in a greedy layer-wise format. After the layers are pre-trained, the learned weights are used to initialize the neural network, and then the standard back-propagation algorithm (explained in Section 2.2.2.2) is used for fine-tuning the network. The advantage of unsupervised pre-training was demonstrated in statistical comparisons [6], until recently when deep neural networks trained in a full supervised manner (using labeled data) started to register similar results in some tasks like object recognition. Ciresan et al. [17] demonstrated that properly trained deep neural networks (under full supervision) are able to achieve top results in many tasks, although not denying that pre-training might help, especially in cases were little data is available for training, or when there are massive amounts of unlabeled data. On the task of image classification, the best published results use a particular type of architecture called Convolutional Neural Network, which is described in below.

2.4 Convolutional Neural Networks

Convolutional Neural Networks are a specialized kind of neural network for processing data that has a known grid-like topology such as image data which can be thought of as a 2D grid of pixels. CNNs are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [38]. This type of network was used to obtain state-of-the-art results in the CIFAR-10 object recognition task [17], and more recently in more challenging tasks such as the ImageNet Large Scale Visual Recognition Challenge [92]. For both tasks, the training process benefited from the significant speed-ups on processing using modern GPUs (Graphical Processing Units), which are well suited for the implementation of convolutional networks.

Essentially, CNNs combine three architectural ideas to ensure some degree of shift and distortion invariance of local receptive fields, shared weights (or weight replication), and, sometimes, spatial or temporal sub-sampling [66]. We are going to explain these basic ideas of CNN while introducing the main components that compose the main body of any CNN architecture.

2.4.1 Convolutional Layer

Each unit of a convolutional layer receives inputs from a set of units located in a small neighborhood in the previous layer. With local receptive fields, neurons can extract elementary visual features such as oriented edges, end-points and corners. These features are then combined by the higher layers [66]. In addition, elementary feature detectors that are useful on one part of the image are likely to be useful across the entire image. This knowledge can be applied by forcing a set of units, whose receptive fields are located at different places on the image, to have identical weight vectors [104]. The outputs of such a set of neurons constitutes a *feature map*. At each position, different types of units in various feature maps compute distinct types of features. Figure 2.5 presents a convolutional layer functionality during forward propagation.

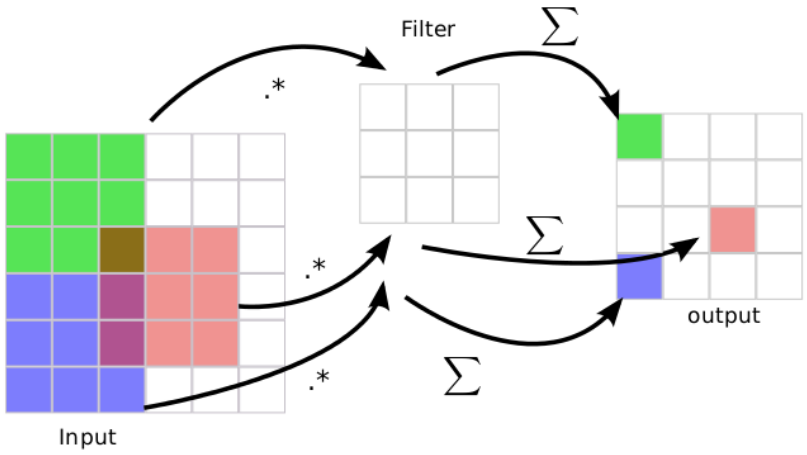


Figure 2.5: The forward propagation phase of a convolutional layer

A sequential implementation of this process, for each feature map, would be to scan the input image with a single neuron that has a local receptive field, and to store the states of this neuron at corresponding locations in the feature map [66].

Units in a feature map are constrained to perform the same operation on different parts ot

the image. A convolutional layer is usually composed of several feature maps (with different weight vectors), so that multiple features can be extracted at each location.
More formally, the definition of a 2D convolution is formulated in equation 2.7. It is the application of a discrete convolution of the inputs y^{l-1} with a filter w^l , adding a bias b^l, followed by the application of a non-linear function σ:

$$y^l_{rc} = \sigma \sum_{i=1}^{F_r} \sum_{j=1}^{F_c} y^{l-1}_{(r+i-1)(c+j-1)} (w^l_{ij} + b^l) \tag{2.7}$$

In this equation, y^l_{rc} is the output unit at {r, c}, F_r and F_c are the number of rows and columns in the 2D filter, w^l_{ij} is the value of the filter at position {i, j}, $y^{l-1}_{(r+i-1)(c+j-1)}$ is the value of input to this layer, at position {r + i - 1, c + j - 1}, and b^l is the bias term.

The equation above is defined for all possible applications of the filter, that is, for $r \in \{1, ..., X_r - F_r + 1\}$ and $c \in \{1, ..., X_c - F_c + 1\}$, where X_r and X_c are the number of rows and columns in the input to this layer. The convolutional layer can either apply the filters for all possible inputs, or use a different strategy. Mainly to reduce computation time, instead of applying the filter for all possible {r, c} pairs, only the pairs with distances are used, which is called the *stride*. A stride, s = 1, is equivalent to apply the convolution for all possible pairs, as defined above.
The inspiration for convolutional layers originated from models of the mammal visual system. Modern research in the physiology of the visual system found it consistent with the processing mechanism of convolutional neural networks, at least for quick recognition of objects [3]. Although being a biological plausible model, the theoretical reasons for the success of convolutional networks are not yet fully understood. One hypothesis is that the small fan-in of the neurons (i.e. the number or input connections to the neurons) helps the derivatives to propagate through many layers - instead of being diffused in a large number of input neurons [27].

2.4.1.1 Weight Sharing

Weight sharing refers to having several connections controlled by a single parameter (weight). Weight sharing can be interpreted as imposing equality constraints among the connection strengths. An interesting feature of weight sharing is that it can be implemented with very little computational overhead [64]. The weight sharing technique has an interesting side effect of reducing the number of free parameters, thereby the capacity of the machine and improving its generalization ability [66].

2.4.2 Pooling/Sub-sampling Layer

Once a feature is detected, its' exact position becomes less important as long as its' approximate position relative to other features is preserved. Furthermore, as the dimensionality of applying a filter is equal to the input dimensionality, we would not be gaining any translation invariance with these additional filters. We would be stuck doing pixel-wise analysis on increasingly abstract features. In order to solve this problem, a *sub-sampling* layer is introduced.

Sub-sampling, or down-sampling, refers to reducing the overall size of a signal. In many cases, such as audio compression for music files, sub-sampling is done simply for size reduction. But in the domain of 2D filter outputs, sub-sampling can also be thought of as reducing the sensitivity of the output to shifts and distortions.

One of the most applied sub-sampling methods used in [65], is known as *max-pooling*. Equation 2.8 presents te formulation of a max-pooling layer.

$$y_{rc}^{l} = \max_{i,j \in \{0,1,\ldots,m\}} y_{(r+i-1)(c+j-1)}^{l-1} \tag{2.8}$$

In this equation, y_{rc}^{l} is the output for index r, c, m is the size of the pooling area, and $y_{(r+i-1)(c+j-1)}^{l-1}$ is the value of the input at the position {r + i - 1, c + j - 1}.

This involves splitting up the matrix of filter outputs into small non-overlapping grids (the larger the grid, the greater the signal reduction), and taking the maximum value in each grid as the value in the reduced matrix. A schematic of max-pooling is depicted in figure 2.6. Similarly for the convolutional layer described above, instead of generating all possible pairs of {i, j}, a *stride, s,* can be used. In particular, a stride s = 1 is equivalent to using all possible pooling windows, and a stride s = m is equivalent of using all non-overlapping pooling windows.

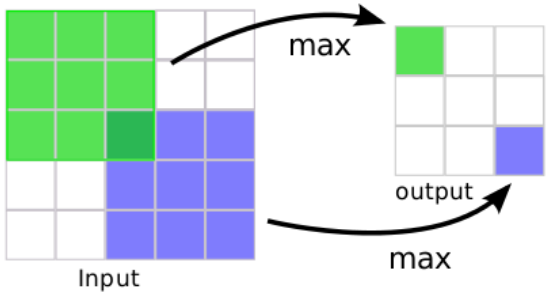


Figure 2.6: The forward propagation phase of a pooling layer with stride equal to 1.

Scherer et al. [93] evaluated different pooling architectures, and note that the max-pooling layers obtained the best results. Pooling layers add robustness to the model, providing a small degree of translation invariance, since the unit activates independently on where the image feature is located within the pooling window [4]. Empirically, pooling has demonstrated to contribute to improved classification accuracy for object recognition [62].

2.4.3 Local Response Normalization

ReLU have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, we still find that *local response normalization*(LRN) scheme aids generalization. This sort of response normalization implements a form of lateral inhibition (the capacity of an excited neuron to reduce the activity of its neighbors) inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels [58]. Figure 2.7 illustrates local response normalization in a CNN.

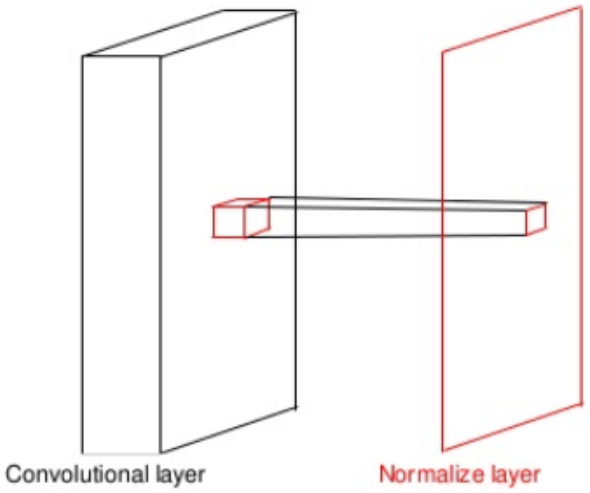


Figure 2.7: Local response normalization in a CNN after the convolution layer.

More formally, the activity $a^i_{x,y}$ of a neuron in bank i at position *(x,y)* in the topographic organization[3] is divided by:

$$\left(1 + \alpha \sum_{j=i-N/2}^{i+N/2} (a^j_{x,y})^2\right) \tag{2.9}$$

where the sum runs over N " adjacent " banks of neurons at the same position in the topographic organization. The ordering of the bank is arbitrary and determined before training begins. The constants N, α and β are hyper-parameters whose values are determined using a validation set.

In other words, we can think of it as helping sharpening the response. Instead of carrying multiple ambiguous representation of a patch, it pushes the network to commit more towards a specific representation, freeing resources to analyze it better.

This scheme bears some resemblance to the local contrast normalization scheme proposed by Jarrett et al. in [54] which has led to error rate reduction in [58] and [48].

2.4.4 Fully Connected/Inner Product Layer

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via *fully connected layers* (FC/IP). A fully connected layer takes all neurons in the previous layer (be it fully connected, pooling, or convolutional) and connects it to every single neuron it has. Fully connected layers are not spatially located anymore (you can visualize them as one-dimensional), so there can be no convolutional layers after a fully connected layer.

[3]The cerebral cortex of mammals primarily consists of a set of brain areas organized as topographic maps. These maps contain systematic two-dimensional representations of features relevant to sensory, motor, and/or associative processing, such as retinal position, sound frequency, line orientation, or sensory or motor motion direction [83].

3 State of the Art Review

In this chapter, we present a review of state-of-the-arts regarding synthetic data generation for deep neural networks to be applied for learning to count tasks. We introduce a summary of cutting-edge research studies from two aspects. First we succinctly review the background of generating synthetic data for different problems, and then a short history of feature detection algorithms will be alluded.

3.1 Synthetic Data Generation

The main purpose of generating synthetic datasets has been to protect the privacy and confidentiality of the actual data [108, 85], since it does not hold any personal information and cannot be traced back by any individual. Problems such as fraud detection [85], or health care [108], are normally tackled by the use of synthetic data. These datasets might seem to be "made up" data, however, there are certain algorithms and generators that are designed to create realistic data [108, 2].

In addition, synthetic data are often used to meet specific needs or certain conditions that may not be found in the original, real data. This can be useful when designing any type of system because the synthetic data are used as a simulation or as a theoretical value, situation, etc. For example, in [49], the authors introduce a method for detecting intrusions where the experiment is done using synthetic data. This data is a representation of the authentic data and may include intrusion instances that are not found in the authentic data. The synthetic data allows the software to recognize these situations and react accordingly.

Furthermore, in computer vision, the use of synthetic images has a longstanding history, as in 2000, Cappelli et al. [9] presents an approach to synthetic fingerprint generation on the basis of some mathematical models that describe the main features of real finger prints. The process of generating finger prints follows a simple course (as shown in Figure 3.1). First, a fingerprint shape, a directional map and a density map are generated separately; then these three features are combined to obtain a fingerprint pattern, which is finally more realistic by adding noise.

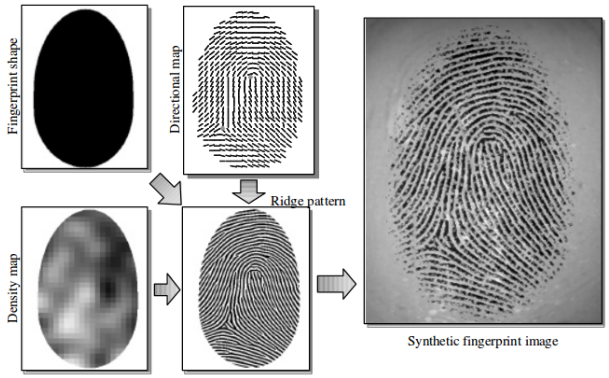


Figure 3.1: The basic idea of the synthetic fingerprint generation approach [9].

More recently, after the success of deep convolutional neural networks in various vision tasks concerning object detection or classification, generation and use of synthetic datasets has been frequently considered. As a viable alternative to complement the real data, synthetically generated training sets can remarkably improve the performance of DCNNs when the real data is limited [24, 51, 53, 33]. For example, in [24], synthetic images are generated to be fed to a DCNN in order to learn how to detect company logo in the absence of a large training set. In order to generate the images, first they select a random perspective transform and warp both image and mask accordingly. Then, they modify the color of the warped image to provide a larger visual variety. Later, the logo is randomly blurred by convolution with a Gaussian kernel (with different σ values) to simulate different scales. Finally, the background is removed of the logo and the warped and color-modified logo into a new image which contains no logo. This process is shown in figure 3.2.

Figure 3.2: An example of synthetically generated images in [24]. Starting from a base image, a random perspective transform is applied, followed by gaussian blur (not shown), color augmentation (exaggerated for viewing purposes) and background replacement.

Moreover, As one of the most recent approaches, Seguí et al. in [94] proposed synthetic data generation to counter lack of data issue for learning to count the number of objects in images using deep convolutional neural networks. In their work, they took advantage of existent unlabeled and labeled datasets to generate realistic synthetic images representative of the actual images. The authors introduce two counting problems, counting number of even-digits in images, and counting the amount of pedestrians in a walkway. In the former problem, they incorporated real labeled hand-written digits to make images with up to five digits in the image. Figure 3.3 depicts an example of generated images in [94].

For the latter one, they used an unlabeled set of images (video frames) of pedestrians in a walkway. The creation of images is as follows: First, a simple background subtraction technique

(a) Synthetic image for even digits counting problem.

(b) Synthetic image for counting the number of pedestrians.

Figure 3.3: An example of synthetically generated images by Seguí et al. [94].

is used. Then a Gaussian filter is applied across the backgrounds to smooth out motion effects. Afterwards, the candidate pedestrians are extracted subtracting the background. Background noise and imperfect shapes are handled using mathematical morphology operators. At last, new artificial images are synthesized by means of a composition of backgrounds and pedestrians' sub-images. Each image contains up to 25 pedestrians [94]. After generating these synthetic datasets, they used deep convolutional neural networks to learn the object features in order to count the number of objects present in the images.

3.2 Feature Detection Algorithms

Counting the number of an object of interest in an image can be approached from two different perspectives: either training an object detector, or training an object counter [94]. In the field of object detection, numerous works have been previously proposed [57, 73]. Most of these research works follow a taxonomy which consists of three paradigms underneath to count the objects [11]:

1. Object detection methods, which were based on boosting motion and appearance features, Bayesian model-based segmentation, integrated top-down and bottom-up processing.

2. Visual feature trajectory clustering. This paradigm counts objects by identifying and tracking visuals over a time period. Feature trajectories with coherent motion are then clustered and the number of clusters is the estimate of the number of objects.

3. Feature-based regression. These methods usually work by first, subtracting the background, second, measuring various features of the foreground pixels such as total area, edge count, or texture; and finally estimating the objects' density or objects' count by a regression function, e.g. linear, piece-wise linear, or neural networks.

   In recent years, feature-based regression has also been applied to outdoor scenes. For example, Kong et al. [57] apply neural networks to the histograms of foreground segment areas and edge orientations. Dong et al. [22] estimate the number of people in each foreground segment by matching its shape to a database containing the silhouettes of possible people configurations, but that would be only applicable when the number of people in each segment is small (empirically, less than 6) [11].

By reason of the fact that almost all the above algorithms detect the whole objects in an image (e.g. whole pedestrians), these methods have moderate performance in very noisy or crowded images with significant occlusion, Wu and Nevatia [106], introduced methods to address this issue. Wu and Nevatia [106] proposed *edgelet features* (an edgelet is a short segment of line or curve) as new type of silhouette oriented features to deal with the problem of detecting individuals in crowded still images.

As a similar line of work in the course of object counting and more specifically crowd counting, in [89, 68], different object tracking approaches were taken to detect and count moving objects in the scene. However, the deployment of these vision surveillance technologies are invariably met with skepticism by society at large, given the perception that they could be used to infringe individuals' privacy rights (through tracking individuals as objects in order to count their multiplicity). While a number of methods that do not require explicit detection or tracking have been previously proposed [57, 73, 22], they have not fully established the viability of the privacy-preserving approach [11]. The tension of privacy-preserving is common in all areas of data-mining.

In order to tackle privacy preserving issue, Chan et al. [11] presented a novel approach with no explicit object segmentation or tracking to estimate the number of people moving in each direction (towards and away from camera) in a privacy-preserving manner. An outline of the crowd counting system appears in figure 3.4:

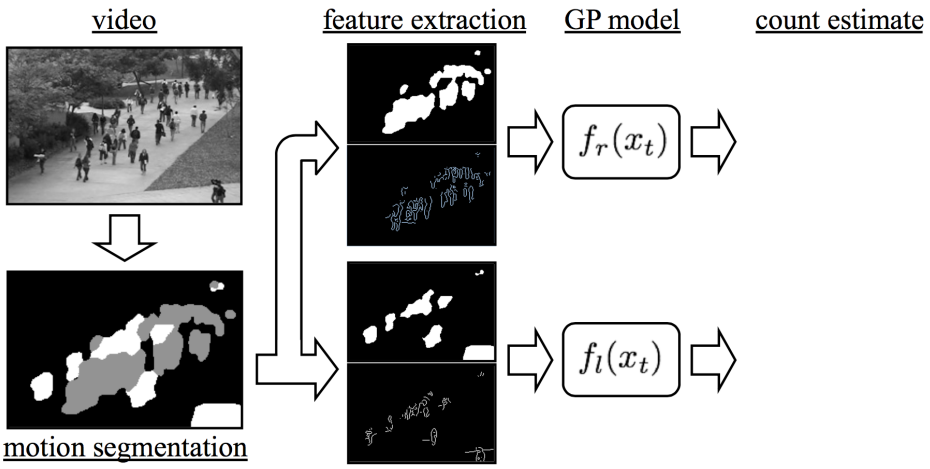


Figure 3.4: Crowd counting system: the scene is segmented into crowds with different motions. Normalized features that account for perspective are extracted from each segment, and the crowd count for each segment is estimated with a Gaussian process [11].

Chan et al. used a mixture of *dynamic textures*[1] to divide the video frames into regions containing moving pedestrians in different directions. When adopting mixture of dynamic textures, the video is represented as collection of spatio-temporal patches which are modeled as independent samples from mixture of dynamic models. The mixture model is learned through Expectation-Maximization(EM) algorithm. Video locations are then scanned sequentially, a patch is extracted at each location, and assigned to the mixture component of largest posterior

[1]Dynamic textures are sequences of images that exhibit some form of temporal stationarity, such as waves, steam, and foliage.

probability. The location is declared to belong to the segmentation region associated with that component [11]. The resulting segmentations of their work are illustrated in figure 3.5:

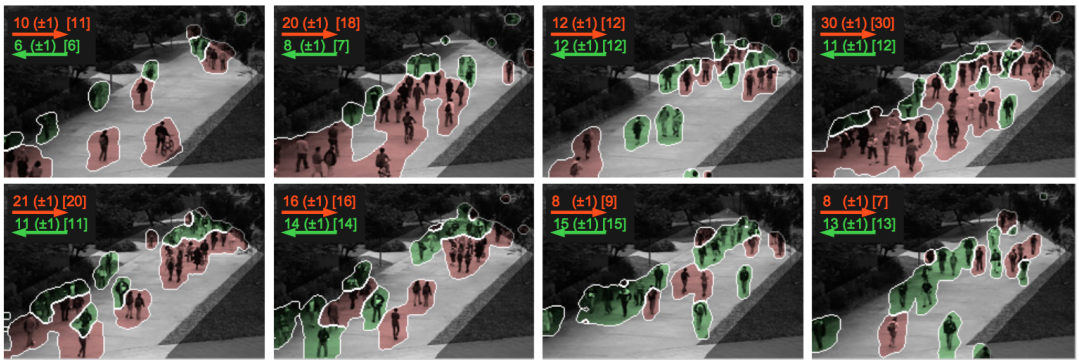


Figure 3.5: Crowd counting results: The red and green segments are the "away" and "towards" crowds. The estimated crowd count for each segment is in the top-left, with the (rounded standard-deviation of the GP) and the ground-truth. The Region Of Interest (the area in the walkway in which the pedestrians are counted and labeled) is also highlighted [11].

After segmenting the moving pedestrians, extracting features from the video segments is done at three phases:

- Segment features to capture segment shape and size. Features such as area, perimeter, perimeter edge orientation and perimeter-area ratio.
- Internal edge features contained in a crowd segment are a strong about the number of pedestrians in the segment [57]. For instance, total edge pixels and edge orientation.
- Texture features which are based on gray-level co-occurrence matrix[2] (see [45] for more details) were applied for image patches classification into 5 classes of crowd density in [73]. Due to the task similarity, Chan et al. adopted a similar set of measurements for counting the number of crowd in each segment, and computed texture properties like homogeneity, energy and entropy.

Having segments' features extracted, a Gaussian Process[3] (GP) was used to regress feature vectors to the number of people per segment. The GP defines a distribution over functions, which is "pinned down" at the training points [11]. Since the classes of function that GP can model is directly dependent on the selected kernel function, they combined the linear and the squared-exponential (RBF) (see [96] for more details) kernels, *i.e.*

$$k(x_p, x_q) = \alpha_1(x_p^T x_q + 1) + \alpha_2 e^{\frac{-\|x_p - x_q\|^2}{\alpha_3}} + \alpha_4 \delta(p, q) \tag{3.1}$$

The linear component of the kernel captures the dominant trend of many features which is linear (*e.g.* segment area), while the RBF component models local non-linearities that arise from a

[2] A matrix over an image to define the distribution of values which co-occur in the existing offset.

[3] Gaussian Process is a statistical distribution in which the occurrence of observations happen in a continuous domain such as time or space. The distribution of a GP is a distribution over functions in a continuous domain.

variety of factors, including occlusion, segmentation errors and pedestrian configuration (*e.g.* spacing within a segment) [11].

For this experiment, they collected an hour of video from a stationary digital camera. The first 2000 frames of the videos were annotated as ground-truth. Moreover, a region-of-interest (ROI) was defined on the walkway (see figure 3.5), and the traveling directions (away from or towards the camera) and visible center of each pedestrian was annotated. Then the video was split into a training set, for learning the GP, and a test set for validation. The training set contains 800 frames, between frame 600 and 1399, with the remaining 1200 frames held out for testing. This dataset is available to the vision community [11].

Although they obtained reasonable results in a privacy-preserving manner, their worked required exhaustive data annotations along with hand-crafting highly specialized image features that were dependent on the object class (pedestrians).

In order to save annotation efforts, different techniques were used to count objects. Multiple Instance Learning (MIL) [29] is a variation of supervised learning in which instances come in bags. These bags contain multiple samples. A bag is labeled positive if there is at least one example with the concept of interest, or labeled negative otherwise. The positive bag can be regarded as a set of attracting instances and the negative one as a set of repulsive instances. In large-scale computer vision, this approach is frequently found under the name of *weakly supervised learning*. There are different definitions for the term "weakly" in the literature. For instance, in [18], it is a surrogate for the concept of noisy labels such as labels provided by different supervisors with distinct quality. However, in [91], it is described for indicating imperfect annotation or even in [103] for specifying only the presence of an object in an image.

Early works used weakly supervised learning in an instantiation of the MIL framework for inferring difficult to describe classes such as in [101] where photometric, geometric, and topological features are recognized. More recently, several works, such as [81], explore the capacity of this technique for simultaneous localization and recognition. Another work using MIL framework was count-based multiple instance learning [29]. In count-based MIL the positive bag is composed of instances where the concept appears within the range of an interval. For example, the positive bag may contain images with 5 to 10 appearances of pedestrians. A major drawback of MIL framework is that even in count-based MIL the problem is casted as a binary task and they would not be applicable in projects where the exact number of objects in the image important.

Furthermore, another approach to reduce the annotation tasks is done in [28], where the labeling process is decreased to dotting(pointing) and the counting process is addressed as image density estimation problem.

Recently, with the success of CNNs in different vision tasks, object detection systems based on deep CNN have made groundbreaking advances on several object detection problems [25, 35] which suggests the use of this technique to learn to count objects. Several advantages can be foreseen from this application, being the most important that of learning image features from samples instead of hand-crafting highly specialized image features that are dependent on the object class [94]. Moreover, CNN have shown their capacity of knowledge transfer for a number of tasks or the ability of simultaneously performing different tasks even when trained for only one [111].

Following this line of work, Seguí et al. in [94] proposed a novel approach for counting objects' representations using deep object features. In their work, objects' features are learned by a deep counting convolutional neural network and are used to understand the underlying representation. Their proposal lies in the middle of weakly supervised learning and fully supervised learning [77].

It is similar to weakly supervised learning because the location of the concept of interest is not given. Whereas, unlike fully supervised learning in which the object boundary or bounding box is given to the learning process, in their proposed architecture, only the multiplicity of the object is provided [94].

To this end, they defined a counting problem for even digits and demonstrated that the internal representation of the network is able to classify digits in spite of the fact that during training, no direct supervision was provided. Moreover, they present preliminary results about a deep network that is able to count the number of pedestrians in a scene [94]. Figure 3.6 illustrates their proposal at a glance in the case of representing hand-written digits:

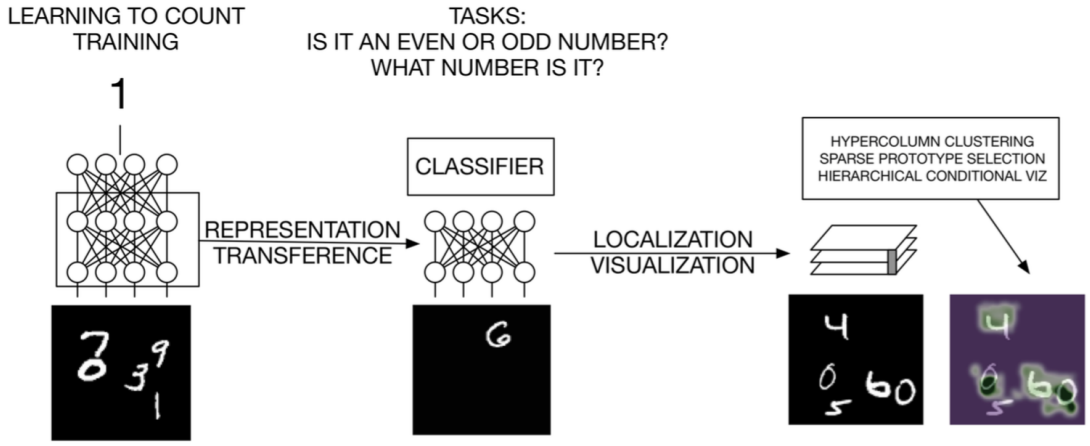


Figure 3.6: Learning to count hand-written digits problem in which the features of a CNN that has been trained to count digits can be readily used for more specific classification problems and even to localize digits in an image [94].

In [94], the main hypothesis is that the number of occurrence of objects in an image provide strong presentational information due to their possible discriminate appearance for a feature learning process to exploit. In order to verify this hypothesis, for both experiments, they considered networks of two or more convolutional layers (since CNNs instinctively handle feature learning [62]) consisting of convolutional filters, ReLU non-linearities, max-pooling layers and normalization layer, followed by one or more fully connected layers (regarding the impressive classification performance on different benchmark problems [58]) [94].

For learning to count in the hand-written digits domain, they synthetically created a set of one million images of size 100×100 including random digits from the MNIST database with maximum 5 digit per image and with no overlapping in the images. Obtaining accuracy of 93.8% on the base network, along with results attained from training a support vector machine (SVM) with the representations learned on different layers of the network, validates this hypothesis that counting process can be considered as a surrogate to potentially extract or infer interesting object descriptors [94].

Additionally, for learning to count the number of pedestrians in a scene, they used UCSD pedestrian database [11]. Once again, Seguí et al. [94] created a set of 200.000 synthetic images each containing up to 25 pedestrians. The results in this scenario are encouraging and reinforce the feasibility of the proposal in front of counting problems. However, still there are some deficiencies that should be obviated. For instance, how would a model trained by synthetic dataset perform on real dataset and in a real world problem? Or would the model still be able

to learn object representations in scaled-up and more complex scenarios?

4 Proposal

This master thesis presents an analysis on the synthetic data generation procedure as a feasible alternative for lacking or missing real data, as well as proposes an application of deep convolutional neural networks for the task of counting objects in the images. The developed systems will address some of the issues common to other object detection algorithms. These issues include:

- Obtaining poor results due to lack of data.
- Painstaking hand-crafted image features which are highly dependent on the object class.
- Scrupulous data annotation for manifold data.
- Establishing the viability of a privacy-preserving approach.
- Being prone to error in noisy or crowded scenes with a noticeable occlusion.

In addition, the state-of-the-art [94] technology lacks applicability for real-world counting problems due to being tested only on synthetic dataset.

The novelty of our approach compared to the state-of-the-art is that we present highly-realistic synthetic data generation algorithms for training DCNN. Consequently, by the use of these realistic datasets, we hypothesize that features learned by DCNN with synthetic dataset, are representative enough to count the number of object of interest in a real dataset. We tackle this task as a regression problem. To the best of our knowledge, the proposed work would be the first one in which a counting system trained with synthetic images is able to be incorporated in real-world similar counting problems.

Henceforth, in the rest of this chapter, we justify both our synthetic data generation process and our proposed methodology along with a comparison to state-of-the-art from different aspects including method selection and architecture.

4.1 Datasets

In this section, starting with a short review of data for DCNN in computer vision, we express a succinct reasoning on the benefit of synthetically generated datasets we used in our experiments. Later on, in the implementation section, the specifications and preprocessing phases regarding each dataset will be described in details.

In spite of remarkable performance of DCNN in some simple recognition benchmarks [15, 16], until recently the models' performances were poor at more complex datasets [40] due to lack of labeled input samples to train the network parameters with. It was with the creation of ImageNet

[19] and GPU implementation [58] (50× speedup over CPU) that efficiency of deep CNN in vision tasks became crystal clear.

However, there are still many areas such as crowd counting in which large training set is either missing or not publicly available due to privacy-preserving policies. In these types of problems, synthetic data generation can be a well-suited surrogate for the missing actual data. Access to unsupervised images and also the existence of powerful image processing tools enable us to create highly realistic images. Hence, in this work we propose two synthetically created datasets related to proposed learning to count problems for the application of deep convolutional neural networks. First learning to count even-digits dataset and then a set of images for counting the number of pedestrians present in the images.

The proposed synthetic datasets facilitate the application of deep convolutional neural networks for problems concerning learning to count the number of object of interest. Moreover, in the case of counting the number of pedestrians, it preserves the individuals' privacy since the data is synthetic and not real (as explained in chapter 3.2, previous crowd counting approaches were somehow infringing individuals' privacy by using object tracking algorithms.). Consequently, the application of DCNNs lead to easier feature detection process and improvement of the system's performance.

4.1.1 Learning to Count Even-digits Dataset

In this study, we introduce a synthetic dataset using MNIST dataset. The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image [59]. In the MNIST dataset, each image contains one hand-written digit of dimensions 28×28. Figure 4.1 shows a selection of MNIST dataset images.

Figure 4.1: A selection of MNIST images of hand-written digits.

For the first empirical experience in our work, we created a large set of images each containing up to 15 MNIST digits. Images are automatically labeled with the number of even digits in each image. Using this dataset, we can examine the digits' representations learned by our deep CNN are not dependent on the specific task we are dealing with. It is done by using our model's learned features for other tasks and analyze the results.

4.1.2 Synthetic Crowd Counting Dataset

As we needed a large dataset to train all the parameters of our model for the second task, counting the number of pedestrians, we synthetically created a large dataset of pedestrians in a walkway with maximum 29 people in each image. Achieving success on crowd counting system using CNN on our synthetic dataset, we save enormous amount of time for data annotation and feature detection.

4.1.3 UCSD Crowd Counting Dataset

However, we need our model to perform well in practice rather than in theoretical experiments. Therefore, we used UCSD crowd counting dataset [11] to validate the performance of our model on it and also make a comparison with their system in [11] to see if we can obtain better results while decreasing feature extraction and labeling efforts. In addition, if we succeed to attain reasonable results on the real UCSD dataset, we have overcome one of the main restrictions of deep learning algorithms usage, which is the necessity for the existence of large amount of data.

4.2 Method Selection

For a long time in Computer Vision, there has been a prevailing paradigm in which we have a set of feature descriptors such as SIFT[1], HOG[2], SURF[3], and others that can be extracted from the image with possible higher level feature building (High-level feature detection algorithms are more in tune with the way we detect objects in real life), following by a classifier like Support Vector Machines (SVM)[4]. In fact, for the most part, these features are not learned, but handcrafted by some vision experts. However, they do indeed demonstrate a decent performance[26, 110]. The analysis of these works leads to the question of what we should focus on in order to improve the vision systems accuracy. Should we try to enhance classifiers, should we increase the amount of data or we should rather provide finer features?

Parikh and Zitnick in [82] analyze the role of features by taking some of the quite successful past deformable models[5], and replace some components of them with humans. The author concludes that features are the main factor contributing to superior human performance. Furthermore, a comparison on 39 different learning kernels with various combination of features in [34], shows that the gain they obtained from the learning operators is not as dramatic as the improvement they achieved from the features itself.

Therefore, since the features are providing the biggest impact in these algorithms, if we improve those features, we can acquire better algorithms. The difficulty of feature improvement and high cost of numerous feature computations on each image brought us into the application of deep convolutional neural networks in order to learn the features themselves rather than handcrafting them. We select deep CNNs due to their capacity of knowledge transfer for a number of tasks and also the ability for performing different tasks simultaneously, even when it has been trained for only one task [111]. Moreover, compare to other methods, DCNN benefit from the powerful speed-ups on GPU which is fit for implementation of DCNN.

[1] Scale Invariant Feature Transform.

[2] Histograms of Oriented Gradients.

[3] Speeded Up Robust Features.

[4] Supervised learning models used for classification and regression in association with learning algorithms.

[5] Deformable models are curves or surfaces defined within an image domain whose deformations are determined by the displacement of a discrete number of control points along the curve [107].

Furthermore, DCNN was used to achieve state-of-the-art results in CIFAR-10 object recognition task [16], in [92], the training process was done on GPU, and a great success in many other recent vision problems [15]. All these features and advantages of CNN assured us to choose deep convolutional neural networks as our proposed method.

4.3 Architecture

With their long heritage, the origins of CNN trace back to [52] where first simple cells (brain cell) detect local features and afterwards, complex cells pool the outputs of simple cells within a retinotopic[6]neighborhood (see [52] for more detailed explanation). Also, Fukushima in [30, 31] introduced a similar architecture with filtering layers followed by pooling layers. However, the first deep CNN architecture was designed by LeCun et al. [62] who demonstrated that this kind of architecture can perform quite well for vision tasks like digit recognition. A schematic of LeCun et al.'s proposed architecture has been depicted in figure 4.2

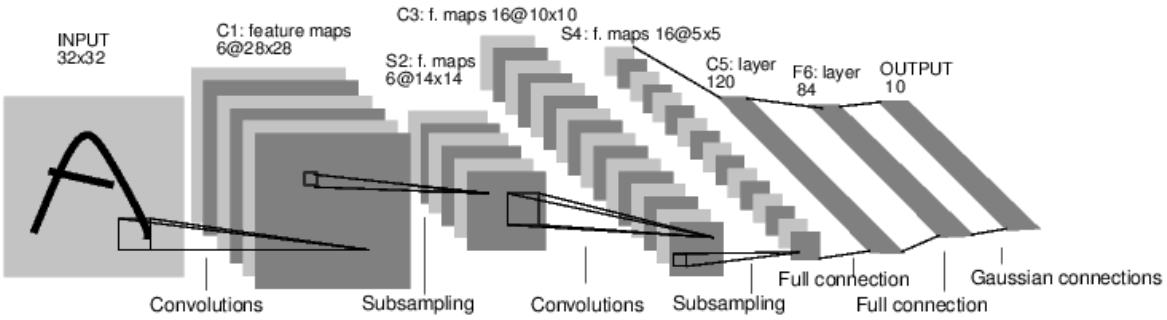


Figure 4.2: Deep CNN proposed by LeCun et al. [62] for MNIST hand-written digits recognition problem.

Following the same structure of classical CNNs, in our architecture, the image pixels are fed to a convolutional layer, where relatively small filters (windows) shift over the image (not necessarily pixel by pixel, as a different stride can be taken) and produce feature maps. Since convolution is a linear operation, in order to make the model non-linear, feature maps are passed to rectified linear units (ReLU) [79] which is applied to each element of the feature map independently.

As discussed in chapter 2.2.1, ReLU is currently favored in the research community given the fact that it fastens the learning process by massively simplifying back propagation, and also avoids saturation issues (when the weighted sum is big, the output of the tanh or *sigmoid* activation functions saturates and the gradient tends to zero. See [44, 1] for more details).

Thereafter, a max-pooling layer takes a special region of the obtained feature maps and takes the maximum pixel value as the strongest structure of that neighborhood (described in section 2.4.2). We chose Max-pooling since it tends to give more discrimination of the features [8]. In our model we use *spatial* pooling. However, depending on the problem we are trying to solve, pooling might be done within feature maps [39]. The main advantage of pooling layers in the architecture can be that it results in invariance to small transformations. In addition to that,

[6]Retinotopy is the mapping of visual input from the retina to neurons, particularly those neurons within the visual stream. For clarity, 'retinotopy' can be replaced with 'retinal mapping', and 'retinotopic' with 'retinally mapped

as we go up in DCNNs, thanks to pooling, each of the units essentially has a larger receptive fields looking back the input so that at the top high-level layers, each unit looks at the entire scene.

To improve the model performance, after each pooling layer in the proposed model, we used local response normalization layer (LRN) to apply a contrast normalization(explained in details in chapter 2.4.3). This type of normalization was introduced and used for the first time in [58]. Empirically, using LRN in the architecture seems to help improving the results [54, 58].

And finally on top of the model, we put fully connected layers in order to do either a classification or regression strategy. In our model, since we are casting the problem as a regression problem, a *Euclidean Loss* layer is stated on top of fully connected layers to project the output as the difference between model prediction and the ground truth. Figure 4.3 illustrates our proposed design of convolutional neural network with just one layer of each type in order to portray the basic structure we are going use in this work.

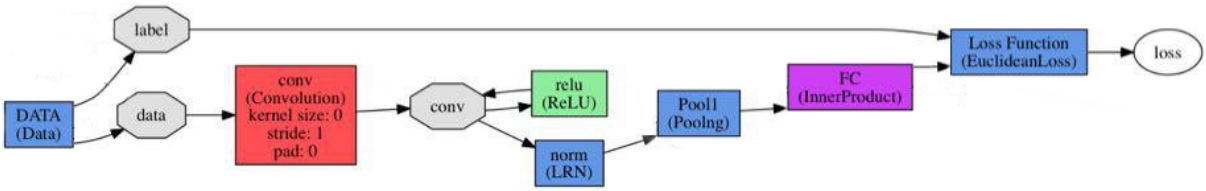


Figure 4.3: The basic design for the proposed DCNNs.

The above explanation describes the reasons behind the selected architecture and how it can help us to overcome the deficiencies of previous related works. However, the most important fact regarding DCNNs' capability to learn features is the depth and settings of the architecture which will be attentively addressed later on (chapter 5.3).

5 Implementation

In this chapter, we provide a detailed implementation of our proposed methodology. We start by giving insight into the introduced algorithms for generating the synthetic datasets to train and test our models. Then we present the software platform we incorporated to shape and design our architectures. Finally, we demonstrate our networks' architecture and settings in detail.

5.1 The Datasets

Here we delve into the data processing part of this work by introducing three different datasets we created or applied for our empirical experiments. To that end, we provide a detailed explanation for each step of synthetic data generation process along with approaches and methods used to improve each dataset.

5.1.1 Even-odd Digits Dataset

Our Even-odd handwritten dataset contains images of size 100×100. Each image is filled with from 0 up to 15 randomly selected digits from MNIST hand-written digits dataset. The process of creating images follows the steps in below:

1. MNIST Digits are resized to 18×18 pixels.
2. Up to 15 digits are randomly put in images of size 100×100 pixels.
3. The images are created with controlled overlapping by ensuring that two different numbers are 18 pixels away from each other, i.e. the distance between two digits centers is larger than 18 pixels.
4. Images are labeled with the number of even digits present in each image.
5. The images are created uniformly, meaning that for example, the number of images containing 0 even digits is equal to the number of images containing 15 even digits.

This dataset has in total 1 million images: 800,000 images for training set and 200,000 as the test. Figure 5.1 illustrates some examples of even-odd digits dataset with different number of even digits in images.

Figure 5.1: An example of even-odd digits images. Form left to right, images contain 0 to 15 even digits.

5.1.2 Synthetic Pedestrians Dataset

Learning features using deep architectures requires a large amount of data. More importantly, for a fully supervised learning, this data should be annotated. Lack of data or its' high annotation cost prohibit the usage of deep learning methods for many problems including crowd counting.

However, in order to lower this cost, in our research, we decided to create a synthetic dataset of pedestrians in a walkway. To do that, we used UCSD unlabeled Anomaly detection dataset of pedestrians collected by Chan et al. and used in [12, 72, 70]. UCSD Anomaly detection dataset contains clips of groups of people walking towards and away from the camera, and consists of 34 training video samples and 36 testing video samples. Each video has 200 frames of each 238×158 pixels. Figure 5.2 depicts some images of UCSD Anomaly dataset.

Figure 5.2: Sample images of UCSD Anomaly detection dataset.

In our study, we used all 70 training and testing video samples to generate the synthetic

pedestrians dataset. To thoroughly demonstrate the generation process of our dataset, we divide this section into data generation and data improvement.

5.1.2.1 Data Generation

In our dataset, we constrained each image by having up to 29 pedestrians in the walkway. The process of generating the data includes the following steps:

1. **Background extraction:** Firstly, we simply subtract the background from each video frame. We extract two types of backgrounds: the median of all the backgrounds in each video (in total, 70 different backgrounds), and the median of all median backgrounds. An example of extracted backgrounds is shown below.

Figure 5.3: One background image extracted from UCSD dataset.

2. **Pedestrian extraction:** Subtracting each image from the mean background, we were able to label the connected regions (each individual in case of our images) of the subtraction using morphological labeling methods. Figure 5.4 shows how images look like after background subtraction.

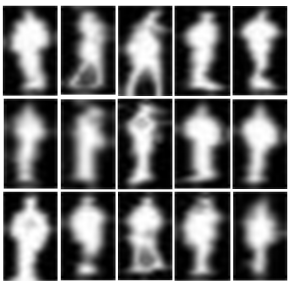
Figure 5.4: Images after background subtraction.

Then, properties of labeled regions are measured and bound-boxed (see [102] for more detailed explanation). Boxes of people are center-based annotated. These labeled boxes shape our initial list of pedestrians with masks of the same size of each box. Figure 5.6 in below demonstrates a selection of extracted pedestrians and pedestrians' masks.

3. **Background generation:** In this step, we tried to make the backgrounds of images as realistic as possible by:

   (a) making a sparse combination of median backgrounds.

(a) Extracted pedestrians.



(b) created pedestrians' masks.

Figure 5.5: Pedestrians and their corresponding masks.

(b) changing the global illumination of the images randomly.

(c) adding some random Gaussian noise to the backgrounds.

As you may notice, in the figure 5.6, the generated background happened to be brighter and noisier than the extracted one.

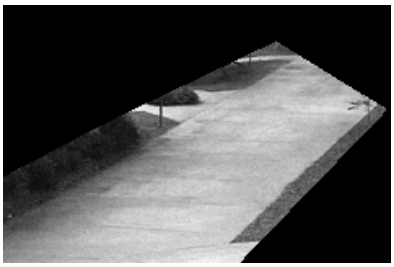
(a) Raw extracted background image.

(b) Generated background after a sparse combination, global illumination change and some noise.

Figure 5.6: A synthetically generated background image

Having backgrounds generated and pedestrians extracted and labeled, backgrounds are selected randomly. Then, for training and comparison purposes, images are masked with a filter of *Region Of Interest* (ROI). The mask and ROI is shown in below (figure 5.7).

(a) ROI mask.



(b) ROI in the image.

Figure 5.7: Applying the mask of region of interest on the background image.

4. **Creating synthetic images:** Afterwards, pedestrians are added to the masked background in a way that the center of each person is placed inside white area of the mask. Finally images are normalized (between 0 and 255) and resized to 158×158 in order to be fed to convolution layers. A selection of created images are depicted in the figure underneath.

Figure 5.8: created synthetic images for counting pedestrians problem.

5.1.2.2 Data Improvement

Although we managed to successfully create synthetic images of people in the street, the generated images were still quite distinguishable from the real dataset. Thus, in order to make images as highly realistic as possible, we improved the dataset as explained underneath:

- **Non-pedestrian objects:** Amongst the extracted boxes of pedestrians, there were some non-pedestrian boxes with objects instead of pedestrians, and yet others with more than one person inside the box. Therefore, we manually removed these outliers. After this edition, we ended with 426 samples of people. A few examples of incorrectly collected or labeled objects are shown in below.

Figure 5.9: A selection of non-human or incorrectly labeled objects. As you may see, there are some images with "half a person" and some others with more than just one person in the image.

- **Lack of pedestrians:** For the sake of generalization, we needed a decent variety of pedestrians in the images to train with. For this purpose, we created 2 versions of current pedestrians list, each darkened by the factor of 20% from each other.

- **Halos around the pedestrians:** Due to lack of accuracy of the region measuring method, a fine layer of the background that pedestrians were extracted from, still remained around the pedestrians. In the created images, depending on where the person was placed, these thin layers appeared like a halo around the person. Figure 5.10 illustrates some images with halos around the pedestrians.

Figure 5.10: A few synthetic images with halos around the pedestrians in the walkway.

To mitigate this issue, we tried two approaches:

1. **Morphological erosion:** Among morphological operations on image, we applied *erosion* [102] to erode the pedestrians masks. In this way, the halos were ignored to some noticeable extent.
2. **Poisson image editing:** Poisson image editing is a technique for seamlessly blending two images together fully automatically [84]. In addition to erosion, we tried Poisson image editing tool to remove the halos. However, due to our gray-scale and low-resolution images, this tool did not have a great impact on our images.

Afterwards, the images look similar to the ones shown in figure 5.11.

Figure 5.11: Some examples of images with no or less halo around the people in the images.

- **Image Perspective:** Since pedestrians of different sizes were put randomly in the images, we considered people's tallness perspective in the images. As you may observe, in the following figure 5.12, image perspective has not been applied in the images and hence, there are some pedestrians closer to the camera but really small and also some pedestrians quite tall at the end of walkway.

Figure 5.12: created images before considering image perspective.

Humans' height almost follows a Gaussian distribution [98]. Therefore, with respect to [98, 32], we mapped individual's heights with the length of the walkway in the image, considering a Gaussian noise with mean $\mu = 0$ and $\sigma = 3.5$. The resulting images are demonstrated as follows.

Figure 5.13: Synthetic images considering image perspective based on the real distribution of people's height. As you may see, after applying perspective, the images look more realistic.

Thusly, we created a set of 1 million images, each of size 158×158 pixels with up to 29 pedestrians. We assigned 800,000 images for training set and 200,000 instances as the test set. We believe the created synthetic dataset of pedestrians is realistic enough to be able to represent a real-world crowd counting scenario. However, the images can be improved in different aspects and by using diverse image editing and manipulation techniques.

5.1.3 UCSD Crowd counting Dataset

To verify and validate our model, we used UCSD crowd counting dataset created by Chan et al. and used in [11, 13, 14]. The dataset contains video of pedestrians on UCSD walkways, taken from a stationary camera. There are currently two viewpoints available among which we used *vidf* videos. All videos are 8-bit gray-scale, each video file has 200 video frames, with dimensions 238×158. In our experiment, the first 20 videos which are labeled with the number of pedestrians, were incorporated. The center point of each pedestrian defines its' location in the image. A selection of UCSD crowd counting images is shown in figure 5.14.

Figure 5.14: Normal UCSD crowd counting dataset images.

Among the labeled images, we selected the ones in which the number of pedestrians does not exceed 29. Then, images were resized to 158×158 pixels and normalized between 0 and 255. Hence, in total we have a dataset of 3375 real images which are masked with the same filter we used for synthetic pedestrians dataset (shown in figure 5.7). At last, the images look like the following images.

Figure 5.15: Real UCSD images after being masked and resized.

We will use this dataset to first, validate the performance of our model trained with synthetic data, on a real dataset, and then to do a comparison between the work done in [11] and our approach.

5.2 Caffe Deep Learning Platform

Caffe is a clean and modifiable framework for state-of-the-art deep learning algorithms and a collection of reference models. The framework is a BSD-licensed C++ library with Python and MATLAB bindings for training and deployment of general-purpose convolutional neural

networks and other deep models on commodity architectures [55]. It powers on-going research projects and large-scale industrial applications in vision, speech and multimedia by CUDA [1] GPU computation.

Caffe is composed of two main components, models' architecture and design, and model optimization. In the rest of this section, we present the main components and parameters of both model implementation and optimization.

5.2.1 Model Implementation and Design

The main components of Caffe architecture are outlined below:

1. **Data storage:** Caffe stores and communicates data in 4-dimensional arrays called *blobs*. Blobs provide a unified memory interface, holding batches of data, parameters, or parameter updates. Blobs conceal the computational overhead by synchronizing from the CPU host to the GPU device as needed. Figure 5.16 shows the blobs connected to a convolution layer implemented by Caffe.

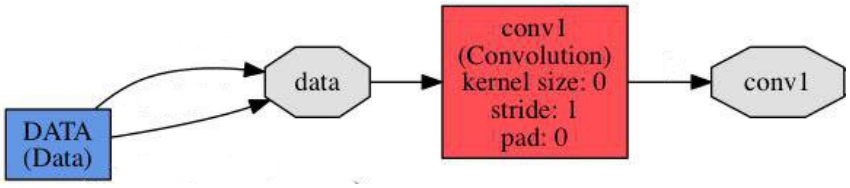


Figure 5.16: Input (data) and output (conv1) blobs of a convolution layer in a CNN implemented in Caffe.

Caffe supports some data sources such as LevelDB or LMDB (Lightning Memory-Mapped Database), HDF5, MemoryData, ImageData, etc. However, large-scale data is stored in LevelDB databases since it reads the data directly from memory [41].

2. **Layers:** A caffe layer takes blobs as input and yields one or more as output. In a network (as described in chapter 2.2.2.2), each layer plays two important roles: a forward pass that takes the inputs and produces the outputs, and a backward pass that takes the gradient with respect to the output, and computes the gradients with respect to the parameters and to the inputs, which are in turn back-propagated to earlier layers [55].

   Caffe supports an exhaustive set of layers, including the followings [55]:

   (a) Convolution, pooling, fully connected,

   (b) Nonlinearities like rectified linear and logistic, local response normalization, element-wise operations, and

   (c) Losses like softmax and hinge

[1] CUDA is a parallel computing platform and application programming interface (API) model created by NVIDIA [42], processing over 40 million images a day on a single K40 or Titan GPU [55].

3. **Networks and run mode:** Caffe ensures the correctness of the forward and backward passes for any directed acyclic graph of layers. A typical network begins with a data layer laying down at the bottom going up to the loss layer that computes tasks' objectives. The network is run on CPU or GPU independent of the model definition.

4. **Training a network:** Training phase in Caffe is done by classical stochastic gradient descent algorithm. When training, images and labels pass through different layers lead into the final prediction into a classification layer that produces the loss and gradients which train the whole network. Figure 5.17 illustrates a typical example of a Caffe network.

   Finetuning, the adaptation of an existing model to new architectures or data, is a standard method in Caffe. Caffe finetunes the old model weights for the new task and initializes new weights as needed. This capability is essential for tasks such as knowledge transfer [21], object detection [35], and object retrieval [43], [55].

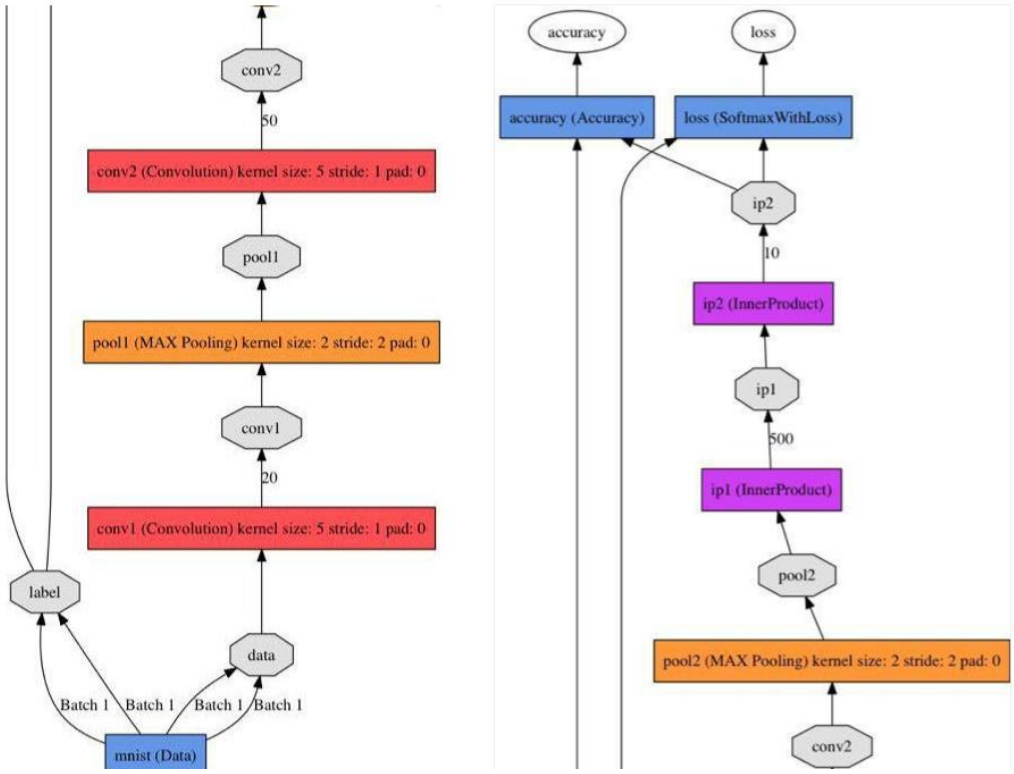


Figure 5.17: From bottom-left to top-right, Lenet architecture for MNIST digit classification example of a Caffe network, where boxes represent layers and octagons represent data blobs produced by or fed into the layers [55].

Justification: We decided to use Caffe because, it addresses computation efficiency problems (as likely the fastest available implementation of deep learning frameworks at the time of performing this study, adheres to software engineering best practices, providing unit tests for correctness and experimental rigor and speed for deployment. It is also well-suited for research use, due to the well-implemented modularity of the code, and the clean separation of network

definition (usually the novel part of deep learning research) from actual implementation[55]. In addition, it provides a python wrapper which exposes the solver module for easy prototyping of new training procedures.

5.2.2 Model Optimization

Our work in general is an optimization problem since we project the results as loss functions we try to minimize. For this reason, model optimization methods have a critical impact on the performance of the model. In Caffe, *Solver* file orchestrates optimization by coordinating the network's forward inference and backward gradients to form parameter updates that attempt to improve the loss. Among Caffe solver methods, we use stochastic gradient descent (explained in chapter 2.2.3). The use of SGD in deep convolutional neural network setting is motivated by the high cost of running back propagation over the full training set. SGD can overcome this cost and still lead to fast convergence.

Although Caffe provides several optimization methods, in this section we briefly describe merely optimization methods and *hyper-parameters*[2] incorporated in our proposed algorithms.

5.2.2.1 Batch Size

Since Caffe is trained using stochastic gradient descent, at each iteration, it computes the (stochastic) gradient of the parameters with respect to the training data and updates the parameters in the direction of the gradient. On the other hand, to compute the gradient w.r.t the input data, we need to evaluate all training samples at each iteration which is prohibitively time-consuming, specially when we are dealing with a great amount of data.

In order to overcome this issue, SGD approximates the exact gradient, in a stochastic manner, by sampling only a small portion of the training data at each iteration. This small portion is the *batch*. In other words, the batch size defines the amount of training data we feed to the network at each iteration. The larger the batch size, the more accurate the gradient estimate at each iteration will be.

5.2.2.2 Learning Rate

Learning rate is a decreasing function of time. It's a common practice to decrease the base learning rate (base_lr) as the optimization/learning process progresses. In Caffe, different learning policies exist among which we tried the followings:

- ***Fixed:*** which always returns the base learning rate.
- ***inv:*** which returns

$$base\_lr \times (1 + \gamma \times iteration)^{(-Power)}$$

where:
γ*: the factor learning rate drops by.*
power: another parameter to compute the learning rate.

[2]Hyper-parameters govern the underlying system on a "higher level" than the primary parameters of interest. They are not model parameters that are learned during training phase and instead, they are set by the designer a priori.

- ***step:*** that returns

$$base\_lr \times \gamma^{floor(\frac{iteration}{step})}$$

where:
γ*: the factor learning rate drops by.*
step: the number of iteration at which the learning rate drops.

- ***multi-step:*** similar to step but it allows non uniform steps defined by step value.

Although there are numerous empirical studies and rules of thumb to treat learning rate [95, 109, 75], basic learning rate and learning policy are highly problem-dependent.

5.2.2.3 Weight Decay

As a part of Back Propagation and a subset of regularization methods, *weight decay* adds a penalty term to the error function by multiplying weights to a factor slightly less than 1 after each update.

It has been observed in numerical simulations that a weight decay can improve generalization in a feed-forward neural network. It is proven that weight decay has two effects in a linear network. Firstly, it suppresses any irrelevant components of the weight vector by choosing the smallest vector that solves the learning problem. Secondly, if the size is chosen correctly, a weight decay can suppress some of the effects of static noise on the targets, which improves generalization significantly [78].

5.2.2.4 Momentum

One of the potential problems with stochastic gradient descent is having oscillations in the gradient, since not all examples are used for each calculation of the derivatives. This can cause slow convergence of the network. One strategy to mitigate this problem is the use of *Momentum.* The momentum method introduced by Polyak, 1964, is a first-order optimization method for accelerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations. Given an objective function $f(\theta)$ to be minimized, momentum is given by:

$$\nu_{t+1} = \mu\nu_t - \alpha\nabla \tag{5.1}$$

$$\theta_{t+1} = \theta_t + \nu_{t+1} \tag{5.2}$$

where $\alpha > 0$ is the learning rate, $\mu \in [0, 1]$ is the momentum coefficient, and $\nabla f(\theta_t)$ is the gradient at θ_t [99].

For example, if the objective has a form of a long shallow ravine leading to the optimum and steep walls on the sides, standard SGD will tend to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum [80]. The objectives of deep architectures have this form near local optima and thus standard SGD can lead to very slow convergence particularly after the initial steep gains.

Momentum, by taking the running average of the derivatives, by incorporating the previous update in the update for the current iteration, is one method for pushing the objective more quickly along the shallow ravine [80].

5.2.2.5 Number of Iterations

In learning process, common convergence criteria are: a maximum number of iteration; a desired value for the cost function is reached; or training until the cost function shows no improvement in a number of iterations. In our implementation, we use the maximum number of iteration as our systems' convergence policy.

The number of iterations plays an important role in the training process. iteration number is in an inverse correlation with the number of instances and also the batch size. In any network, batch size and iteration number compensate for one another. For instance, in case of lack of memory, one option would be to decrease the batch size and increase the number of iterations accordingly.

5.3 The Architecture

Having Caffe platform introduced, we propose two CNN-based deep architectures for the analysis we did regarding two learning to count problems, counting the number of even-digits and the number of pedestrians in an image. In learning object features for vision tasks and by the use of DCNN, the depth of network plays a crucial role. The deeper the model, the better it learns. However, issues like overfitting[3] and underfitting should not be left neglected. Therefore, in this section, networks' settings and architectures for even-digits and crowd counting problems will be described separately.

5.3.1 Even-digit Counting

For learning to count even digits problem, since we used MNIST dataset to generate our images, we considered the architecture proposed by LeCun et al. for classic MNIST hand-written digit recognition problem [65] (shown in figure 5.17), as the base of our design. In addition, we took into account the architecture used in state-of-the-art [94]. Figure 5.18 demonstrates this architecture. From there, we modified the architecture to optimize the performance of the network.

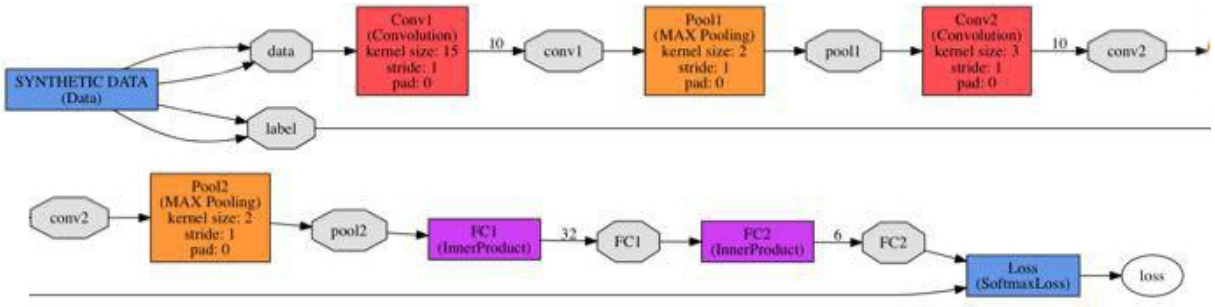


Figure 5.18: From top-left to bottom-right, the network proposed in [94] for Even digits recognition task.

In a similar experiment with maximum 5 digits in images, Seguí et al. in [94] defined a four layers CNN with two convolutional layers for classifying even digits present in images. Each convolu-

[3]Overfitting occurs when a statistical model describes random error or noise instead of the underlying relationship, and vice versa for underfitting.

tional layer consists of several elements: a set of convolutional filters, ReLU non-linearities, max pooling layers and normalization layers. The first convolutional layer outputs 10 filters of dimensions 15×15, and the second convolution layer has 10 filters, each of size 3×3. Each convolution layer is followed by a max-pooling layer with a 2×2 kernel. The output of last pooling layer is fed to two fully-connected layers with respectively 32 and 6 number of outputs. They also used Softmax loss layer on top of their architecture to compute the multinomial logistic loss layer[4]. Table 5.1 presents the main components of the network designed by [94].

Network parameters	
Layers	**setting**
Conv1	$10 \times 15 \times 15$
Pool1	$max(2 \times 2)$
Conv2	$10 \times 3 \times 3$
Pool2	$max(2 \times 2)$
FC1	32 outputs
FC2	6 outputs

Table 5.1: The DCNN proposed by [94] for learning the number of even digits present in the images.

However, in our implementation, due to higher complexity, we decided to apply a deeper CNN with more parameters. In our network, the data layer fetches the images and labels from the disk, passes it through the first convolutional layer with 20 filters, each of size 15×15 followed by a ReLU non-linearity and LRN normalization layer. Then the output is max-pooled by the kernel size of 2×2. This process repeats again but this time with the second convolutional layer having 50 filters of size 3×3. In all convolution and pooling layers, the *stride* = 1 and *padding*[5] = 1 are considered (see section 2.4.1 for more explanation regarding padding and stride).
the output of the second pooling layer is fed to two fully connected (inner product) layers with respectively 64 and 1 number of outputs (since the problem is approached as a regression task). Also the first fully connected layer is followed by ReLU non-linearity.

However, a well-designed network cannot solely guarantee an optimal performance for the model. The responsibilities of learning are divided between the network for yielding loss and gradients, and the optimization methods (solver) and parameters for overseeing the optimization and generating parameter updates. Among Caffe solvers, As described in section 2.2.3, we use Stochastic Gradient Descent optimization method. Apart from solver method, the solver parameters (network's hyper-parameters) need to be set attentively in order to optimize the model performance. Therefore, to optimize the model performance, the networks' hyper-parameters are set as below while table 5.2 provides a summary of the hyper-parameters' settings.

- **Batch size:** Due to the non-complex and low-resolution dataset we are training on, we were able to use batches of size 256 for our training and testing phases.
- **Learning rate:** After trying different initial values in range of $(10^{-6}, 1)$, we set the basic learning rate to $\alpha = 0.0001$. However, for our experiment we chose *multi-step* learning

[4]This function displays a similar convergence rate to the hinge loss function, and since it is continuous, gradient descent methods can be utilized. Also, functions which correctly classify points with high confidence are penalized less. This structure is highly sensitive to outliers in the data.

[5]Zero padding is a simple concept; it simply refers to adding zeros to end of an image to increase its length.

policy in which, after each *stepsize*=40000 iterations, the learning rate drops by the rate of Gamma $\gamma = 0.1$. This initialization is based on rules of thumb used in [58].

- **Momentum:** We use momentum $\mu = 0.9$. Because, momentum setting μ effectively multiplies the size of our updates by a factor of $\frac{1}{1-\mu}$. Hence, changes in momentum and learning rate ought to be accompanied with an inverse correlation. When momentum $\mu = 0.9$, we have an effective update size of 10 since we also drop the learning rate by the factor of $\gamma = 0.1$.
- **Weight decay:** Weight decay as a penalty term to the error function, has a constant value of 0.0005. This decay constant is multiplied to the sum of squared weights.
- **Iterations:** Given the time and hardware we had, we managed to let the system train for 1,600,000 iterations.

Network hyper-parameters	
Hyper-parameters	**setting**
Learning rate	0.0001
Learning policy	*Multi_step*
Momentum	$\mu = 0.9$
Weight decay	0.0005
Batch size	256
Iterations	1600000

Table 5.2: Proposed settings for network's hyper-parameters for counting number of even digits task.

We should also mention that at the top layer of the network, we used Euclidean Loss layer to compute the euclidean distance between the predictions and the ground truth. Figure 5.19 shows a scheme of the architecture.

Figure 5.19: Proposed network architecture for Even digits counting task

The architecture has been designed intuitively and based on a heuristic approach where we tried different values for each hyper-parameter at a time and selected the values that provided us with an improvement in the performance of the network. Hence, one may be able to improve the performance by implementing a different architecture or with another settings for the hyper-parameters of the network.

5.3.2 Crowd Counting

For the task of counting pedestrians, due to the more complex images than the previous task, we designed a deeper architecture. Moreover, we applied the same settings of hyper-parameters to a different architecture. However, as a reminder, table 5.3 summarizes the network's settings for the hyper-parameters.

Network hyper-parameters	
Hyper-parameters	**setting**
Learning rate	0.0001
Learning policy	*Multi_step*
Momentum	$\mu = 0.9$
Weight decay	0.0005
Batch size	256
Iterations	1600000

Table 5.3: Proposed settings for network's hyper-parameters for counting pedestrians.

As a baseline, we considered state-of-the-art architecture designed by Seguí et al. 94 for counting the number of pedestrians in the images. As shown in figure 5.20, in their work, they used a five layer architecture CNN with two convolutional layers followed by three fully connected layers. Convolutional layers contain their main elements (ReLU and LRN layers), and each outputs 8 filters with kernel sizes 9×9 and 5×5 respectively.

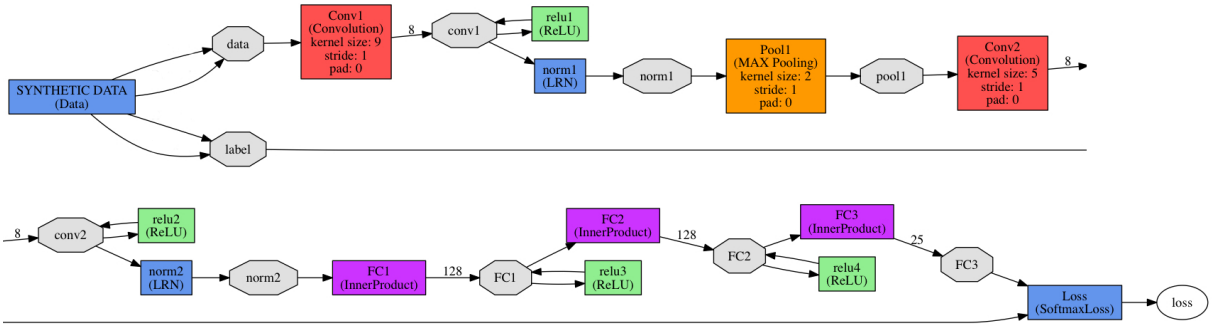


Figure 5.20: From top-left to bottom-right, the DCNN proposed by [94] for learning to count the number of pedestrians in a walkway.

They applied only one max-pooling layer and after the first convolutional layer in their design. Finally, three fully connected layers with number of outputs 128, 128 and 25 classify the output of the network into 25 classes (maximum pedestrians in the images) of the number of people in

the image. Similar to the previous task, a softmax loss layer projects the performance of the model. Table 5.4 describes a summary of their network.

Network parameters	
Layers	**setting**
Conv1	$8 \times 9 \times 9$
Pool1	$max(2 \times 2)$
Conv2	$8 \times 5 \times 5$
FC1	128 outputs
FC2	128 outputs
FC3	25 outputs

Table 5.4: The deep CNN proposed by Seguí et al. 94 for learning the number of pedestrians in a walkway.

However, since up to 29 pedestrians are present in our images with high amount of overlapping, in our design we added one convolutional layer to the base architecture. We also increased the number of parameters in the network. In our architecture, the data blobs pass through 4 convolutional layers. The first convolutional layer has 4 filters, each with 5×5 kernel and the other 3 layers have again 4 filters but each of size 3×3. Similar to the previous model, each convolutional layer is followed by ReLU non-linearity layer and LRN normalization layer. Also the stride and padding values for all the convolutional layers are respectively equal to 1 and 0.

In order to not lose information, we used merely two pooling layers for the first two convolutional layers. Each pooling layer has a kernel size of 2×2 with $stride = 1$ and $padding = 0$. Figure 5.21 illustrates the proposed architecture.

Figure 5.21: Proposed network architecture for learning the number of pedestrians in the image.

As you may see in figure 5.21 above, there are three fully connected layers to regress the number of pedestrians in images. The first two fully connected layers have 16 outputs each,

and are connected to ReLU non-linearity layers. The last layer however, with solely one output, passes the models' prediction of the number of pedestrians to the Euclidean loss layer to calculate the sum of squares of differences of its two inputs, the true labels and predictions.

To the best of our knowledge, the designed architectures outperform other architectures we tried previously (a different range of architectures starting from 1 up to 5 convolutional layers with different settings), while also fastening the training phase . However, apart from the basic knowledge about network architectures, hyper-parameters initialization and some rules of thumb of successful experiences in similar works, the rest of design has been done intuitively. Therefore, similar to the first architecture (problem of counting even digits), there would be other ways to improve the architecture and performance of the model.

6 Experiments and Results

Prior to jumping to the final results obtained with our algorithm, this chapter will attempt to give insight into the experiments we designed to verify our hypothesis. We examine our proposed methodology on two different but related scenarios. In the former scenario, we tackle the learning to count even hand-written digits, and for the latter one, crowd counting problem is considered. The findings of the experiments will determine to what extent synthetic data generation could help deep convolutional networks to learn when dealing little amount of data. Hence, this chapter gives full particulars of each experiment, and presents obtained results as compared to similar works.

6.1 Learning to Count Even-odd Handwritten Digits

We intend to explore the features learned when training a deep CNN, in order to understand the underlying representations. To this end, we designed a synthetic problem of counting even digits in images. This experiment clearly illustrates the basic idea behind this work, where we hypothesize that the features learned during this task are:

1. Efficient descriptors of digits to enable the model to count them.
2. Sufficiently representative to be applied for the digit recognition tasks.

In this experiment, we feed images of digits into a deep convolutional neural network. Images are labeled with the number of even digits in each image. We took a regression analysis to tackle this problem. Therefore, the output of the network is a single number determining the model's prediction for the amount of even digits present in the images.

6.1.1 Dataset

As it was meticulously described in section 5.1.1, we synthetically generated Even-odd digits dataset to use for this task. The dataset holds the following properties:

- Images of MNIST hand-written digits where each image contains up to 15 digits. Also the images are made in gray-scale.
- Each image has a dimension of 100×100 and each digit in the image is 18×18 pixels.
- A minimum distance of 18 pixels is considered between each two digits (from center to center) in an image to prevent overlapping.

- The dataset is generated uniformly. It means that there are equal number of images for each amount of even digits in the image. For instance, the number of images with 5 even digits is the same as number of images containing 10 even digits.
- The dataset of 1,000,000 samples is divided into a training set of 800,000 and a test set of 200,000.

Since we use Caffe platform to perform our experiments, we convert the images into LMDB format. LMDB uses memory-mapped files, so it has the read performance of a pure in-memory database while still offering the persistence of standard disk-based databases, and is only limited to the size of the virtual address space (it is not limited to the size of physical RAM). Therefore, we created two LMDB files for training and testing sets to be fed to the network.

6.1.2 Learning process

As discussed in chapter 5.3.1, for learning to count the number of even digits, we designed a deep CNN. Table 6.2 shows a review of the network's specification and architecture while table 6.1 recaps the settings for the hyper-parameters.

Network hyper-parameters	
Hyper-parameters	**setting**
Learning rate	0.0001
Learning policy	*Multi_ step*
step size	40000
Momentum	$\mu = 0.9$
Weight decay	0.0005
Batch size	256
Iterations	1600000

Table 6.1: Proposed settings for network's hyper-parameters.

Network parameters	
Layers	**setting**
Conv1	$20 \times 15 \times 15$
ReLU1	max(x,0)
LRN1	α=0.0001, β=0.75
Pool1	$max(2 \times 2)$
Conv2	$50 \times 3 \times 3$
ReLU2	max(x,0)
LRN2	α=0.0001, β=0.75
Pool2	$max(2 \times 2)$
IP1	64 outputs
ReLU3	max(x,0)
IP2	1 outputs

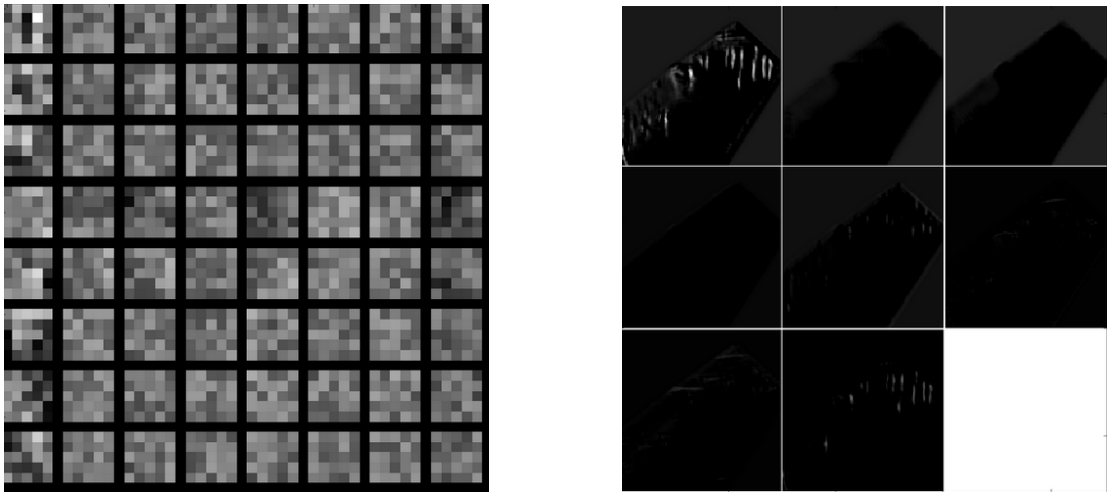
Table 6.2: Proposed architecture's settings.

The algorithm trained on a GPU NVIDIA [56] Tesla K40 [71]. It took almost 4 days to train 800,000 samples and test over 200,000 images. The learning curves of this process are shown in the figure 6.1.

Figure 6.1

ADD EXPLANATION OF THE LEARNING CURVES

Moreover, figure 6.2 illustrates the filters and the output (rectified responses of the shown filters) of the first convolutional layer for one input sample in the network.



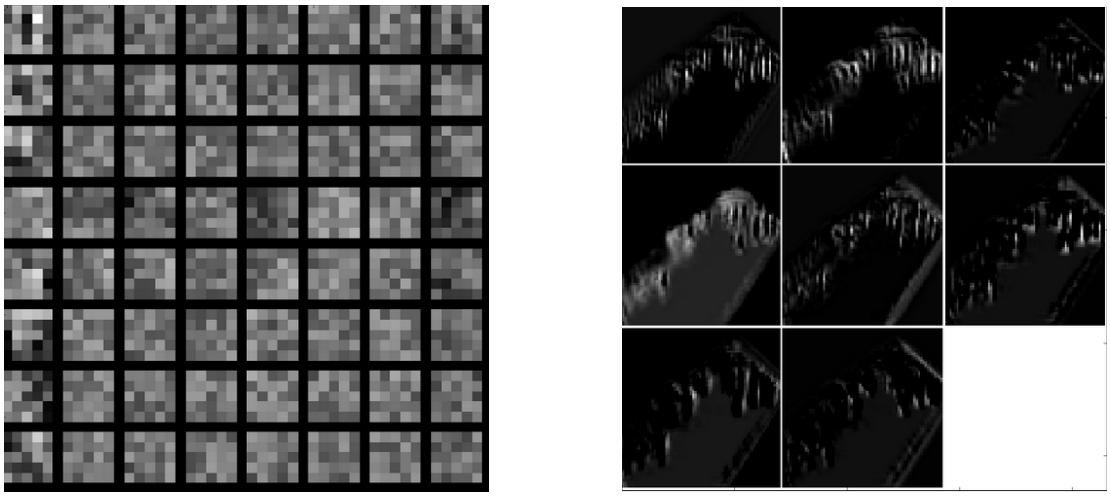
(a) The obtained filters. **(b)** The output correspondent to the filters.

Figure 6.2: Filters and output of the first convolutional layer in the network.

A SHORT EXPLANATION ABOUT THE FIGURES

Accordingly, figure below demonstrates the filters and their corresponding output after the second convolutional layer.

A SHORT TRANSITIVE PARAGRAPH TO GO TO RESULTS



(a) The obtained filters. **(b)** The output correspondent to the filters.

Figure 6.3: Filters and output of the second convolutional layer in the network.

6.1.3 Experimental Results

In addition to the Euclidean loss function that we implemented in our architecture, we use two other error measurements to evaluate and compare the performance of our model. These measurements are briefly described and justified in below:

1. **Mean squared error (MSE)**: Mean squared error is arguably the most important criterion used to evaluate the performance of a predictor or an estimator. The mean squared error is also useful to relay the concepts of bias, precision, and accuracy in statistical estimation. The MSE is the second moment (about the origin) of the error, and thus incorporates both the variance of the estimator and its bias [67]. The MSE can be estimated by

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{x_{pred}} - x_{obs})^2$$

where:
N: the number of instances
$\hat{x_{pred}}$: a vector of N predictions
x_{obs}: a vector of observed value (ground truth) for input x

The mathematical benefits of mean squared error are particularly evident in its use for analyzing the performance of linear regression, as it allows one to partition the variation in a dataset into variation explained by the model and variation explained by randomness.

2. **Mean absolute error(MAE)**: MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. In other words, it measures how close predictions are to the eventual outcomes [105]. MAE can simply be computed by:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|x_{pred} - x_{obs}|$$

where:
N: number of instances
x_{pred}: prediction for instance x
x_{obs}: observed value (ground truth) for instance x

Having chosen error measurements explained, for the task of counting even digits, using 800,000 training samples and 200,000 test images, and after 1,600,000 iterations, our model obtained the results shown in table 6.7.

Results

Error measurement	Error
Euclidean loss	0.20
Mean squared error	0.20
Mean absolute error	0.20

Table 6.3

Accordingly, the spread-error plot corresponding to the model is provided.

Figure 6.4

As you may observe in Figure 6.8, most of the frames are correctly labeled. Moreover, most of the errors correspond to adjacent number values.
A similar experiment was done by Seguí et al.. In their work, they used images with up to 5 digits with slightly different architecture. Also each digit in the images had a dimension of 28×28. They approached the problem as a classification task.

Therefore, in order to provide a fair comparison, we demonstrate how far each model's accuracy stands from random. These comparison has been shown in the table 6.4.

Performance Comparison

Experiments	Chance	Accuracy
Our results	0.20	0.30
Results obtained by Seguí et al.	0.20	0.30

Table 6.4: .

EXPLANATION OF THE COMPARISON

6.1.4 Conclusion

CONCLUSION AFTER THE EXPERIMENT

6.2 Counting Pedestrians in a Walkway

The first experiment proved that features can be learned automatically using deep CNN. It also shows that these deep features can be used to detect the object of interest in different but similar tasks. However, the use of deep architectures for fully supervised learning problems requires a large amount of annotated data. At the moment, for crowd counting problems, such data does not exist for research purposes.

Hence, we propose a crowd counting problem using synthetic dataset in order to examine first, how well the model trained with synthetic images would perform. And then, whether the model is applicable for a real-world crowd counting scenario or not. We believe that this experiment will enlighten these questions.

6.2.1 Datasets

As fully discussed in chapter 5.1.2, for the particular case study, we synthetically created a dataset of 1 million images. To recap, the dataset specifications are the followings:

- Each image is in gray-scale, 158×158 pixels, normalized, and contains up to 29 pedestrians in a walkway. Images are labeled with the number of pedestrians present in the image.
- Pedestrians' location in the images are center-based. A mask (region of interest) has been applied to images that defines the area in which the pedestrians are counted. Thus, the person is labeled if it's center is put in the region of interest.
- Out of 1 million images, 800,000 are considered for training and 200,000 for testing sets.

In addition, in order to examine the performance of our model in a real-world scenario, we used UCSD crowd counting dataset [11] with the underneath properties:

- The dataset consists of 3375 video frames shot by a stationary camera. Each image has up to 29 pedestrians present in the image.
- Images are in gray-scale, resized to 158×158 pixels, normalized (0 to 255) and filtered by the same mask (region of interest) as the other dataset. Labels are the number pedestrians placed in the region of interest.
- Pedestrians annotation is center-based. The ones inside the mask are labeled.

- Images are continuous video frames of 20 different videos. Each video contains up to 200 images.

Once again, to improve the training time, we converted the synthetic data into LMDB format for Caffe to read it in the fastest way.

6.2.2 The Learning Process

As fully described in chapter 5.3.2, we implemented a deep convolutional neural network in Caffe to learn the number of pedestrians in the walkway. As a reminder, the designed network has the components mentioned in table 6.6. The network trains with stochastic gradient descent and our justification for the setting the solver parameters (hyper-parameters) remains the same as previous experiment (explained in section 5.3.2). However, once again table 6.5 provides a review of the settings for network's hyper-parameters.

Network parameters	
Layers	**setting**
Conv1	$20 \times 15 \times 15$
ReLU1	max(x,0)
LRN1	α=0.0001, β=0.75
Pool1	$max(2 \times 2)$
Conv2	$50 \times 3 \times 3$
ReLU2	max(x,0)
LRN2	α=0.0001, β=0.75
Pool2	$max(2 \times 2)$
IP1	64 outputs
ReLU3	max(x,0)
IP2	1 outputs

Table 6.5: Proposed architecture's settings.

Network hyper-parameters	
Hyper-parameters	**setting**
Learning rate	0.0001
Learning policy	*Multi_ step*
step size	40000
Momentum	$\mu = 0.9$
Weight decay	0.0005
Batch size	256
Iterations	1600000

Table 6.6: Proposed settings for network's hyper-parameters.

Similarly, we used GPU NVIDIA TESLA K40 to train our network. After 5 days of training, we obtained the following learning curves :

Figure 6.5

The network was able to learn sufficient features corresponding to the pedestrians. To illustrate, the learned features for one input sample, and after each convolutional layer in the network, has been depicted in the following figures:

Figure 6.6

And finally, the network along with the outputs of different layers is shown in a schematic form in figure 6.7.

Figure 6.7

A SHORT TRANSITIVE PARAGRAPH TO GO TO RESULTS

6.2.3 Experimental Results

Equivalently to the first analysis, we use mean squared error, mean absolute error and the default euclidean loss function to present the results obtained in this experiment. Having that said, the following results were obtained:

Results

Error measurement	Error
Euclidean loss	0.20
Mean squared error	0.20
Mean absolute error	0.20

Table 6.7: .

These results were attained after 1,600,000 iterations on 800,000 train and 200,000 test sets. Correspondingly, figure 6.8 depicts the spread-error plot of the model performance on the test set.

Figure 6.8

As you may notice, the network's predictions closely follow the target values. Also a direct correlation between the number of pedestrians and error deviation can be inferred from the spread-error plot which is reasonable due to the more overlapping in the crowded scenes.
In state-of-the-art, Seguí et al. did a similar experiment with maximum 25 pedestrians in each image. Beside that, the difference between their work with this analysis, mainly revolves around the architecture design. Nonetheless, the table of comparison between these two approaches are shown in below:

Performance Comparison

Experiments	Chance	Accuracy
Euclidean loss	0.20	0.30
Mean squared error	0.20	0.30

Table 6.8: Table caption font is different from the normal text font in order to get a better differentiation – I like it that way.

As you may observe, although we there are more pedestrians in our images, our model shows a more promising performance.
As an innovative part of this Master thesis, we were interested to figure out if we have been able to overcome the exhaustive labeling in problems such as crowd counting. For this reason, we tested our model on a real UCSD crowd counting dataset [11]. Table 6.9 demonstrates the performance of our model on UCSD dataset.

Results

Error measurement	Error
Euclidean loss	0.20
Mean squared error	0.20
Mean absolute error	0.20

Table 6.9: Table caption font is different from the normal text font in order to get a better differentiation – I like it that way.

As it can be inferred from the above table, the results are convincing and rational given the results we achieved on the synthetic test set and the inevitable differences between the synthetic and real datasets.
Moreover, we compared our model with Chan et al. research study in [11] regarding crowd counting using exhaustive labeling and hand-crafted feature detectors using UCSD dataset. One difference we need to mention is that in their work, they computed the results for pedestrians towards camera and away from camera separately. However, in our dataset, we consider all pedestrians as one. Hence, we compare the average results of people towards and away from camera with our results. This comparison has been drawn in the following table:

Performance Comparison

Experiments	Chance	Accuracy
Euclidean loss	0.20	0.30
Mean squared error	0.20	0.30

Table 6.10: Table caption font is different from the normal text font in order to get a better differentiation – I like it that way.

6.2.4 Conclusion

CONCLUSION AFTER THE EXPERIMENT

7 Conclusions and Future Work

In this thesis we have made several contributions on the benefits of synthetic data generation for the application of deep convolutional neural networks for problems with small training sets casted as learning to count problems. Our contributions include 1) two synthetically generated and automatically annotated datasets for even-odd handwritten digit recognition and crowd counting in the images, 2) counting the number of pedestrians in images in a privacy-preserving manner (due to the synthetic nature of images) involving no object tracking or detection techniques, 3) deep convolutional-based architectures to sufficiently learn object representations without the need for hand-crafted feature detectors, 4) a deep architecture trained with synthetic data which is capable of counting pedestrians in a real-world problem, and 5) the experimental validation of our improvements with the previous feature detecting techniques as well as with related methods in the state-of-the-art.

7.1 Conclusions

This work started off with verifying our very first hypothesis. We hypothesized that synthetic data can be applied as a well-suited surrogate for the application of deep convolutional neural networks in problems with small datasets. We introduce tasks of counting objects as problems in which large amount of labeled data is missing or hard to collect. To this end, we introduced a synthetic set of images with up to 15 MNIST digits in the images with the number of even digits as labels (section 5.1.1). In addition, we configured a deep CNN for an even-digits counting problem. In addition, Having a deep model trained, we proved the functionality of synthetic datasets for counting objects using DCNN. Moreover, we obtained noteworthy results implying applicability of deep CNN as a surrogate for previous object detection tasks using manually designed feature detectors.

Furthermore, we examined our methodology on a more complex problem of counting the number of pedestrians in a walkway. Applying deep learning methods for supervised learning problem relies on availability of big annotated data. Exhaustive labeling becomes prohibitive in most of the cases. To overcome this issue, we created another synthetic dataset of pedestrians in the walkway labeled with the number of people present in the images (see chapter 5.1.2). Designing another deep network, once again we achieved satisfactory results while discarding extensive labeling effort and hand-crafting feature detectors.

In addition to the introduced synthetic data generation algorithms, one of our main contributions and the novelty of this work was to evaluate our proposed methodology from a practical aspect by testing it on real-world problems. For that purpose, we tested our model which was trained with highly-realistic synthetic dataset, on a similar but relatively small set of real images which were labeled manually with the number individuals in the image(see section 5.1.3).

Comparing our model's performance with a related crowd counting study (using manual labeling and highly specialized feature descriptors) on the same dataset, we may conclude that the application of deep features, not only softens annotation efforts and omits the usage of numerous hand-crafted feature detectors, but also improves the results of such previous approaches. In addition, preserving individuals' privacy will no longer be a matter of concern since our methodology does not include any object tracking techniques and applies synthetic images.
In conclusion, the findings of this master's thesis suggest the generation of synthetic images for deep learning techniques as a suitable replacement for the lack of training data in counting problems. In addition, this work shows the applicability of deep convolutional neural networks as a surrogate for previous object counting problems, by showing representativity for the object of interest in similar but different tasks. Furthermore, it substantiates the feasibility of incorporation of such methods in real world problems by alleviating the learning process while improving the results.

7.2 Future Work

Any research study opens up some interesting lines of research that deserve further attention and study. There are questions and limitations that need to be addressed along with the improvements that can be made. Our work is no exception. In this master's thesis, we have observed and pointed out shortcomings and areas in which, one can improve our work (regarding datasets (chapter 5.1), and networks architectures (chapter 5.3)). Some of the deficits of our study along with some hints to advance and enrich this research which were out of scope of this work, are mentioned below:

1. We generated synthetic gray-scale datasets to tackle our proposed problems. Although we endeavored to improve the data to make it look as realistic as possible, there are still many other ways to improve the data and undoubtedly the performance of the model. One idea could be feeding colorful images to a deeper network for learning more features and predicting more accurate forecasts.

2. Due to the intuitive nature of deep networks design, one can configure and design a model more in-tune with the problems. Making the network deeper with more parameters might be an option.

3. Another idea of ours for improving the performance of our crowd counting model on the real dataset is to use transfer learning[1]. In this case we could couple two architectures: one with the synthetic data and label and the other with the real images. In this way, we can use the features learned on the synthetic data architecture for testing the real data simultaneously and update the weights of the training architecture with synthetic samples.

4. Another analysis could be a combination of deep CNN to learn to count out of sample. For instance, our even-digits counting dataset contained up to 15 digits. However, one could train a deep network with images with up to for example 30 digits in each image and label images with the number of all digits in the image and not just even digits. We assume that a combination this model and ours could count the number of even digits in images with more than 15 even digits. The same experiment can be done for counting pedestrian

[1]Transfer learning is the improvement of learning by transferring the knowledge from a related learned task.

task in order to provide a model capable of counting people in very crowded scenes such as demonstration, marathon, etc.

5. A combination of such deep algorithm for crowd counting can be combined with faster programs to develop real-time pedestrian detection. Due to the different size and appearance of real pedestrians, deep algorithms might not be fast enough to detect pedestrians in the moment. However, faster vision detection algorithms such as *cascade detection* can be used in early stages of the system to fasten the detection process while a deep classifier can improve the accuracy of the algorithm at the final stage. An algorithm with such optimal trade-off between detection accuracy and speed can be used in various industries such as pedestrian detection systems in self-driving vehicles.

6. Another process to obtain more realistic pedestrians would be using 3D human models in cross platforms Cinema 4D or Unity to create super-realistic set of various pedestrians with different poses and style. Generating pedestrians in this way would completely remove the halo issue (described in chapter 5.1.2.2) around the pedestrians.

References

[1] Amit, D. J., Gutfreund, H., and Sompolinsky, H. (1987). Statistical mechanics of neural networks near saturation. *Annals of physics*, pages 30–67.

[2] Arasu, A., Shriraghav, K., and Li, J. (2012). Synthetic data generation. US Patent App. 13/166,831.

[3] Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

[4] Bengio, Y. (2013). Deep learning of representations: Looking forward. In *Statistical language and speech processing*, pages 1–37. Springer.

[5] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pages 1798–1828.

[6] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153.

[7] Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

[8] Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118.

[9] Cappelli, R., Erol, A., Maio, D., and Maltoni, D. (2000). Synthetic fingerprint-image generation. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 3, pages 471–474. IEEE.

[10] Chan, A. and Vasconcelos, N. (2013). Ground truth annotations for ucsd dataset. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[11] Chan, A. B., Liang, Z.-S. J., and Vasconcelos, N. (2008). Privacy preserving crowd monitoring: Counting people without people models or tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–7. IEEE.

[12] Chan, A. B., Morrow, M., and Vasconcelos, N. (2009). Analysis of crowded scenes using holistic properties. In *Performance Evaluation of Tracking and Surveillance workshop at CVPR*, pages 101–108.

[13] Chan, A. B. and Vasconcelos, N. (2009). Bayesian poisson regression for crowd counting. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 545–551. IEEE.

[14] Chan, A. B. and Vasconcelos, N. (2012). Counting people with low-level features and bayesian regression. *Image Processing, IEEE Transactions on*, pages 2160–2177.

[15] Ciresan, D. and Meier, U. (2015). Multi-column deep neural networks for offline handwritten chinese character classification. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–6. IEEE.

[16] Cireşan, D., Meier, U., Masci, J., and Schmidhuber (2012). Multi-column deep neural network for traffic sign classification. *Neural Networks*, pages 333–338.

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

[18] Dekel, O. and Shamir, O. (2009). Good learners for evil teachers. In *Proceedings of the 26th annual international conference on machine learning*, pages 233–240. ACM.

[19] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE.

[20] Deng, L. and Yu, D. (2014). Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, pages 197–387.

[21] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2013). Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*.

[22] Dong, L., Parameswaran, V., Ramesh, V., and Zoghlami, I. (2007). Fast crowd segmentation using shape indexing. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

[23] Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

[24] Eggert, C., Winschel, A., and Lienhart, R. (2015). On the benefit of synthetic data for company logo detection. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 1283–1286. ACM.

[25] Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. (2014). Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2154.

[26] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32:1627–1645.

[27] Fergus, R. (2013). Nips 2013 tutorial on deep learning for computer vision.

[28] Flaccavento, G., Lempitsky, V., Pope, I., Barber, P., Zisserman, A., Noble, J., and Vojnovic, B. (2011). Learning to count cells: applications to lens-free imaging of large fields. *Microscopic Image Analysis with Applications in Biology*, 1:3.

[29] Foulds, J. and Frank, E. (2010). A review of multi-instance learning assumptions. *The Knowledge Engineering Review*, 25:1–25.

[30] Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, pages 121–136.

[31] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, pages 193–202.

[32] Garcia, J. and Quintana-Domeque, C. (2007). The evolution of adult height in europe: a brief note. *Economics and Human Biology*, pages 340–349.

[33] Gaur, S., Sonkar, S., and Roy, P. P. (2015). Generation of synthetic training data for handwritten indic script recognition. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 491–495. IEEE.

[34] Gehler, P. and Nowozin, S. (2009). On feature combination for multiclass object classification. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 22–228. IEEE.

[35] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

[36] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

[37] Golik, P., Doetsch, P., and Ney, H. (2013). Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *INTERSPEECH*, pages 1756–1760.

[38] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

[39] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. *arXiv preprint arXiv:1302.4389*.

[40] Griffin, G., Holub, A., and Perona, P. (2007). Caltech-256 object category dataset. California Institute of Technology.

[41] group, C. (2008a).

[42] group, N. (12008b). Cuda.

[43] Guadarrama, S., Rodner, E., Saenko, K., Zhang, N., Farrell, R., Donahue, J., and Darrell, T. (2014). Open-vocabulary object retrieval. In *Robotics: science and systems*, volume 2, page 6.

[44] Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 993–1001.

[45] Haralick, R. M., Shanmugam, K., and Dinstein, I. H. (1973). Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*, pages 610–621.

[46] Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the theory of neural computation*, volume 1. Basic Books.

[47] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18:1527–1554.

[48] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[49] Hofmeyr, S. A., Forrest, S., and Somayaji, A. (1998). Intrusion detection using sequences of system calls. *Journal of computer security*, pages 151–180.

[50] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, pages 359–366.

[51] Hu, G., Peng, X., Yang, Y., Hospedales, T., and Verbeek, J. (2016). Frankenstein: Learning deep face representations using small data. *arXiv preprint arXiv:1603.06470*.

[52] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, pages 106–154.

[53] Jaderberg, M., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Synthetic data and artificial neural networks for natural scene text recognition. *arXiv preprint arXiv:1406.2227*.

[54] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE.

[55] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM.

[56] Kirk, D. et al. (2007). Nvidia cuda software and gpu parallel computing architecture. In *ISMM*, pages 103–104.

[57] Kong, D., Gray, D., and Tao, H. (2005). Counting pedestrians in crowds using viewpoint invariant training. In *BMVC*. Citeseer.

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

[59] LeCun, J., Cortes, C., and Burges, C. J. (1999). The mnist dataset of handwritten digits. *URL http://yann. lecun. com/exdb/mnist*.

[60] LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

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

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

[63] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, pages 2278–2324.

[64] LeCun, Y. et al. (1989b). Generalization and network design strategies. *Connections in Perspective. North-Holland, Amsterdam*, pages 143–55.

[65] LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., et al. (1995). Comparison of learning algorithms for handwritten digit recognition. In *Comparison of learning algorithms for handwritten digit recognition.*

[66] LeCun, Y., Kavukcuoglu, K., Farabet, C., et al. (2010). Convolutional networks and applications in vision. In *ISCAS*, pages 253–256.

[67] Lehmann, E. L. and Casella, G. (1998). Theory of point estimation (springer texts in statistics).

[68] Leibe, B., Schindler, K., and Van Gool, L. (2007). Coupled detection and trajectory estimation for multi-object tracking. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

[69] Lempitsky, V. and Zisserman, A. (2010). learn. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1324–1332. Curran Associates, Inc.

[70] Li, W., Mahadevan, V., and Vasconcelos, N. (2014). Anomaly detection and localization in crowded scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(1):18–32.

[71] Lindholm, E., Nickolls, J., Oberman, S., and Montrym, J. (2008). Nvidia tesla: A unified graphics and computing architecture. *IEEE micro*, pages 39–55.

[72] Mahadevan, V., Li, W., Bhalodia, V., and Vasconcelos, N. (2010). Anomaly detection in crowded scenes. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Francisco, CA, 2010.*

[73] Marana, A., Costa, L. d. F., Lotufo, R., and Velastin, S. (1998). On the efficacy of texture analysis for crowd monitoring. In *Computer Graphics, Image Processing, and Vision, 1998. Proceedings. SIBGRAPI'98. International Symposium on*, pages 354–361. IEEE.

[74] McCULLOCH, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5:115–133.

[75] Minai, A. A. and Williams, R. D. (1990). Acceleration of back-propagation through learning rate and momentum adaptation. In *Proceedings of International Joint Conference on Neural Networks*, pages 676–679.

[76] Mitchell, T. M. et al. (1997). Machine learning.

[77] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of machine learning*. MIT press.

[78] Moody, J., Hanson, S., Krogh, A., and Hertz, J. A. (1995). A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4:950–957.

[79] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814.

[80] Ng, A. and colleagues (2013). http://ufldl.stanford.edu/tutorial/supervised/ optimization-stochasticgradientdescent/.

[81] Nguyen, M. H., Torresani, L., de la Torre, F., and Rother, C. (2009). Weakly supervised discriminative localization and classification: a joint learning process. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1925–1932. IEEE.

[82] Parikh, D. and Zitnick, C. L. (2010). The role of features, algorithms and data in visual recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2328–2335. IEEE.

[83] Patel, G. H., Kaplan, D. M., and Snyder, L. H. (2014). Topographic organization in the brain: searching for general principles. *Trends in cognitive sciences*, pages 351–363.

[84] Perez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. In *ACM Transactions on Graphics (TOG)*. ACM.

[85] Phua, C., Lee, V., Smith, K., and Gayler, R. (2010). A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*.

[86] Pollock, R. J. (1996). *The automatic recognition of individual trees in aerial images of forests based on a synthetic tree crown image model*. PhD thesis, Concordia University.

[87] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[88] Preparata, F. P. and Shamos, M. (2012). *Computational geometry: an introduction*. Springer Science & Business Media.

[89] Rabaud, V. and Belongie, S. (2006). Counting crowded moving objects. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 705–711. IEEE.

[90] Rahmalan, H., Nixon, M. S., and Carter, J. N. (2006). On crowd density estimation for surveillance. In *Crime and Security, 2006. The Institution of Engineering and Technology Conference on*, pages 540–545. IET.

[91] Raykar, V. C., Yu, S., Zhao, L. H., Jerebko, A., Florin, C., Valadez, G. H., Bogoni, L., and Moy, L. (2009). Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *Proceedings of the 26th annual international conference on machine learning*, pages 889–896. ACM.

[92] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, pages 211–252.

[93] Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010*, pages 92–101. Springer.

[94] Seguí, S., Pujol, O., and Vitria, J. (2015). Learning to count with deep object features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 90–96.

[95] Senior, A., Heigold, G., Ranzato, M., and Yang, K. (2013). An empirical study of learning rates in deep neural networks for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6724–6728. IEEE.

[96] Shashua, A. (2009). Introduction to machine learning: Class notes 67577. *arXiv preprint arXiv:0904.3664*.

[97] Song, H. A. and Lee, S.-Y. (2013). Hierarchical representation using nmf. In *Neural Information Processing*, pages 466–473. Springer.

[98] Subramanian, S., Ozaltin, E., and Finlay, J. E. (2011). Height of nations: a socioeconomic analysis of cohort differences and patterns among women in 54 low-to middle-income countries. *PLoS One*, page e18962.

[99] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147.

[100] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

[101] Todorovic, S. and Ahuja, N. (2006). Extracting subimages of an unknown category from a set of images. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 927–934. IEEE.

[102] Van Der Walt, S., Schonberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., and Yu, T. (2014). scikit-image: image processing in python. *PeerJ*.

[103] Wang, S., Joo, J., Wang, Y., and Zhu, S.-C. (2013). Weakly supervised learning for attribute localization in outdoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3111–3118.

[104] Williams, D. R. G. H. R. and Hinton, G. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

[105] Willmott, C. J. and Matsuura, K. (2005). Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, page 79.

[106] Wu, B. and Nevatia, R. (2005). Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 90–97. IEEE.

[107] Xu, C., Pham, D. L., and Prince, J. L. (2000). Image segmentation using deformable models. pages 129–174.

[108] Yao, W., Basu, S., Lee, W.-n., and Singhal, S. (2015). Synthetic healthcare data generation. US Patent 20,150,370,992.

[109] Yu, X.-H., Chen, G.-A., and Cheng, S.-X. (1995). Dynamic learning rate optimization of the backpropagation algorithm. *Neural Networks, IEEE Transactions on*, pages 669–677.

[110] Yu, Y., Zhang, J., Huang, Y., Zheng, S., Ren, W., Wang, C., Huang, K., and Tan, T. (2010). Object detection by context and boosted hog-lbp. In *VOC Workshop Talk*, page 104.

[111] Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., and Oliva, A. (2014). Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*, pages 487–495.