UNIVERSITAT POLITÈCNICA DE CATALUNYA
UNIVERSITAT DE BARCELONA
UNIVERSITAT ROVIRA I VIRGILI

Master in Artificial Intelligence

Master of Science Thesis

MY MASTER THESIS TITLE

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)

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

The main purpose of this master thesis is to enhance feature detection and learning process by the use of deep learning algorithms as well as mitigating data annotation efforts by training deep Convolutional Neural networks with synthetically created dataset.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivations

The concept of learning to count is an important educational/developmental milestone which constitutes the most fundamental idea of mathematics. In Computer Vision[128], 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[138, 109, 69, 16, 117]. It arises in many real-world applications, including cell counting in microscopic images[41], monitoring crowds in surveillance systems[110, 131], and performing wildlife census or counting the number of trees in an aerial image of a forest[13, 106, 85].

Artificial intelligence and computer vision share topics such as pattern recognition and machine learning[93, 95] techniques. 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. 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, restrict their usage in large-scale computer vision applications thus demanding more efficient solutions to alleviate, expedite and improve this process.

One recent and commonly used method to facilitate feature detection process is application of deep Convolutional Neural Networks(CNN)[124, 71, 74, 119, 65, 125]. One of the promises of deep CNN is replacing handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction[121]. CNNs have been claimed and practically proven to achieve the most assuring performance in different vision benchmark problems concerning feature detection and classification[27, 124, 25].

1.2 Objectives

This work sets forward several objectives:

1. To apply deep CNN for feature detection and classification in learning to count problems where the concept of interest will be counted by no given explicit information about what we are counting to the system, except for its multiplicity in the image.

2. To demonstrate that deep features learned by deep CNN can be an appropriate surrogate for hand-crafted feature descriptors.In addition, tzo synthetically create datasets of images, as realistic as possible, and completely automatically annotated for CNN to train with.

3. To explore the behavior of proposed algorithm on synthetic datasets of different types and compare the performance with a state-of-the-art outcomes[117].

4. To analyze the performance of designed system on real world crowd counting problem and compare the results with state-of-the-art [16].

5. To apply deep CNN for feature detection and classification in a learning to count problem where the concept of interest will be counted by not giving explicit information about what we are counting to the system, except for its multiplicity in the image.

6. To synthetically create datasets of images, as realistic as possible, and completely automatically annotated for CNN to train with.

7. To explore the behavior of proposed algorithm on synthetic datasets of different types comparing the performance with a state-of-the-art outcomes[117].

8. To analyze the performance of designed system on real world crowd counting problem comparing the results with state-of-the-art results[16].

1.3 Contributions

This thesis contains the following contributions:

1. 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 pedestrians in the image that does not depend on object detection or feature tracking. The model is also privacy-preserving in a sense that it can be implemented with hardware that does not produce a visual record of the individuals in the scene.

2. It provides a synthetically generated and automatically labeled dataset of pedestrians using unlabeled University of California San Diego(UCSD) pedestrian dataset used in [90], to train a counting deep convolutional neural network which is adequate for apprehending the underlying representations of the learned features. To this end, we describe a counting problem for MNIST dataset to demonstrate the capability of the internal representation of the network for classifying digits with no direct supervising while training.

3. 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 training set. To our knowledge, this is the first crowd counting system trained by synthetic data that successfully operates on real data.

4. It provides a synthetically generated and automatically labeled dataset of pedestrians using unlabeled University of California San Diego (UCSD) pedestrian dataset used in [90], to train a counting deep convolutional neural network which is adequate for apprehending the underlying representations of the learned features. To this end, we describe a counting problem for MNIST dataset to demonstrate the capability of the internal representation of the network for classifying digits with no direct supervising while training.

5. 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 training set. To our knowledge, this is the first crowd counting system trained by synthetic data that successfully operates continuously on real data.

6. Along with the validation of our proposal in the following ways:

   - First, we learn to count even hand-written digits using MNIST dataset.
   - Second, we validate a more complex model quantitatively on a large synthetic dataset of pedestrians, containing 100,000 images with maximum 30 pedestrians in each image.
   - Last but not least, we count the number of pedestrians in a manually labeled dataset of images provided by Chan and Vasconcelos 15.

1.4 Organization

This report takes off with the review of Deep Learning(DL) (chapter 2.1) as a branch of Artificial Intelligence (AI) which deep convolutional neural networks belong to, and moves on to introduce a deep CNN's basic architecture and components in details (chapter 2.3).

Chapter 3 presents a review of state-of-the-art of feature detection and learning to count.

In section 4, we describe 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.

Section 6 revolves around the empirical experiments and analysis. Obtained results and comparisons drawn to state-of-the-art also is placed 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 deep Convolutional neural networks in question by describing the main components of CNNs.

2.1 Deep Learning

One of the central challenges of Artificial Intelligence (AI) 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*[50].

Modern deep learning provides a very powerful framework for 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[50].

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. Deep Learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound and text[127].

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 [5, 6].

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

main experts [6]. This is especially important, as noted by Bengio in [5], for domains where the features are hard to formalize, such as for object recognition and detection tasks.

In the task of object object detection, deep architectures have been widely used to achieve state-of-the-art results as in[70, 71], where the top published results use Convolutional Neural Networks [25]. 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 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 [59]. Artificial Neural Networks date back to 1943, with work by McCulloch [92]. 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 [143], 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.

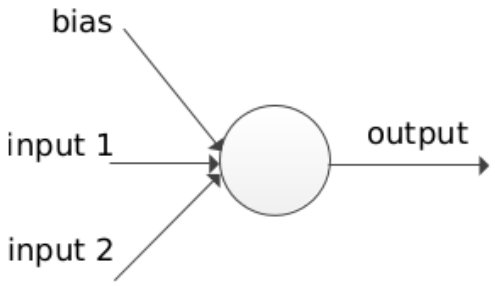


Figure 2.1: A single artificial neuron

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

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[75]. 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[9].

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[75]. A rectified linear unit has been illustrated in figure 2.2

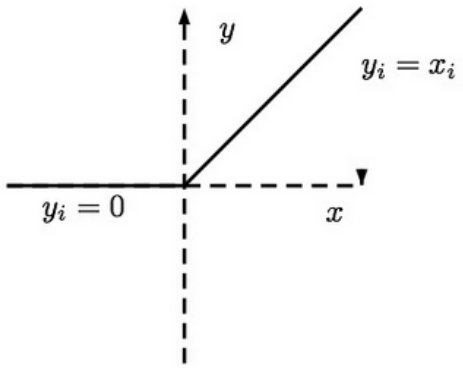


Figure 2.2: A Rectified Linear Unit (ReLU)

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 byGlorot et al. suggest that hard zeros can actually help supervised training[48].

2.2.2 Multi-layer Neural network

Models based on a single neuron, also called Perceptrons, have severe limitations. As noted by Preparata and Shamos, 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. 62, 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 3 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.

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

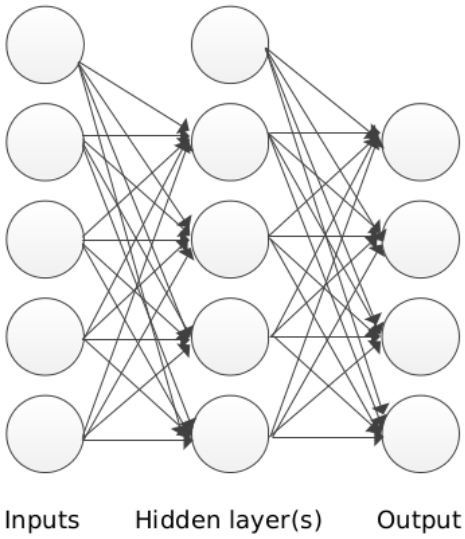


Figure 2.3: A neural network composed of three layers

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. 49, 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 (L2)loss 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 [77]. 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[143].

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* algorithm (SGD).

Stochastic gradient descent has often been proposed to minimize the *empirical risk* (training set performance measure. For more detailed description, see [134]) using *gradient descent*(GD) [11]. The standard gradient descent algorithm updates the parameter θ of the objective *J(θ)* with *learning rate* α[1] as:

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

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set (Empirical Risk Minimization[2] (ERM)). Stochastic Gradient Descent (SGD) 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:

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

with a pair $(x^{(i)}, y^{(i)})$ from the training set [99].

The SGD algorithm, as defined in [36], 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.

[1]The learning rate α is the weight of the negative gradient (will be described fully in section 5.1.2.2).

[2]Empirical risk minimization defines a family of learning algorithms and is used to give theoretical bounds on the performance of learning algorithms.

Algorithm 1 Stochastic Gradient Descent

Require: W, X, y, *batch_size*, α

$W \leftarrow$ random values

repeat

x_batch, $y\_batch \leftarrow$ next *batch_size* examples in (X, y)

$network\_state \leftarrow$ **ForwardProp**(W, x_batch)

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

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

$W \leftarrow W + \Delta W$

until Convergence_Criteria()

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. 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 [99]:

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[115]. The depth of an architecture refers to the number of 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 [5].

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 are composed of multiple levels of non-linear operations, such as those present in neural nets with many hidden layers[5]. figure 2.4 demonstrates a deep neural network.

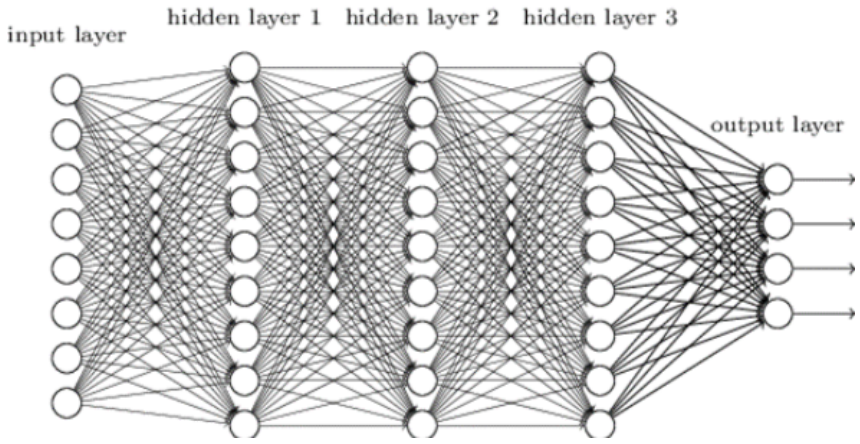


Figure 2.4: A deep neural network with three hidden layers

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 [60]. In 2006, Hinton et al. obtained good results using Restricted Boltzmann Machines (a generative model) to perform unsupervised training of the layers [60]. 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 [8]. 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 2.2.2.2) is used for fine-tuning the network. The advantage of unsupervised pre-training was demonstrated in several statistical comparisons [8, 72, 37], until recently, when deep neural networks trained only with supervised learning started to register similar results in some tasks (like object recognition). Ciresan et al. 25 demonstrate that properly trained deep neural networks (with only supervised learning) 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 2.4.

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. CNN are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [50]. This type of network was used to obtain state-of-the-art results in the CIFAR-10 object recognition task [25] and, more recently, to obtain state-of-the-art results in more challenging tasks such as the ImageNet Large Scale Visual Recognition Challenge [113]. 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[81]. 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[81]. 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[143]. 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.

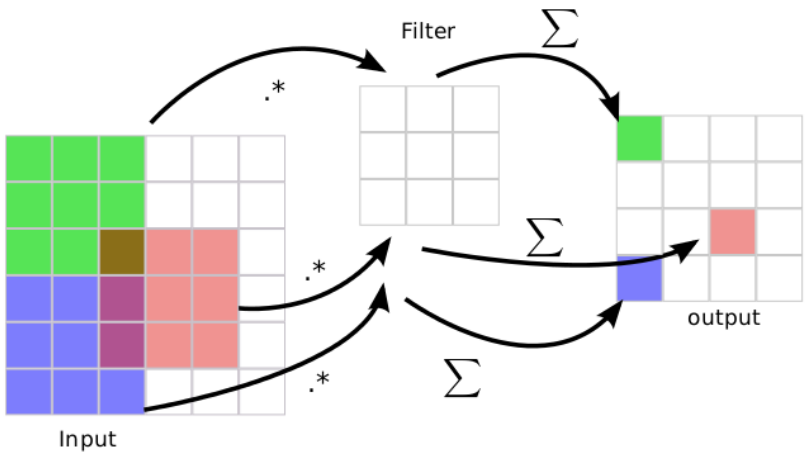


Figure 2.5: The forward propagation phase of a convolutional layer

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[81].

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:

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

l 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 [5]. 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.

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[79]. 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[81].

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 [129]. 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 [80], 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.

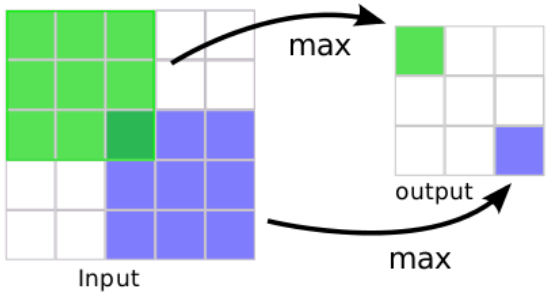


Figure 2.6: The forward propagation phase of a pooling layer with stride equal to 1.

Scherer et al. 114 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 [6]. Empirically, pooling has demonstrated to contribute to improved classification accuracy for object recognition [76].

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[71]. Figure 2.7 illustrates local response normalization in a CNN.

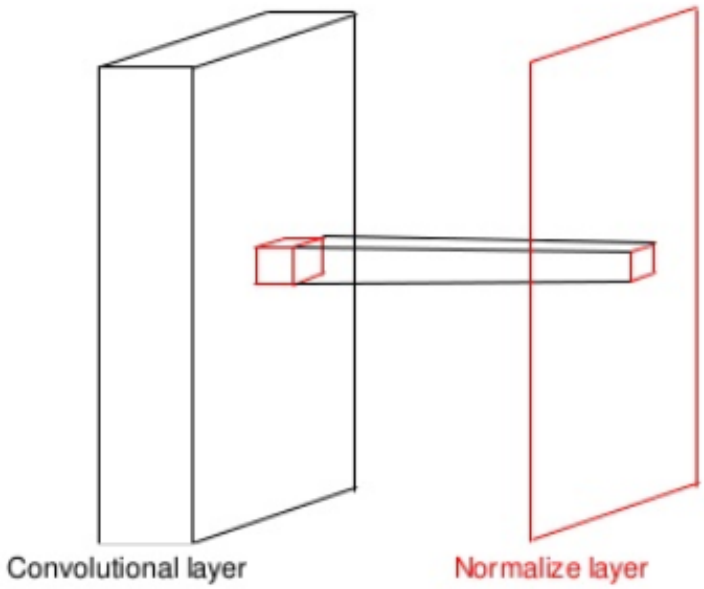


Figure 2.7: Local response normalization in a CNN after the convolution layer.

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 [64] which has led to error rate reduction in [71] and [61].

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*(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 [104].

3 State of the art review

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 [117]. In the field of object detection, numerous works have been previously proposed [102, 22, 112, 29, 69, 91, 137]. Most of these research works follow a taxonomy which consists of three paradigms underneath to count the objects:

1. Object detection, which are based on boosting appearance and motion features [138, 137], Bayesian model-based segmentation [150], integrated top-down and bottom-up processing [84, 101, 16].

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 moving people [109, 14, 16].

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 [102, 29], edge count [22, 112], or texture [91]; and finally estimating the crowd density or crowd count by a regression function, e.g. linear[102, 29], piece-wise linear [112], or neural networks[22, 112].

   In recent years, feature-based regression has also been applied to outdoor scenes. For example, Kong et al. 69 applies neural networks to the histograms of foreground segment areas and edge orientations. Dong et al. 34 estimates the number of people in each foreground segment by matching its shape to a database containing the silhouettes of possible people configurations, but is only applicable when the number of people in each segment is small (empirically, less than 6)[16].

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 145, Lin et al. 87, introduced methods to address this issue. Wu and Nevatia 145 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 [109, 14, 83], 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 on the individuals' privacy rights. While a number of methods that do not require explicit detection or tracking have been previously proposed [102, 22, 112, 29, 69, 91, 34], they

have not fully established the viability of the privacy-preserving approach[16]. The tension of privacy-preserving is common in all areas of data-mining [130, 136].

In order to tackle privacy preserving issue, Chan et al. 16 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 3.1:

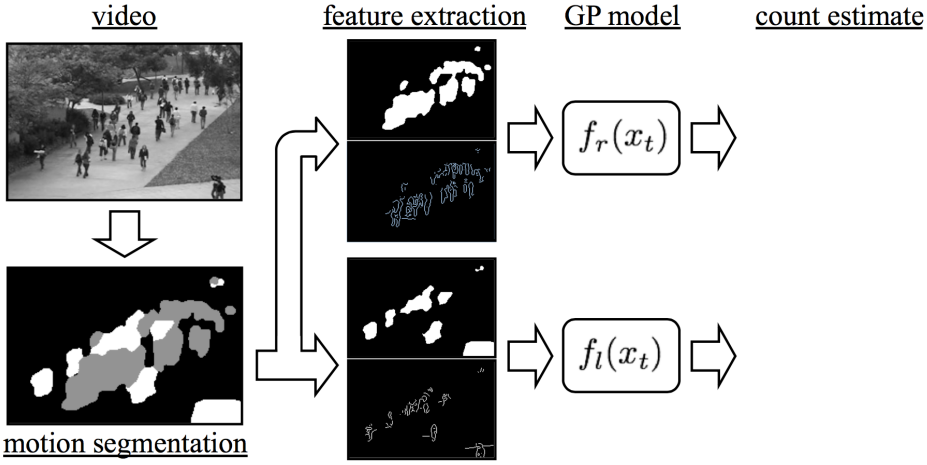


Figure 3.1: Crowd counting system: the scene is segmented into crowds with different motions. Normalized features that account for perspective are extracted from each segment, and the crowd count for each segment is estimated with a Gaussian process[16].

Chan et al. used a mixture of *dynamic textures*[1] [35, 18] 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 [35]. The mixture model is learned through Expectation-Maximization(EM) algorithm [18]. Video locations are then scanned sequentially, a patch is extracted at each location, and assigned to the mixture component of largest posterior probability. The location is declared to belong to the segmentation region associated with that component [16]. The resulting segmentations of their work are illustrated in figure 3.2:

[1]Dynamic textures are sequences of images of moving scenes that exhibit temporal regularity, intended in a statistical sense, like sea-waves, smoke, foliage, whirlwind but also talking faces, traffic scenes etc.

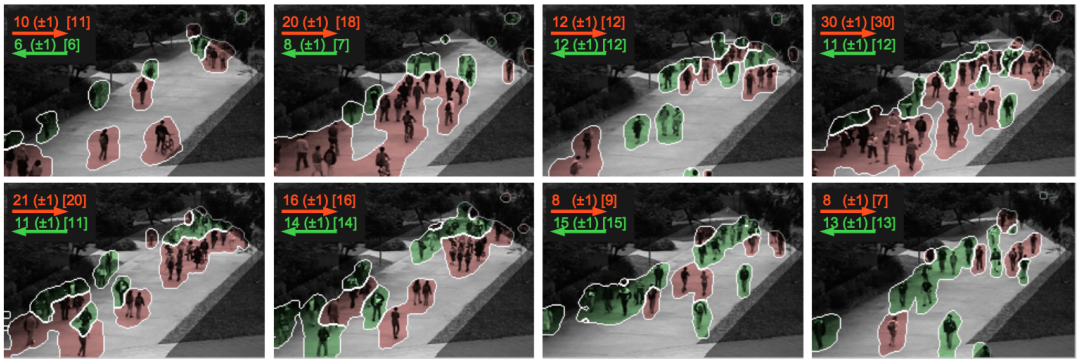


Figure 3.2: Crowd counting results: The red and green segments are the "away" and "towards" crowds. The estimated crowd count for each segment is in the top-left, with the (rounded standard-deviation of the GP) and the ground-truth. The Region Of Interest (the area in the walkway in which the pedestrians are counted and labeled) is also highlighted [16].

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 [29, 69]. For instance, total edge pixels and edge orientation.
- Texture features which are based on gray-level co-occurrence matrix(GLCM) (see [57] for more details) were applied for image patches classification into 5 classes of crowd density in [91]. 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 features from the segments extracted, a Gaussian Process(GP) [142] 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 [16]. Since the classes of function that GP can model is directly dependent on the chosen kernel function, they combined the linear and the squared-exponential(RBF)(see [21, 135, 120] 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 variety of factors, including occlusion, segmentation errors and pedestrian configuration (*e.g.* spacing within a segment) [16].

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 selected on the walkway(see figure 3.2), 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 [16].

The obtained results for crowd counting in [16] are expressed as both mean-squared-error(MSE) and mean-absolute-error(MAE) between the estimate and ground-truth. This reasonable results given the small dataset and also its' privacy-preserving manner notwithstanding, this work, like the aforementioned methods, requires not only exhaustive data annotations and large training set, but also hand-crafting highly specialized image features that are dependent on the object class.

In order to save annotation efforts, different techniques were used to count objects. Multiple Instance Learning(MIL) [42] 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*[141, 40]. There are different definitions for the term "weakly" in the literature. For instance, in [30], it is a surrogate for the concept of noisy labels such as labels provided by different supervisors with distinct quality. However, in [111], it is described for indicating imperfect annotation or even in [140] 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 for inferring difficult to describe classes such as in [126] where photometric, geometric, and topological features are recognized. More recently, several works, such as [100], explore the capacity of this technique for simultaneous localization and recognition. Another work using MIL framework was count-based multiple instance learning [42]. 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 [41], 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 [149, 38, 47, 58, 38] 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[117]. 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 [151].

Following this line of work, Seguí et al. in [117] 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 [96]. 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 [117].

To this end, they defined a counting problem for even digits using *MNIST* data 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 [117]. Figure 3.3 illustrates their proposal at a glance in the case of representing hand-written digits:

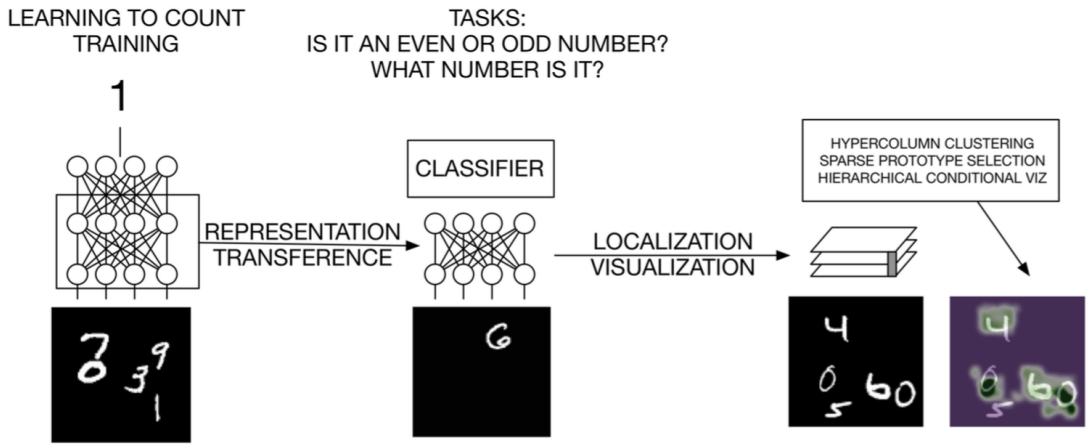


Figure 3.3: Learning to count hand-written digits problem in which the features of a CNN that has been trained to count digits can be readily used for more specific classification problems and even to localize digits in an image[117].

In [117], 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 [76]) 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 [71, 67, 27])[117].

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, which are incredibly promising, validates this hypothesis that counting process can be considered as a surrogate to potentially extract or infer interesting object descriptors [117].

Additionally, for learning to count the number of pedestrians in a scene, they used UCSD pedestrian database [16]. Once again, Seguí et al. 117 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 still the model be able to learn object representations in scaled-up and more complex scenarios?

4 Methodology

This master thesis proposes an application of deep CNNs to the task of counting objects in the image. The developed systems will address some of the issues common to other object detection algorithms. These issues include:

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

In addition, the state-of-the-art [117] 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 hypothesize that features learned by training a deep CNN 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 our methodology along with a comparison to state-of-the-art from different aspects including method selection, architecture, dataset and its application.

4.1 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 [89], HOG [28], SURF [4], 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) [133, 10]. In fact, for the most part, these features are not learned, but hand-crafted by some vision experts. However, they do indeed demonstrate a decent performance. For instance, in one of the most successful works in object detection [39], the author essentially introduces a linear classifier on top of HOG features, or regarding classification approaches that work quite well, Yu et al. use all manners of features (HOG, SIFT, Color SIFT, etc) extracted from the images and consequently obtain impressive results.

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 [103] analyze the role of features by taking some of the quite successful past deformable models [1] (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 [146]), and replace some components of them with humans. They present identical learning tasks *i.e.* the same feature representation and the same training data to machines and humans which allows drawing a comparison between the two. The author concludes that features are the main factor contributing to superior human performance.

Furthermore, in [46], the authors compared 39 different learning kernels with various combination of features to see which kernel outperforms the rest and how it should be weighted. Although they attained a big improvement over the existing methods, the analysis of their results 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 learning in order to learn the features themselves rather than hand-crafting them.

In deep learning techniques, we essentially have a hierarchy of feature extractors which attempts to model high-level abstractions in data [32, 5, 7, 3, 116], where each layer of hierarchy extracts features from output of previous layer. Designing such trainable feature extractors, we would be able to build a multi-stage model which goes all the way from image pixels up to a high-level feature vector which we can be fed to a standard classifier.

Thus, from the family of DL, 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 [151]. Moreover, compare to other methods, CNN benefit from the powerful speed-ups on GPU which is fit for implementation of CNN.

Furthermore, CNN was used to achieve state-of-the-art results in CIFAR-10 object recognition task [24], in [113], the training process was done on GPU, and a great success in many other recent vision problems [26, 23, 139]. All these features and advantages of CNN assured us to choose convolutional neural networks as our proposed method

4.2 Architecture

With their long heritage, the origins of CNN trace back to [63] where first simple cells (brain cell) detect local features and afterwards, complex cells pool the outputs of simple cells within a retinotopic[1]neighborhood (see [63] for more detailed explanation). Also, Fukushima in [43, 44] introduced a similar architecture with filtering layers followed by pooling layers. However, the first deep CNN architecture was designed by LeCun et al. 76 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 4.1

[1]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

Figure 4.1: Deep CNN proposed by LeCun et al. 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) [98] 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 [56, 2] 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 [12]. In our model we use *spatial* pooling. However, depending on the problem we are trying to solve, pooling might be done within feature maps [51]. The main advantage of pooling layers in the architecture can be that it results in invariance to small transformations. In addition to that, as we go up in CNNs, 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 [71]. Empirically, using LRN in the architecture seems to help improving the results [64, 71].

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 4.2 illustrates our proposed design of deep convolutional neural network with just one convolutional layer in order

to portray the basic structure we are going use in this work.

Figure 4.2: The proposed design for a CNN with just one convolution layer.

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 CNNs' capability to learn features is the deepness and settings of the architecture which will be attentively addressed later on (chapter 5.2).

4.3 Datasets

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

In spite of remarkable performance of CNN in some simple recognition benchmarks [26, 23, 139, 24], until recently the models' performances were poor at more complex datasets [52] due to lack of labeled input samples to train the network parameters with. It was with the creation of ImageNet [31] and GPU implementation [71] (50× speedup over CPU) that efficiency of deep CNN in vision tasks became crystal clear.

4.3.1 MNIST pool of 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 [73]. In the MNIST dataset, each image contains one hand-written digit of dimension 28×28. Figure 4.3 shows a selection of MNIST dataset images.

Figure 4.3: 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.

4.3.2 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.

4.3.3 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 [16] to validate the performance of our model on it and also make a comparison with their system in [16] 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.

5 Implementation

In this chapter, we provide a detailed implementation of our proposed methodology. We start with presenting the software platform we incorporated to shape and design. Then we demonstrate our network's architecture in detail. Finally, we attempt to give insight into the datasets we used to train and test our model.

5.1 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 [66]. 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.

5.1.1 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 5.1 shows the blobs connected to a convolution layer implemented by Caffe.

[1] CUDA is a parallel computing platform and application programming interface (API) model created by NVIDIA [54], processing over 40 million images a day on a single K40 or Titan GPU [66].

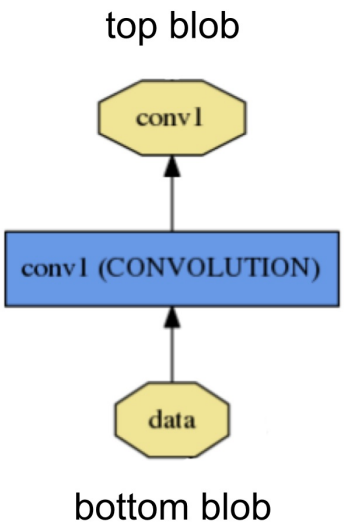


Figure 5.1: Input and output blobs of a convolution layer in a CNN implemented in Caffe.

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 [53].

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 [66].

   Caffe supports an exhaustive set of layers, including the followings[66]:

   (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

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 5.2 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 [33], object detection [47], and object retrieval [55], [66].

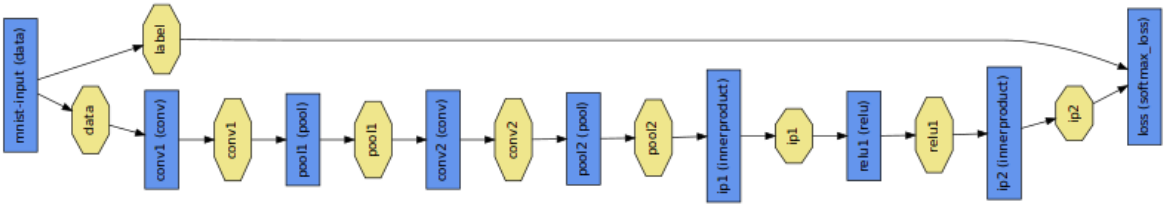


Figure 5.2: An MNIST digit classification example of a Caffe network, where blue boxes represent layers and yellow octagons represent data blobs produced by or fed into the layers[66].

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[66]. In addition, it provides a python wrapper which exposes the solver module for easy prototyping of new training procedures.

5.1.2 Model Optimization

Our work in general is an optimization problem since we project the results as loss function we try to minimize. For this reason, model optimization methods have a critical impact on the performance of the model. Since Caffe provides several optimization methods, in this section we briefly describe merely optimization methods and hyper-parameters incorporated in our proposed algorithm.

In Caffe, *Solver* file orchestrates model optimization by coordinating the network's forward inference and backward gradients to form parameter updates that attempt to improve the loss.

5.1.2.1 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, 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 the network at each iteration. The larger the batch size, the more accurate the gradient estimate at each iteration will be.

5.1.2.2 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.

- ***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 thumbs to treat learning rate [118, 147, 94], basic learning rate and learning policy are highly problem-dependent.

5.1.2.3 Weight Decay

As a part of BP algorithm 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 a 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 right, a weight decay can suppress some of the effects of static noise on the targets, which improves generalization significantly[97].

5.1.2.4 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:

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

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

where $\alpha > 0$ is the learning rate, $\mu \in [0, 1]$ is the momentum coefficient, and $\bigtriangledown f(\theta_t)$ is the gradient at θ_t [123].

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[99]. 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 [99].

5.1.2.5 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 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.

5.2 The architecture

In learning features or object representations for vision tasks and by the use of neural networks, the depth of network plays an important role. The deeper the model, the better it learns. However, issues like overfitting[2] and underfitting should not be left neglected. Having Caffe platform introduced, we propose the designed network for two experiments we did regarding learning to count problems. Therefore, in this section, networks' settings and architectures for even digit recognition and crowd counting problems will be described separately.

5.2.1 Even digit counting

For learning to count even digits problem, since we used MNIST dataset to generate our dataset, we decided to start with an architecture similar to the classic MNIST hand-written digit recognition problem [80]. From there, we modified the architecture to optimize the performance of the network.

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*[3] $= 1$ 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. The networks' parameters are set as below:

- **Learning rate:** The basic learning rate is 0.0001. However, for our experiment we chose *multi-step* learning policy in which, after each *stepsize*=40000 iterations, the learning rate

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

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

drops by the rate of Gamma $\gamma = 0.1$. This initialization is based on rule of thumb used in [71].

- **Momentum:** We use momentum $\mu = 0.9$. This selection also is based on the rule of thumb. As momentum value 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 $\gamma = 0.1$.

- **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.

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 5.3 shows a schematic of the architecture.

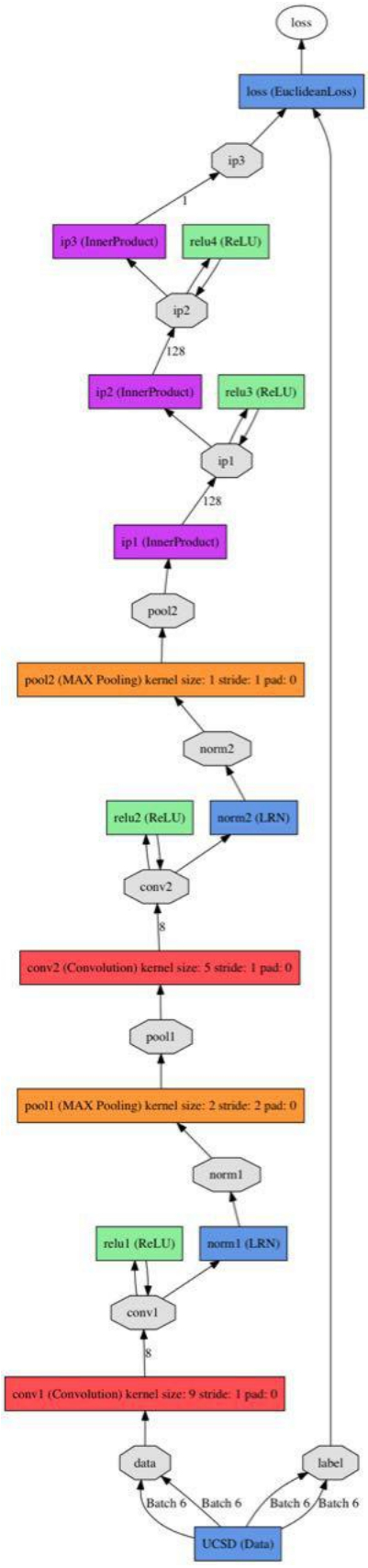


Figure 5.3: Proposed network architecture for Even digits recognition task

5.2.2 Crowd counting

In the case of counting pedestrians task, we applied the same settings to a different architecture. This time, due to more complexity of images, we considered a deeper network. Here the data blobs pass through 4 convolutional layers. 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 $= 0$.

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. In figure 5.4, an illustration of this architecture has been presented.

To the best of our knowledge and experience, the designed architectures outperform other architectures by which we examined models' performance (a different range of architectures starting from 1 up to 5 convolutional layers with different settings), while fasten 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.

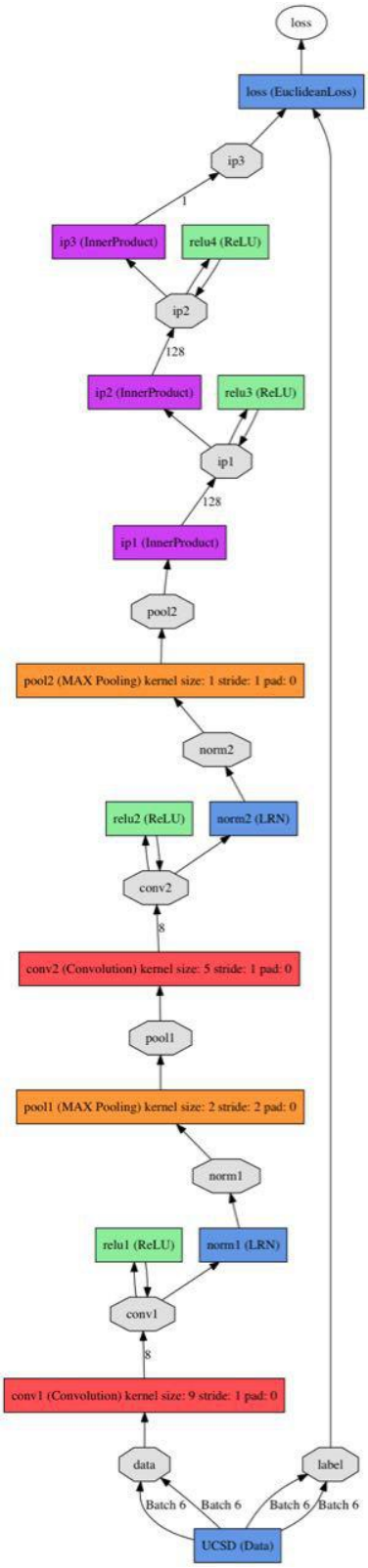


Figure 5.4: Proposed network architecture for Even digits recognition task

5.3 The datasets

Now, we delve into the data processing part of this work by introducing three different datasets we generated or chose for our empirical experiments. To that end, we provide a detailed explanation of the approaches and methods used to generate and improve each dataset.

5.3.1 Even-odd digits dataset

Our Even-odd handwritten dataset contains images of size 100×100. Each image is filled with 0 up to 15 randomly selected digits from MNIST dataset. Digits are resized to 18×18 pixels and randomly put in the image. 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. For the training process, images are labeled with the number of even digits present in each image. Figure 5.5 illustrates some examples of even-odd digits dataset with different number of even digits in images.

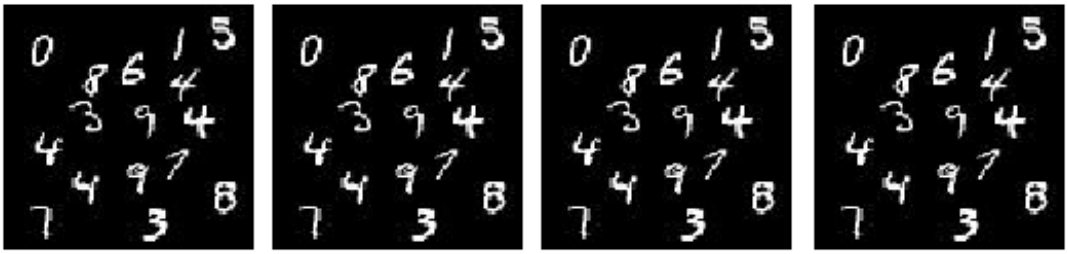


Figure 5.5: An example of even-odd digits images. Form left to right, images contain 0, 5, 10 and 15 even digits.

This dataset has in total 1 million images, 800,000 images for training set and 200,000 as the test. Also, the dataset is uniformly generated, meaning that for instance, the number of images containing 0 even digits are equal to the number of images containing 15 even digits.

5.3.2 Synthetic pedestrians dataset

Learning features using deep architectures require 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 soften this cost, in our research, we decided to synthetically create a 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 [17, 90, 86]. UCSD Anomaly detection dataset contains clips of groups of people walking towards and away from the camera, and some amount of perspective distortion. Contains 34 training video samples and 36 testing video samples. Each video has 200 frames of each 238×158 pixels. Figure 5.6 depicts some images of UCSD Anomaly dataset.

Figure 5.6: 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 pore over the generation of our dataset, we divide this process into data generation and data improvement.

5.3.2.1 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:

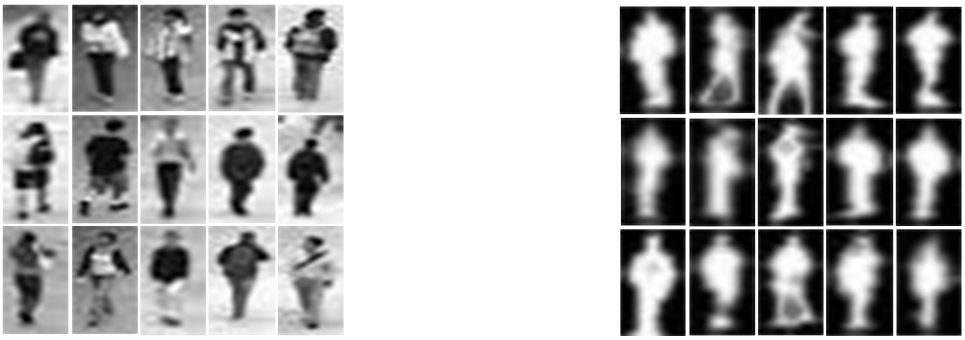
1. **Background extraction:** Firstly, we simply subtract the background from each video frame. We extract two types of backgrounds, the median of all the backgrounds in each video (in total, 70 different backgrounds), and the median of all median backgrounds. An example of extracted backgrounds is shown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

Figure 5.7: 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 of the subtraction using morphological labeling methods. Figure **??** shows how images look like after background subtraction.

Figure 5.8: Images after background subtraction.

Then, properties of labeled regions are measured and bound-boxed (see [132] 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. Below figure demonstrates a selection of extracted pedestrians and pedestrians' masks.



(a) Extracted pedestrians from different video frames.

(b) created masks corresponding to the extracted pedestrians.

Figure 5.9: Pedestrians and their corresponding masks.

3. **Background generation:** In this step, we tried to make the backgrounds of images as realistic as we could, by first, make a sparse combination of median backgrounds, secondly, changing the global illumination of the images randomly and at last, adding some *Gaussian noise* to the backgrounds. As you may notice, in the figure 5.10, the generated background happened to be brighter and noisier than the extracted one.

4. **Generating the images:** 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.

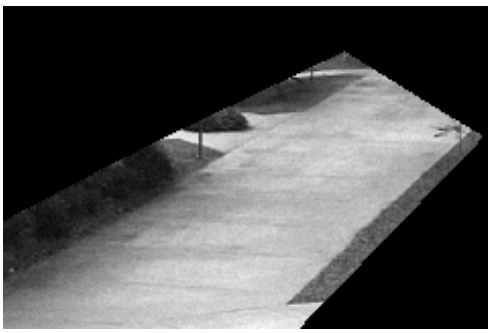
   Afterwards, pedestrians are added to the masked background in a way that the center of each person is placed inside white area of the mask. Finally images are normalized and resized to 158×158 in order to be fed to convolution layers. A selection of created images are depicted in the figure underneath.

(a) Raw extracted background image from the previous step.

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

Figure 5.10: A synthetically generated background image

(a) The mask to obtain region of interest.



(b) Region of interest (ROI). The pedestrians will only be put and counted in the non-zero pixels.

Figure 5.11: Applying the mask of region of interest on the background image.

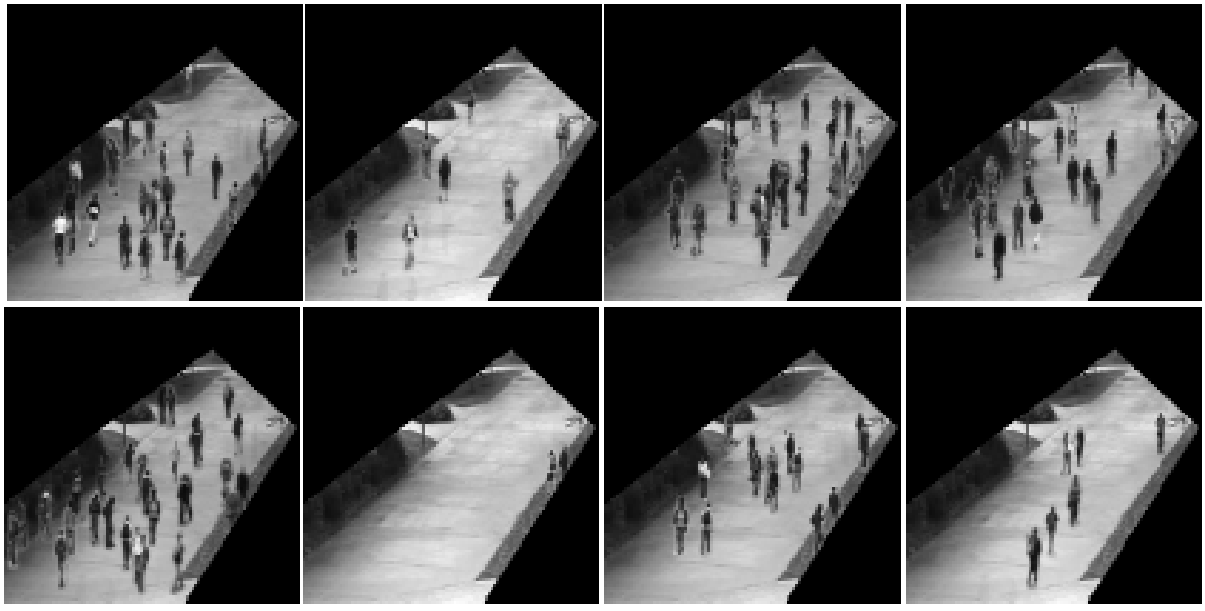


Figure 5.12: created synthetic images for counting pedestrians problem.

5.3.2.2 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 better and enrich images, we improved the dataset in the aspects explained underneath:

- **Non-pedestrian objects:** Amongst the extracted boxes of pedestrians, there were some non-pedestrian boxes. Some with objects instead of pedestrians and 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 5.13: A selection of non-human or incorrectly labeled objects.

- **Lack of pedestrians:** For the sake of generalization, we needed a decent variety of pedestrians in the images to train with. To that end, 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 the pedestrians were extracted from, were still remained around the pedestrians. In the new images, depending on where the person was placed, these thin layers appeared like a halo around the person. Figure **??** illustrates some images with halos around the pedestrians.

Figure 5.14: 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* [132] 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 [105]. 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 **??**.

Figure 5.15: 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, 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 5.16: created images before considering image perspective.

Human height almost follows a Gaussian distribution [122]. Therefore, with respect to [122, 45], 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 in below.

Figure 5.17: Synthetic images considering image perspective based on the real distribution of people's height.

As you may see, considering perspective in the images, the images and people's position in 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.

5.3.3 UCSD crowd counting dataset

To verify and validate our model, we used UCSD crowd counting dataset created by Chan et al. and used in [16, 19, 20]. 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 are shown in figure **??**.

Figure 5.18: Normal UCSD crowd counting dataset images.

Among the labeled images, we selected the ones in which the number of pedestrians do 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. At last, the images look like the following images.

Figure 5.19: Real UCSD images after being masked and resized.

We will use to first, validate the performance of our model trained with synthetic data, on a real dataset, and also to comparison between work done in [16] and our approach.

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. This chapter contains experiments details, obtained results, comparison to similar works and finally a conclusion about the performance of our models in each of the experiments.

6.1 Learning to count Even-odd hand-written 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 of the digits to be applied for digits 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 apply a regression strategy 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 5.3.1, 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 are 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 do 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 5.2.1, for learning to count the number of even digits, we designed a deep CNN. Table 6.1 shows a review of the network's specification and architecture.

Table 6.1: The deep CNN with 2 convolution layers to count the number of even digits.

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

However, a well-designed network solely cannot guarantee an optimal performance for the model. The responsibilities of learning are divided between the Net 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, we solver parameters need to be set attentively

in order to optimize the model performance. Table 6.2 summarizes the initial settings for the solver parameters while the upcoming items justify our selections.

Table 6.2: The solver parameters settings for even digit counting problem.

Optimization parameters	
Parameter	**value**
base_lr	0.0001
batch size	256
momentum	0.9
weight decay	0.0005
step size	40000
gamma(γ)	0.1
iterations	1,600,000

- **Learning rate:** The basic learning rate is 0.0001. However, for our experiment we chose *multi-step* learning policy in which, after each *stepsize*=40000 iterations, the learning rate drops by the rate of $\gamma = 0.1$. This initialization is based on rules of thumbs used in [71].
- **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.
- **Momentum:** We use momentum $\mu = 0.9$. This selection also is based one rules of thumbs. 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

The algorithm trained on a GPU NVIDIA [68] Tesla K40 [88].It took almost 4 days to train 800,000 samples and tested over 200,000 images. The learning curves of this learning process are shown in the figure 6.1

Figure 6.1

ADD EXPLANATION OF THE LEARNING CURVES
Moreover, figure 6.2 illustrates the features learned from one input sample at each convolutional layer in the network.

Figure 6.2

ADD EXPLANATION OF THE FEATURES
A SHORT TRANSITIVE PARAGRAPH TO GO TO RESULTS

6.1.3 Experimental results

In addition to the Euclidean loss function 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 [82]. 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 at 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 [144]. 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:

Table 6.3: Table caption font is different from the normal text font in order to get a better differentiation – I like it that way.

Results

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

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

Figure 6.3

As you may observe in Figure 6.7, 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
Euclidean loss	0.20	0.30
Mean squared error	0.20	0.30

Table 6.4: Table caption font is different from the normal text font in order to get a better differentiation – I like it that way.

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, is the model applicable for a real-world crowd counting scenario? We believe that this experiment will enlighten and address these questions.

6.2.1 Datasets

As fully discussed in chapter 5.3.2, for this 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 problem, we used UCSD crowd counting dataset [16] 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 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 5.2.2, 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 6.5.

Table 6.5: The deep CNN with 2 convolution layers to count the number of even digits.

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

The model's optimization method and parameters (solver parameters) are set similar to the learning to count problem. The network trains with stochastic gradient descent and our justification for the setting the solver parameters remains the same previous experiment (explained in section 6.1.2). Therefore, the repetitive explanations are skipped and only the final settings are described in table 6.6.

Table 6.6: The solver parameters settings for even digit counting problem.

Optimization parameters	
Parameter	**value**
base_lr	0.0001
batch size	256
momentum	0.9
weight decay	0.0005
step size	40000
gamma(γ)	0.1
iterations	1,600,000

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

Figure 6.4

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 6.5

And finally, the network along with the outputs of different layers is shown in a schematic form in figure 6.6.

Figure 6.6

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

These results were attained after 1,600,000 iterations on 800,000 train and 200,000 test sets. Correspondingly, figure 6.7 depicts the spread-error plot of the model performance on the test set.

Figure 6.7

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 [16]. 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 [16] 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 various contributions to the object representation problem casted as learning to count tasks. Our contributions include 1) a deep algorithm to sufficiently learn object representations without the need for hand-crafted feature detectors. 2) counting th number of pedestrians in images in a privacy-preserving manner involving no object tracking or detection techniques. 3) two synthetically generated and automatically annotated datasets for even-odd hand written digit recognition and crowd counting in the images. 4) a deep architecture trained with synthetic data which is capable of counting pedestrians in a real-world problem. 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, where we hypothesized that deep features learned by deep convolutional neural network can be a surrogate for object detection tasks using manually designed feature detectors. To this end, we configured a deep CNN for an even-digits counting problem. We trained the network with synthetic dataset of up to 15 MNIST digits in the images with the number of even digits as labels (chapter 5.3.1). Having a deep model trained, we obtained noteworthy results implying applicability of deep CNN as a surrogate for previous feature detection approaches.

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 depends upon a large amount of 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 5.3.2). Designing another deep network, once again we achieved satisfactory results while discarding extensive labeling effort and hand-crafting feature detectors.

One 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. To this end, we tested our model which was trained with synthetic dataset, on a similar but real relatively small set of images which were labeled manually with the number individuals in the image(see section 5.3.3).

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.

To sum up, the findings of this Master thesis suggest deep learning techniques as a well-suited surrogate for previous learning to count problems by showing representativity of the learned features for the object of interest in similar but different tasks. It also proposes the usage of synthetic dataset to ease the annotation effort in supervised learning approaches. Last but not least, 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 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 thesis, we have observed and pointed out shortcomings and areas in which one can improve our work. Some of the deficits of our study along with some hints to advance and enrich this line of research which were out of scope of this work, are mentioned in the followings:

1. We generated synthetic gray-scale datasets to tackle our proposed problems. Although we did our best 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 to improve the performance of our crowd counting model on the real dataset is to couple two architecture, 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 approach could be 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 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 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. Sue 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.

References

[1] Albrecht, T., Luthi, M., and Vetter, T. (2015). Deformable models. *Encyclopedia of Biometrics*, pages 337–343.

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

[3] Arel, I., Rose, D. C., and Karnowski, T. P. (2010). Deep machine learning-a new frontier in artificial intelligence research [research frontier]. *Computational Intelligence Magazine, IEEE*, pages 13–18.

[4] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer.

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

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

[7] 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.

[8] 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.

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

[10] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.

[11] Bottou, L. (2010). Large-scale machine learning with stochasticgradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

[12] 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.

[13] Brandtberg, T. and Walter, F. (1998). Automated delineation of individual tree crowns in high spatial resolution aerial images by multiple-scale analysis. *Machine Vision and Applications*, 11(2):64–73.

[14] Brostow, G. J. and Cipolla, R. (2006). Unsupervised bayesian detection of independent motion in crowds. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 594–601. IEEE.

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

[16] 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.

[17] 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.

[18] Chan, A. B. and Vasconcelos, N. (2008). Modeling, clustering, and segmenting video with mixtures of dynamic textures. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(5):909–926.

[19] 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.

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

[21] Chang, Y.-W., Hsieh, C.-J., Chang, K.-W., Ringgaard, M., and Lin, C.-J. (2010). Training and testing low-degree polynomial data mappings via linear svm. *The Journal of Machine Learning Research*, 11:1471–1490.

[22] Cho, S.-Y., Chow, T. W., and Leung, C.-T. (1999). A neural-based crowd estimation by hybrid global learning algorithm. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(4):535–541.

[23] 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.

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

[25] 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.

[26] Cireşan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2011). Convolutional neural network committees for handwritten character classification. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1135–1139. IEEE.

[27] Ciresan, D. C., Meier, U., Masci, J., Maria Gambardella, L., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, page 1237.

[28] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

[29] Davies, A. C., Yin, J. H., and Velastin, S. A. (1995). Crowd monitoring using image processing. *Electronics & Communication Engineering Journal*, 7(1):37–47.

[30] 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.

[31] 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.

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

[33] 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*.

[34] 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.

[35] Doretto, G., Chiuso, A., Wu, Y. N., and Soatto, S. (2003). Dynamic textures. *International Journal of Computer Vision*, 51:91–109.

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

[37] Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., and Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. In *International Conference on artificial intelligence and statistics*, pages 153–160.

[38] 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.

[39] 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.

[40] Fergus, R., Perona, P., and Zisserman, A. (2003). Object class recognition by unsupervised scale-invariant learning. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, pages II–264. IEEE.

[41] 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.

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

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

[44] 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.

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

[46] 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.

[47] 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.

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

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

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

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

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

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

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

[55] 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.

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

[57] 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.

[58] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37(9):1904–1916.

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

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

[61] 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*.

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

[63] 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.

[64] 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.

[65] Ji, S., Xu, W., Yang, M., and Yu, K. (2013). 3d convolutional neural networks for human action recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):221–231.

[66] 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.

[67] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

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

[70] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

[71] 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.

[72] Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM.

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

[74] 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.

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

[76] 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.

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

[78] LeCun, Y., Cortes, C., and Burges, C. J. (1998b). The mnist database of handwritten digits.

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

[80] 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.*

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

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

[83] 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.

[84] Leibe, B., Seemann, E., and Schiele, B. (2005). Pedestrian detection in crowded scenes. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 878–885. IEEE.

[85] 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.

[86] 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.

[87] Lin, S.-F., Chen, J.-Y., and Chao, H.-X. (2001). Estimation of number of people in crowded scenes using perspective transformation. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 31(6):645–654.

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

[89] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee.

[90] 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*.

[91] 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.

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

[93] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.

[94] 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.

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

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

[97] 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.

[98] 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.

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

[100] 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.

[101] Oliva, A., Torralba, A., Castelhano, M. S., and Henderson, J. M. (2003). Top-down control of visual attention in object detection. In *Image processing, 2003. icip 2003. proceedings. 2003 international conference on*, volume 1, pages I–253. IEEE.

[102] Paragios, N. and Ramesh, V. (2001). A mrf-based approach for real-time subway monitoring. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–1034. IEEE.

[103] 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.

[104] 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.

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

[106] 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.

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

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

[109] 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.

[110] 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.

[111] 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.

[112] Regazzoni, C. S. and Tesei, A. (1996). Distributed data fusion for real-time crowding estimation. *Signal Processing*, 53(1):47–63.

[113] 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.

[114] 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.

[115] Schmidhuber, J. (2015a). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

[116] Schmidhuber, J. (2015b). Deep learning in neural networks: An overview. *Neural Networks*, pages 85–117.

[117] 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.

[118] 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.

[119] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.

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

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

[122] 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.

[123] 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.

[124] 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.

[125] Taylor, G. W., Fergus, R., LeCun, Y., and Bregler, C. (2010). Convolutional learning of spatio-temporal features. In *Computer Vision–ECCV 2010*, pages 140–153. Springer.

[126] 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.

[127] Tutorial, D. L. (2014). Lisa lab. *University of Montreal*.

[128] Umbaugh, S. E. (1997). *Computer Vision and Image Processing: A Practical Approach Using Cviptools with Cdrom*. Prentice Hall PTR.

[129] University, S. (2013). http://white.stanford.edu/teach/index.php/an_introduction_to_convolutional_neural_networks.

[130] Vaidya, J., Clifton, C. W., and Zhu, Y. M. (2006). *Privacy preserving data mining*, volume 19. Springer Science & Business Media.

[131] Valera, M. and Velastin, S. A. (2005). Intelligent distributed surveillance systems: a review. In *Vision, Image and Signal Processing, IEE Proceedings-*, volume 152, pages 192–204. IET.

[132] 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*.

[133] Vapnik, V. and Chervonenkis, A. (1964). A note on one class of perceptrons. *Automation and remote control*, 25(1).

[134] Vapnik, V. N. and Vapnik, V. (1998). *Statistical learning theory*, volume 1. Wiley New York.

[135] Vert, J.-P., Tsuda, K., and Schölkopf, B. (2004). A primer on kernel methods. *Kernel Methods in Computational Biology*, pages 35–70.

[136] Verykios, V. S., Bertino, E., Fovino, I. N., Provenza, L. P., Saygin, Y., and Theodoridis, Y. (2004). State-of-the-art in privacy preserving data mining. *ACM Sigmod Record*, 33(1):50–57.

[137] Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *International journal of computer vision*, 57(2):137–154.

[138] Viola, P., Jones, M. J., and Snow, D. (2005). Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161.

[139] Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066.

[140] 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.

[141] Weber, M., Welling, M., and Perona, P. (2000). *Unsupervised learning of models for recognition*. Springer.

[142] Williams, C. K. and Rasmussen, C. E. (2006). Gaussian processes for machine learning. *the MIT Press*, 2(3):4.

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

[144] 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.

[145] 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.

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

[147] 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.

[148] 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.

[149] Zhang, Y., Sohn, K., Villegas, R., Pan, G., and Lee, H. (2015). Improving object detection with deep convolutional networks via bayesian optimization and structured prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 249–258.

[150] Zhao, T. and Nevatia, R. (2003). Bayesian human segmentation in crowded situations. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–459. IEEE.

[151] 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.