UCLouvain

École polytechnique de Louvain

Crowd counting

Applied in UCLouvain auditoriums

Authors: **Henri Collin, Louis Robins**
Supervisor: **Pierre Schaus**
Readers: **Benoît Macq, Jean Gillain**
Academic year 2021–2022
Master [120] in Computer Science and Engineering

Acknowledgments

We would like to thank our supervisor Pr. Pierre Schaus for his help throughout the year. The frequent meetings as well as his ideas and advices kept us motivated and moving forward with the thesis even during the Erasmus and internships.

We would also like to thank Christophe Poncin who permitted us to have access to the security cameras and who was always available to answer our questions when we had a problem.

Finally, we would like to thank our families for supporting us all the way.

Contents

List of Tables

List of Figures

Introduction

Crowd counting is the act of counting, estimating how many people are present in a crowd. With the growing number of people in the world, crowd counting has become more and more useful over the years. Until recently, it was done manually by human people with a poor precision and very different numbers, for example in the strikes in France.

There are several methods for estimating the number of people in a place. One can use a CO2 captor, look at the number of users connected to the Wi-Fi or even place sensors at doors to detect entry and exit. With the recent advances in the domain of machine learning, another method has emerged : crowd counting based on images. Indeed, counting the number of people on crowded scenes has been the subject of various studies, especially in China, where the density of population is really high. Different models have been developed and tested to count people on an image, with small error margin.

Crowd counting at UCLouvain

The University of Louvain-la-neuve can also benefit from crowd counting analysis since they installed security cameras in most auditoriums. Counting the number of people in an auditorium can have several direct applications. The main one is to make a correct assignation of the auditoriums. In fact, the number of students going to the lectures is not easy to estimate and some auditoriums are overcrowded while some others are empty. During the Covid-19 pandemic, since the number of people in auditoriums was limited due to social distancing, having the best optimization of the available study spaces was even more important. More than allocating the auditoriums, using the cameras for counting the people attending a course can help for a better allocation of other resources (lighting, heating, cleaning).

Specific crowd counting

Most of the currently used models are trained to perform well in predicting the number of people in from places and from different points of view. Since UCLouvain only needs to count people in auditoriums and always from the same point of view in each of them, the main goal of this thesis is to try to improve the accuracy of the predictions by training models on the auditoriums.

In this master thesis, we will work on the training of a model that is performant on the auditoriums and which is easily accessible through a library. Moreover, to be able to use the model without any knowledge of computer sciences, we will create a web interface allowing users to download or take pictures and get the number of people present on it.

Chapter 1

State-of-the-art approaches

There are many solutions for the problem of counting people. The following section will briefly introduce them and their advantages or disadvantages.
Even though the objective of this work is to use the images from the security cameras, other opportunities for crowd counting have been developed over the years [17] :

- **Wi-Fi**
  Different studies developed some methods to count the number of people in a room based on the way these people alter the Wi-Fi signals. As shown in figure 1.1, the Wi-Fi access point and receiver are placed on either side of the room. The intuition is that regardless the background environment, the peaks of the Doppler spectrum increase as the crowd density increases [1]. The authors of the articles propose to analyse different characteristics of the signals using machine learning algorithms.
  On the one hand, the Received Signal Strength Indicator (RSSI) [18] can be analysed. It is an indicator of the signal strength that goes from 0 to 100. On the other hand, one can analyse the Channel State Information (CSI), that contains information about how the signal propagates from the transmitter to the receiver[19].

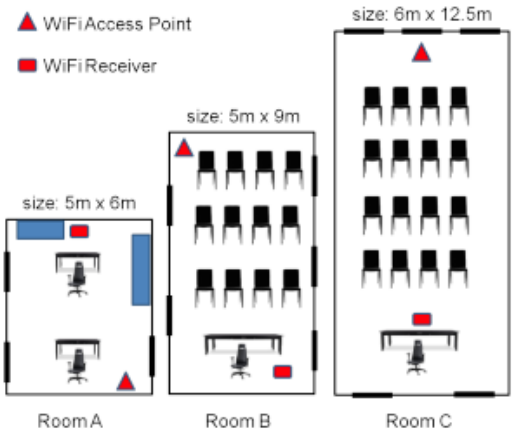


Figure 1.1: Wi-Fi devices positions [1]

This approach is efficient when it comes to small rooms with of a few people (less than 10), but it has never been generalized for bigger groups.

- CO^2 **concentration**
  In this method, the number of people is considered as a function of the CO^2 concentration in the room. Estimate the number then become a regression problem. Appendix A summarizes what a regression problem is and the differences with classification. This regression problem can be solved using different machine learning techniques such as SVM, deep learning or ELM. However, having good prediction with this technique is challenging, especially when the number of indoors occupant is large (more than a few tens) [20].

- **Environmental sensors**
  Several models have been developed using data from different sensors other than the CO^2 concentration, such as temperature, pressure, or humidity and combinations of these parameters. This method is similar to the previous one and is mainly used for a smaller number of people rather than in auditoriums.

The rest of this section will detail the different approaches for crowd counting based on images.

1.1 Detection-based approach

Detection based models have been used for a long time to detect people or objects on images. In order to count the number of people on an image, the goal of detection based models is to detect each person on the image using a trained classifier such as SVM's or Random Forests. There are several approaches to detect people in images [2] :

- **Monolithic detection**. In this approach, a model is trained to detect people based on features from their full body. The classifier is then applied in a sliding window through the image to detect the potential pedestrians. Since the same pedestrian can be detected several times, the less confident candidates are discarded using non-maximum suppression [21]. In fact, the goal is to keep only one candidate by pedestrian. This approach performs poorly on crowded scenes due to occlusion. The body of most people is not completely visible on the images so the classifiers can not detect people efficiently.

- **Part based detection** The principle of the part based detection is similar to monolithic detection but the classifier is only train to recognize one part of the body of people, often their head as in figure 1.2.b. The objective of this variation is to minimize the impact of occlusion. However, detecting just the head region is not sufficient for a reliable detection due to its shape and appearance variations. Including other parts of the body such as the shoulders tend to give better results [22].

- **Shape matching** In shape matching the classifier is trained to detect shapes on the images. These shapes correspond to parts such as the legs, arms, head, ... They can then be used to count the number of people, as shown in figure 1.2.c. Furthermore, since several parts of the body are detected there is also information about the position of each person in the scene.

- **Multi-sensor detection** If several cameras with different point of view of the same scene are available, their results can be combined to establish the number and possible locations of people, leading to better results.

(a) (b) (c)

Figure 1.2: Examples of image detection [2]

In general, detection based approaches do not lead to good performances on crowded scenes due to occlusion. Even if some tried to just use some particular parts of the body [23] or just the face [24], the result was still not really performing as the body is rarely visible, the faces often have really different angles with the camera and the brightness of the scenes often change during the day. All these factors make the training of reliable detection classifiers impossible.

1.2 Cluster based approach

This approach assumes that the way people move is unique. Based on the features of this movement, it is possible to group clusters together to represent moving entities, as shown in figure 1.3. The main advantage of this approach is that it is unsupervised learning. It means

(a) (b)

Figure 1.3: Examples of cluster-based detection [2]

that training images do not need to be labelled as for the other approaches. The disadvantage of this approach is that it requires several consecutive images of the same scene with people moving. It is thus not relevant for crowd counting in auditoriums.

1.3 Regression-based approach

In the early stages of crowd counting, this approach was the most used and the one that led to the best results. The idea of counting by regression is to avoid the detection and the tracking of individuals as in the previous approaches, which was not really efficient. It is rather to considerate the crowd as a whole and estimate the number of people based on crowd patterns.

As shown on figure 1.4, counting by regression is divided in three steps :

- **Feature extraction** This step concerns the selection of low level visual properties of an image that will be given as input to the regression model. There are 3 kinds of features, illustrated on figure 1.4. It is interesting to combine them :
  - The foreground segmentation features : They capture the global properties of a segment such as the area or the perimeter (in pixels).
  - The edge features : These features will capture additional information inside the segment. Moreover, crowded scenes will have more complex edges than non-crowded ones.
  - Texture and gradient features : Analysing the texture and the gradient can help a lot for estimating the number of people. Stronger textures mean more crowded scenes. It also exists texture descriptors such as Gray-level co-occurrence matrix (GLCM) or local binary pattern (LBP) [25] that are good examples of texture and gradient features.

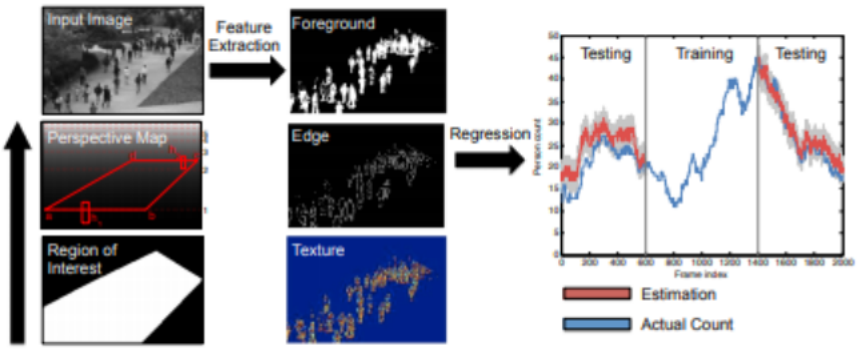


Figure 1.4: Typical pipeline for counting by regression [2]

- **Geometric correction** One of the recurrent problems of counting by regression is the geometric distortion : far objects appear smaller than close objects. This problem is detailed in section 4.1.2. Several methods have been used to compensate the errors due to geometric distortion while counting by regression. The first one is to divide the image in smaller cells, each of which with a different regression function. Other methods of perspective normalization are described in article [2].
- **Regression** A regression model is trained to use the features extracted in the first step of the process as input and give the number of people on the image as output. A lot

of different regression models with different configurations were tested over the years, such as Linear Regression, Kernel Ridge Regression, Support Vector Regression, Gaussian Process Regression, Bayesian Poisson regression [26], etc. More information about each of these models can be found in [2].

To enhance the results of the regression models, especially on extremely dense crowded images, some researchers performed regression on features from different points of view of the same scene [27].
The main problem of regression is that it is really dependent on the point of view on which it has been trained. Models will perform poorly on images from a new point of view.

1.4 CNN-based approach

Recently, most of the performing algorithms for crowd counting use Convolutional Neural Networks (CNN) because of its success in image classification and recognition. The website paperswhitecode[1] lists and compares some algorithms for crowd counting on several datasets, and most of the best results are held by CNN with different configurations. CNN are Deep Neural Networks (DNN), in the following section you can find a reminder of how DNN and especially CNNs work and why they have such good results. Among the good performing CNN for crowd counting is the CSRNet model created in 2018 [3] and used in this thesis. This model stayed simple and a lot of algorithms for crowd counting were based on it during the following years [28]. Its architecture is described with more details in section 3. The principle of the CNN for crowd counting is to separate the process in two steps. The first step is to create a density map of the scene, as shown on figure 1.5.

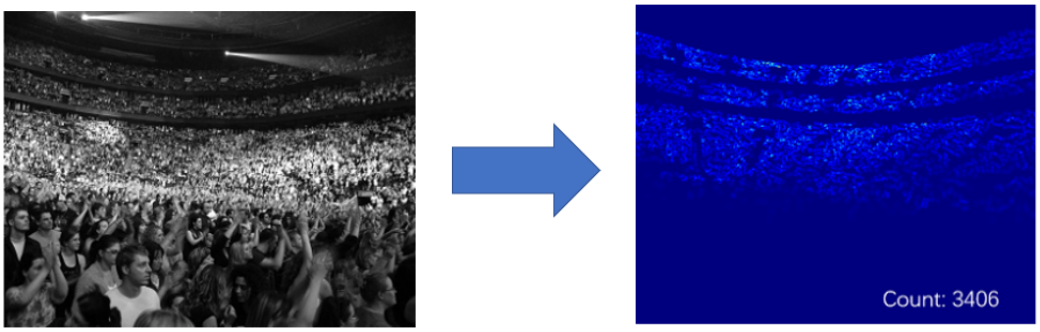


Figure 1.5: Density map exemple [3]

Each pixel of the density map correspond to a pixel of the image and has a value between 0 and 1 (it gets bigger as the density of people augment in this area). The density map is created by the detection of the people's heads, more information is available in the next sections. This first step also permits to have more information than a simple counting, such as the distribution of the crowd.

The second step is to integrate the density map into an actual number of people. In this case, it is equivalent to do the sum of all the values of the pixels of the density map.

[1] https://paperswithcode.com/task/crowd-counting

A big advantage of this method is that it respects the privacy of people on the pictures because there is no recognition of head but recognition of some combination of pixels.

Chapter 2

Convolutional neural networks

Before going in further details about the architecture of CSRNet, this section present how deep neural networks and more precisely convolutional neural networks work.

2.1 Neural networks

2.1.1 Biological inspiration

Neural networks are inspired from the human brain, in particular the way human neurons communicate with each other. Figure 2.1 shows a basic neuron. It gets information from the other neurons via its dendrites, processes the information in the cell body (also called soma) and then transmits it to other neurons by the axon [4].

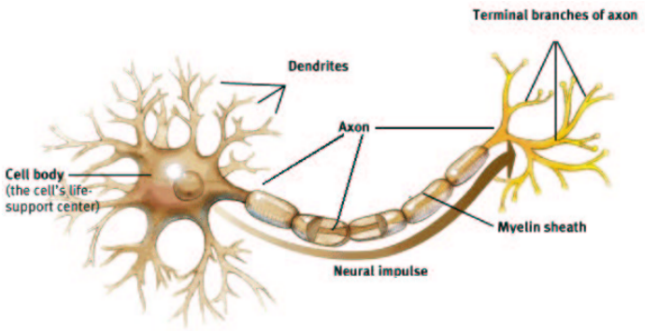


Figure 2.1: Structure of a neuron [4]

Neural networks are composed of layers of communicating artificial neurons called nodes. The structure of a basic node is represented in figure 2.2. The inputs are similar to the information arriving from the dendrites. Each input is multiplied by a weight, it helps to determine the importance of a given input, if the input is not important its weight will be close to 0. These inputs are then summed in the transfer function. At this point of the neuron, the net input has the form : $\sum_{i=1}^{n} x_i w_i + bias$.

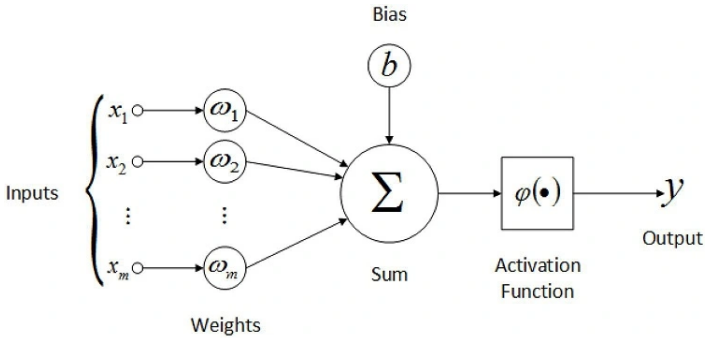


Figure 2.2: Structure of a Neural network node [5]

The weighted sum is then given to an **activation function** and produces an output. If we want to draw a parallel with the neuron, we can see the output as an axon that can be connected to the output of the network or to others neurons.

Activation functions

If the weighted sum does not pass in an activation function, the output of the network would be a linear function of degree one. This kind of equation is easy to solve but its complexity is limited and it lacks the ability to learn and recognize complex mappings from data [29]. On the other hand, activation functions are non-linear functions applied in each node to make the network non-linear. That way, the network can learn and compute any function to approximate the best results. In order to be able to train the networks, the activation functions have to be differentiable (see section 2.1.2).

The most widely used activation function for hidden layers is the Rectified Linear Unit (ReLU), its graph is available in figure 2.3.

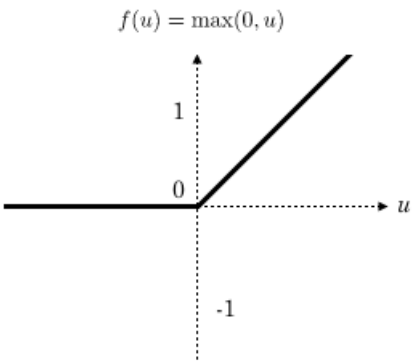


Figure 2.3: Graph of the ReLU function [6]

This function was proposed in 2000 as a replacement for classical activation functions that were sigmoid and tanh. It has a lot of advantages compared to these :

- Efficient to compute because all the neurons are not activated at the same time.
- Better gradient propagation that helps to solve the vanishing gradient problem. This problem prevents the correct training of all artificial neurons in deep neural networks.

- Many variants of ReLU exist, that can be used depending on the problem the network has to solve.

Layer organization

The nodes are organized in layers, as shown in figure 2.4. The output of one layer of nodes is the input of the next layer. In deep neural networks, there are several hidden layers of nodes. In contrast, networks with only one hidden layer are called shallow networks. Shallow networks have the universal approximation property that states that one hidden layer is enough to approximate any function if it is wide enough. They do not suffer from the vanishing gradient problem but they have never convinced since their results are often less good than the deep learning results [7].

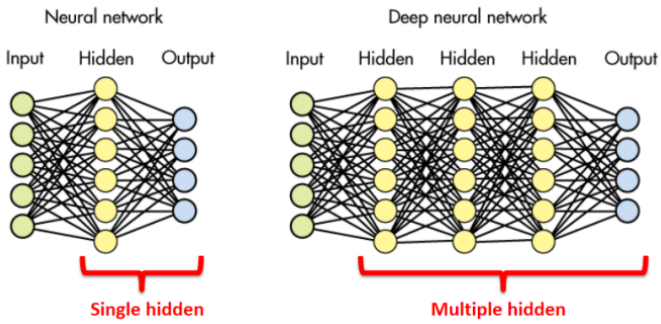


Figure 2.4: Representation of a deep neural network [7]

2.1.2 Model training

The main advantage of neural networks is their ability to learn how to accomplish the task they are designed for. During the learning, inputs are furnished to the network with their correct output. What is called "learning" is actually adjusting the weights of each neuron, so that the network will improve its performance.

Loss function

In order to improve the performance, we need a performance measure. This measure is called the loss function. The result of the loss function represents the error in the network's predictions over the training examples. The goal during the training is therefore to minimize it. The most used loss function is the MSE (mean squared error) defined as :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

where $\hat{y}_i$ is the predicted outcome of training example i, y_i is the real outcome and n the total number of training examples. Since every prediction $\hat{y}$ is depending on all the weights of the network, the loss function is depending on lots of parameters and is therefore hard to minimize.

Stochastic gradient descent

The process used to minimize a multi-variable function is called gradient descent. It is based on the principle that, if a multi-variable function $F(X)$ is defined and differentiable in the neighbourhood of a point a, then $F(X)$ decreases the fastest in the direction of the negative gradient $-\Delta F(a)$.
Knowing that, the principle of the gradient descent is to start at a point in the function and then to go step by step in the direction of the negative gradient, to a minimum, as shown in figure 2.5 for a function of two parameters.

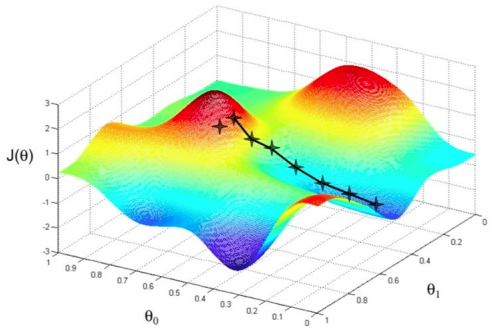


Figure 2.5: Illustration of gradient descent in 3D [8]

In the case of machine learning, the principle stays the same, but the function to minimize is the loss function and its parameters are all the weights of the network. After each training example, the weights are updated in the direction of the inverse of the gradient, as in the following equation :

$$w := w - \eta \Delta L(w)$$

With w being the weights, L the loss function and η the learning rate, discussed in the next section.

Learning rate

Another important parameter is the size of the steps towards the minimum called the learning rate. The learning rate must be chosen to be a good balance between efficiency and precision. As illustrated in figure 2.6, if the learning rate is too small, the algorithms takes a lot of time to find the minimum but if it is too large it might never converge. A lot of algorithms now use an adaptive learning rate that starts larger and decreases as the minimum of the function approaches.

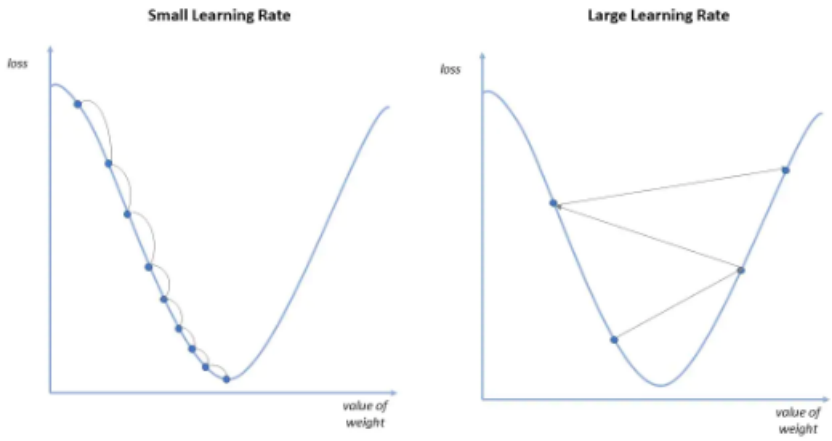


Figure 2.6: Learning rate [9]

2.2 Image representation

In machine learning, the models need to learn from data and in this thesis, the models will learn from images. The model need a way to analyse images. In our case, the input format is an RGB image. RGB stands for Red Green Blue. RGB images are represented by a tensor of shape Width x Height x 3 as shown on figure 2.7. Since all the spectrum of colours can be formed from these three colours, each matrix of shape Width x Height represent the intensity (on a scale from 0 to 255) of the colour needed for each pixel in the image.

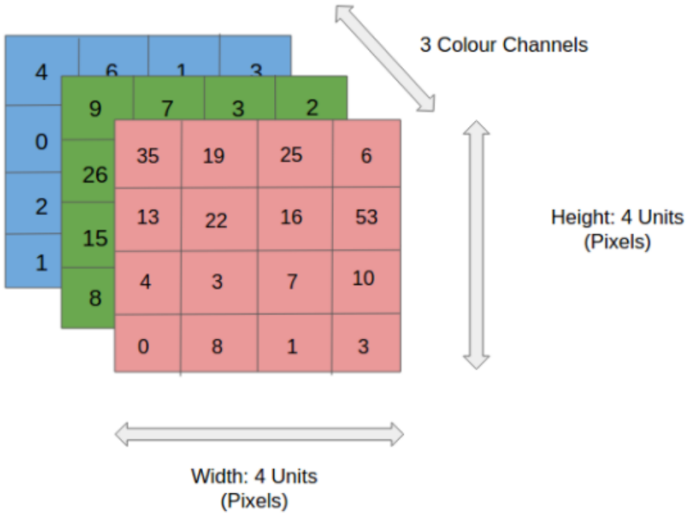


Figure 2.7: RGB representation of an image [10]

2.3 Convolutional Neural Networks

Convolutional neural networks are the best performing deep networks when it comes to computer vision tasks such as analysing images. They are composed of three main types of layers : convolutional layers, pooling layers and fully connected layers.

2.3.1 Convolutional layers

Convolutional layers perform a convolution on the inputs. They act as pattern detectors and are the key layers of the convolutional neural networks.

Convolution

Convolution in image processing is related to the mathematical convolution, but under a matrix form. The convolution is realized between the input matrix and the filter (also called convolution matrix or mask). The complete formula is shown in figure 2.8.

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} * \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{(m-i)(n-j)} y_{(1+i)(1+j)}$$

Input matrix — Convolution matrix

Figure 2.8: Matrix convolution

In convolutional layers, the **filter** is applied to an area of the input image and the result is fed to an output array, as shown in figure 2.9.

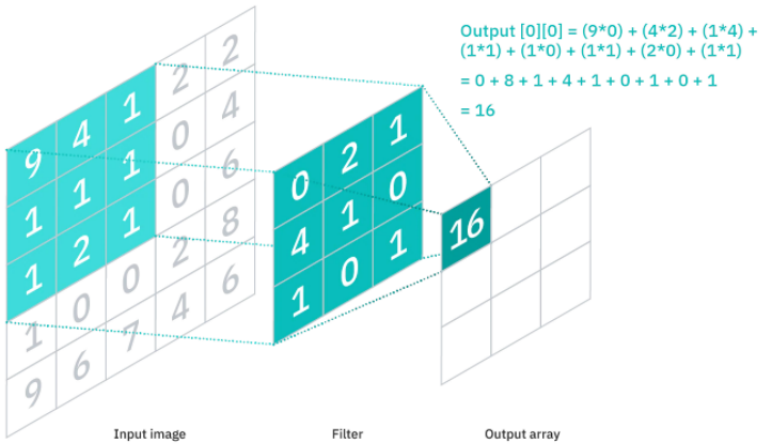


Figure 2.9: Convolutional layer [11]

After that, the filter moves by a fixed number of pixels, this number is called **stride**. The stride is often 1 or 2 but rarely superior. An issue of this method is that we will lose pixels on the

perimeter after each convolution. A solution to solve this problem is to add a **padding**, the padding corresponds to extra pixels with values of 0 that are added next to the boundaries of the input. As visible in figure 2.10, thanks to the padding, the output size is the same as the input size.

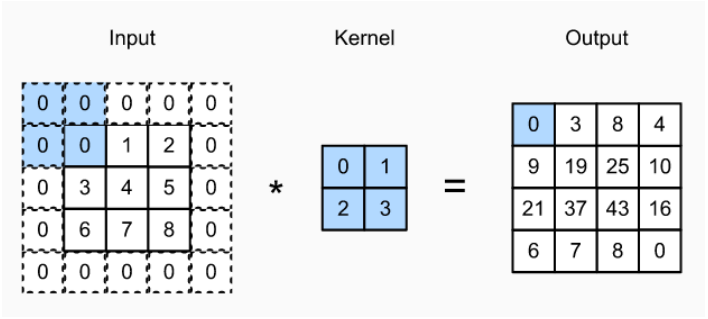


Figure 2.10: Padding representation [12]

Pattern detector

Convolutional layers act as a pattern detector, the more the input matrix and the convolution matrix are similar, the more the convolution will be high. This is shown in figure 2.11, the filter and the first image convolve to lead to the second image. The filter has increasing values from left to right. Hence, when it convolves with an edge in the picture the convolution product is high because edges are also composed of really different values on the left and right.

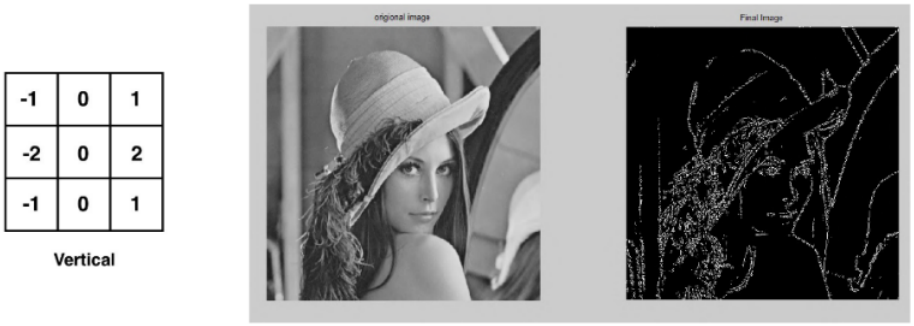


Figure 2.11: Convolution effect on an image [7]

Most CNN have several consecutive convolutional layers. This allows next layers to see the input through the receptive field of the previous layers. Figure 2.12 shows how consecutive layers would detect a complicated pattern such as a bicycle. The bicycle can be seen as a sum of simpler patterns such as the wheels, the pedals, etc. The first layer will detect these simple patterns and next layers will combine them in more complicated patterns, creating a hierarchy in the CNN.

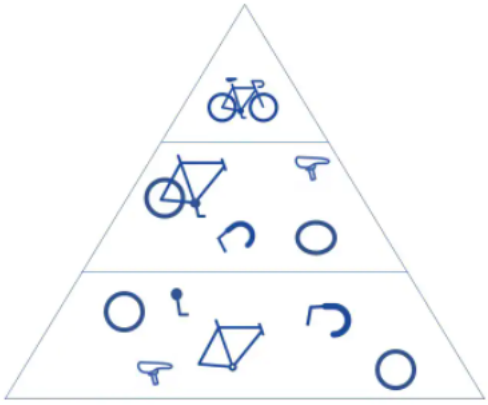


Figure 2.12: Pattern detection example [11]

2.3.2 Pooling layers

Pooling layers are known as downsampling layers. Their goal is to reduce the size of the input. It reduces the number of parameters and by the same occasion the number of computations. The pooling layers summarize the features present in a region of the feature map generated by convolutional layers. The principle of the pooling, illustrated in figure 2.13, is to apply a filter (often of size 2x2) to a window of the input and reduce it to one value. There are two types of pooling. In average pooling, the filter calculates the average value within the receptive field. In a max pooling, only the maximum value of the filter is sent to the output map. Max pooling is mostly used since it often performs better because it is noise suppressant.

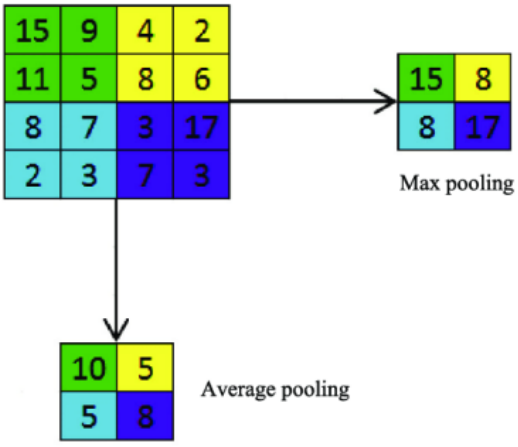


Figure 2.13: 2x2 pooling with stride 2 [7]

2.3.3 Fully connected layers

Fully connected layers are the layers used for classification. Unlike convolutional layers, each node of a fully connected layer takes the input from all nodes in the previous layer, as shown in figure 2.14. Fully connected layers usually use a softmax activation function to assign a probability between 0 and 1 to each possible output [11]. Some CNN such as CSRNet do not have fully connected layers since they are not used for classification.

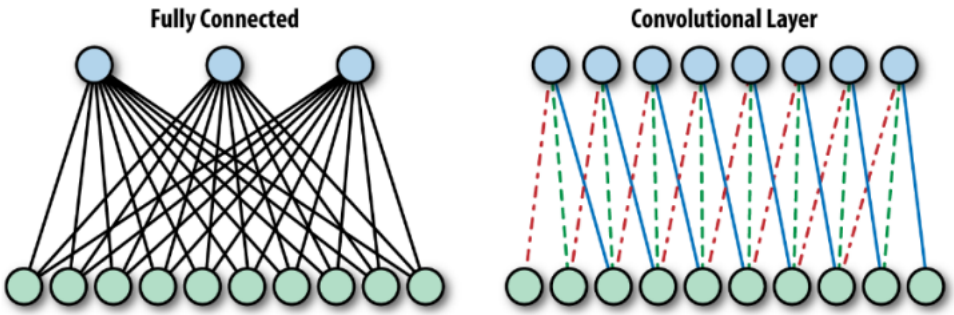


Figure 2.14: Fully connected layers [7]

2.3.4 Full network

Figure 2.15 put together all types of layers discussed in the previous sections to form a simple network called LeNet. It was created in 1998 by Yann Lecun to detect numbers from 0 to 9 on 32x32 images and it was the first occurrence of machine learning in computer vision.
The network is formed by alternating convolutional and pooling layers and ends with 3 fully connected layers to put each image in one of the 10 classes (from 0 to 9).

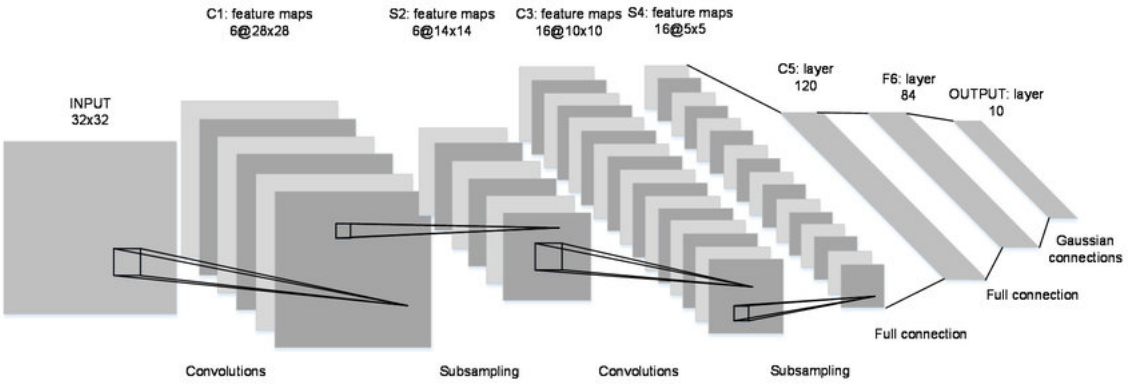


Figure 2.15: Full architecture LeNet [7]

2.4 Transfer learning

Transfer learning is the fact to reuse knowledge gained while solving a problem and apply it to another different but related problem. It is really common [30] in image data processing to use a pre-trained model for image classification task such as VGG or ImageNet as a base for new models that perform more specific tasks. In fact, these models are robust since they have been trained on a large number of various images. The last layers of these models are then replaced by new layers that are more convenient for the specific task and only these new layers are trained on the new dataset, as shown in figure 2.16.

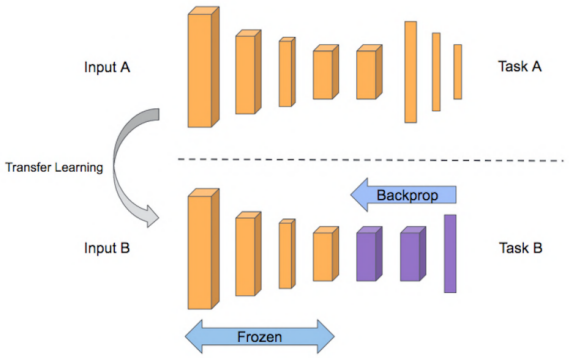


Figure 2.16: Transfer learning representation [7]

Transfer learning allows to significantly reduce the training time and the number of data needed to train new models.

Chapter 3

Network architecture

The architecture that we used for the crowd counting is based on the CSRNet architecture [3]. It uses a CNN-based approach that will capture high-level features and generate high-quality density maps. The main advantage is the small network complexity : the results are good even though the training time is not too long.

3.1 Frontend

The front-end of CSRNet is based on a well-known architecture called VGG-16 [13].

3.1.1 Known architecture : VGG

Initially, this network was called ConvNet and VGG was the name of the team at its origin but people later called it VGG. The VGG networks are popular and efficient convolutional networks for large-scale image classification. The number of weight layers of the VGG networks goes from 11 to 19 but in the one used in CSRNet, there are 16 layers as its name says so.

The exact architecture of all VGG networks is shown in figure 3.1 and the VGG-16 implementation used by CSRNet is highlighted. In these implementations almost all convolutional layers have a receptive field of 3x3. It is the smallest size that allow to use the notions of left, right, centre. Moreover, stacking several layers with a small receptive field (without any pooling between them) has two advantages compared to using one filter with a larger receptive field [13] :

- It reduces the number of trainable parameters. Indeed, using one 7x7 convolutional layer instead of three 3x3 convolutional layers increases the number of parameters by 81%. If we assume that the outputs have C channels, we have :

  $\rightarrow$ $3(3^2C^2) = 27C^2$ weights for the layers with a receptive field of 3x3.

  $\rightarrow$ $7^2C^2 = 49C^2$ weights for the layer with bigger receptive field.

- It incorporates three non-linear rectification layers instead of one. The advantage is that the decision function will be more discriminant (section 2.1).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 3.1: Architectures of the ConvNets [13]

About the width of the convolutional layers, it starts at 64 and increase of a factor 2 after each pooling layer. Also, the pooling layers are only max-pooling layers carried over a 2x2 pooling window with stride 2 [13].

The main advantages of using the VGG-16 implementation is the transfer learning ability that is very strong and the flexible architecture that can generate the density map by concatenating the back-end.

3.1.2 CSRNet frontend

As said before, the initial purpose of VGG-16 is object classification but as for some other works [31] [32], the architecture was modified to suits the actual problem.

The frontend of CSRNet is an adaptation of the VGG16 network. Indeed, the three fully connected layers, which are useful for classification are removed. There is a suitable architecture but it is still important to look for the best trade-off between accuracy and resource overhead such a training time, memory consumption or the number of parameters. It is important because a very fast model can be very poor in precision but a very precise model that is too costly is not useful either. Experiments showed that only keeping the 10 first layers of VGG-16 with only 3 max-pooling layers was the best trade-off between accuracy and resource overhead [3]. The final architecture is shown in figure 3.2.

input(unfixed-resolution color image)
front-end (fine-tuned from VGG-16)
conv3-64-1 conv3-64-1
max-pooling
conv3-128-1 conv3-128-1
max-pooling
conv3-256-1 conv3-256-1 conv3-256-1
max-pooling
conv3-512-1 conv3-512-1 conv3-512-1

Figure 3.2: Architecture of the frontend [3]

3.2 Backend

Now that we have the frontend of the architecture, let's focus on the backend. The idea is to generate high-quality density maps but the problem is that we can not keep the same structure (convolutional and pooling layers) as for the frontend because this will further reduce the output size. Generating high-quality density maps with a small output size is difficult, it is why we need another solution.

Even if pooling layers are good at maintaining invariance and regulating overfitting there is a price to pay. These layers reduce the spatial resolution which means that some information about the feature map is lost. We then will discuss two possible solutions. The first one are the dilated convolutional layers and the second one consist of adding a layer that will upsample its input so that we can have more information.

3.2.1 Dilated convolutional layers

The idea about these layers is to replace the combination of max-pooling, convolutional and upsampling layers. Indeed, when applying convolutional and pooling layers, the size of the output is reduced but using dilated convolutional layers permits to enlarge the receptive field. The max pooling layers are therefore not necessary [3]. So, it is interesting to think about them.
The good thing about dilated convolutional layers is that they have led to the improvement of accuracy in segmentation task. We can define a 2-D dilated convolutional layer as follows :

$$y(m,n) = \sum_{i=1}^{M} \sum_{j=1}^{M} x(m + r \times i, n + r \times j) w(i,j)$$

where x(m, n) is the input of the layer and y(m, n) is the output of the layer. The filter is represented by w(i, j), the length by M and the width by N. The parameter r correspond to the dilation rate and the figure 3.3 clearly explain its utility.

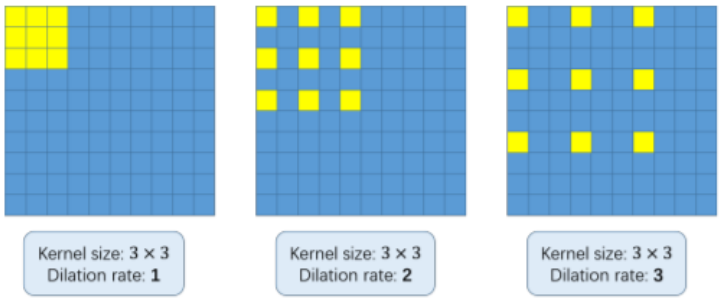


Figure 3.3: Dilated convolutional kernels [3]

3.2.2 Upsampling the data

In these methods, we will not substitute the max-pooling and convolutional layers but we will add a new layer after them to retrieve information.

Unpooling layers

Using pooling layers is a good idea when talking about classification because it will only keep the most robust activations but it will lose spatial information that are used for semantic segmentation. To solve this problem, we can use unpooling layers that will do the reverse operation as represented in figure 3.4.

The method for doing this is to record the positions of the maximum activations retained during pooling in switch variables. We can now replace the activations from the input at their initial positions thanks to the switch variables in order to reconstruct an approximate version [14] [33]

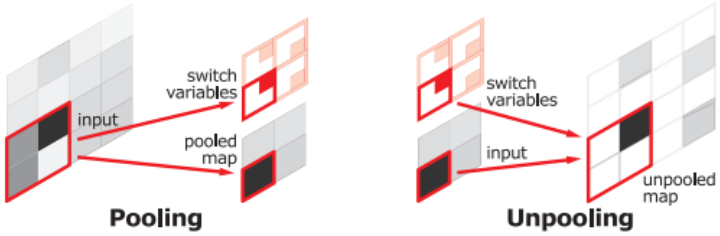


Figure 3.4: Unpooling illustration [14]

Deconvolutional layers

The unpooling layers do indeed provide an enlarged activation map but it is a sparse one. By using deconvolutional layers, we can minimize the loss of dilated and unpooling layers but it also adds complexity and latency that are not desired. Indeed, the deconvolutional layers assign several outputs to a single input as shown in figure 3.5. The final output is therefore a large and dense activation map. [14]

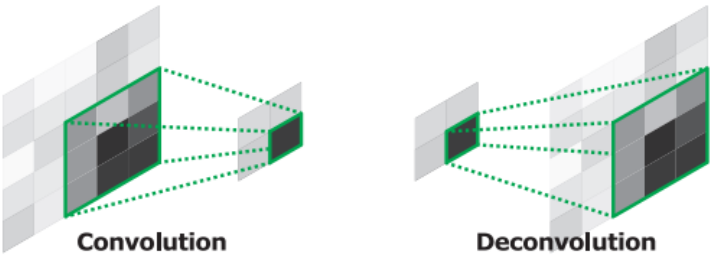


Figure 3.5: Deconvolution illustration [14]

Bilinear interpolation

When upscaling the size in a network, it is frequent to use a bilinear interpolation. This method will calculate the value of a point depending on its closest diagonal points. Let's start with the square in figure 3.6 where we know the values and the positions of each corner. The formula to compute the value I of a point in the square is the following [34] :

$$I(x,y) = (1-a)(1-b)I(x',y')+(1-a)(b)I(x',y'+1)+(a)(1-b)I(x'+1,y')+(a)(b)I(x'+1,y'+1)$$

Which is the sum of the values of the four corners as a function of the distance from the current point.

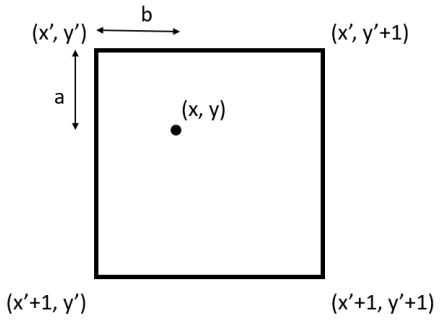


Figure 3.6: Bilinear interpolation : example

3.2.3 Choice

On the one hand, we will talk about the upsampling methods. We can say that the deconvolutional layers are more interesting because they retain more spatial information.
On the other hand, a comparison has been made to compare the dilated convolutional layers and the option with the deconvolutional ones. We can see in figure 3.7 that the dilated convolutional layers provide more information and a better resolution. This method do not increase the parameters or the complexity but will enlarge the receptive field and is thus used in the CSRNet backend.

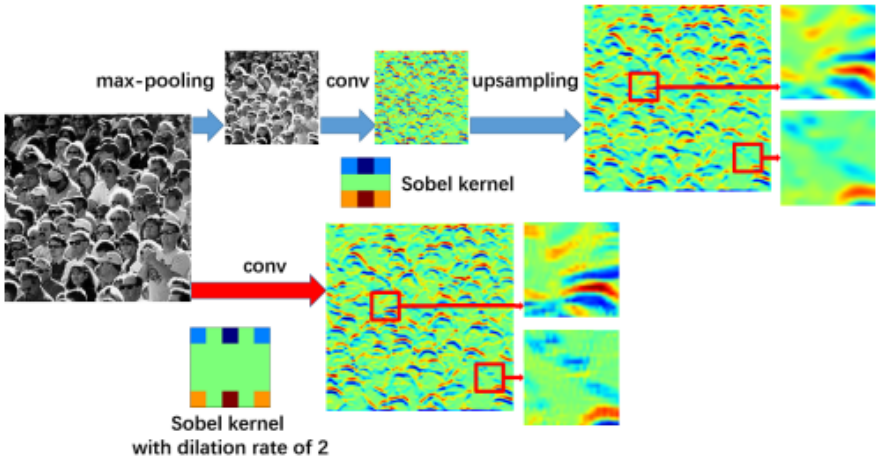


Figure 3.7: Methods comparison [3]

3.3 Network configurations

There are four different configurations of CSRNet where each configuration corresponds to a different dilation rate in the backend while the frontend stays the same. These configurations are available in figure 3.8

Configurations of CSRNet			
A	B	C	D
input(unfixed-resolution color image)			
front-end (fine-tuned from VGG-16)			
conv3-64-1 conv3-64-1			
max-pooling			
conv3-128-1 conv3-128-1			
max-pooling			
conv3-256-1 conv3-256-1 conv3-256-1			
max-pooling			
conv3-512-1 conv3-512-1 conv3-512-1			
back-end (four different configurations)			
conv3-512-1 conv3-512-1 conv3-512-1 conv3-256-1 conv3-128-1 conv3-64-1	conv3-512-2 conv3-512-2 conv3-512-2 conv3-256-2 conv3-128-2 conv3-64-2	conv3-512-2 conv3-512-2 conv3-512-2 conv3-256-4 conv3-128-4 conv3-64-4	conv3-512-4 conv3-512-4 conv3-512-4 conv3-256-4 conv3-128-4 conv3-64-4
conv1-1-1			

Figure 3.8: Configurations of CSRNet [3]

Since there are three max pooling layers, the size of the density maps are 1/8 of the size of the input. For those who would like to have an output of the same size as the input, it is proposed to use a bilinear interpolation (with a factor of 8). The best configuration is the B and it is the one that we used [3].

Chapter 4

Training of the model

In the following section, we detail the ground truth generation with a concrete example, the loss function and some details about the training.

4.1 Ground truth

We need images of the auditoriums to train the models but we also need their ground truth. In our case, the ground truth correspond to the density map of the image. The quality of these density maps has to be as good as possible given that the quality of the training will impact the quality of the predictions. The idea is to convert annotated images to a map usable by the model.
These maps are generated based on a .mat file with all the coordinates of the heads [35]. The method for generating these maps is the one described in the article [36]. The generation can be done in Python by following the related Notebook in the GitHub repository B.3.3.

4.1.1 Generate the density function

The first step of this method is to generate a crowd density map. For doing so, we need an approach to represent the pixels corresponding to an annotated head. Actually, each of these pixels, denoted x_i, will be represented by a delta function. Following this method for an image containing N heads gives the following function :

$$H(x) = \sum_{i=1}^{N} \delta(x - x_i) \tag{4.1}$$

The second step is to transform this function into a continuous one by using a Gaussian kernel G_σ [37]. The new density function is therefore :

$$F(x) = H(x) \times G_\sigma(x) \tag{4.2}$$

There is an important comment to tell about this function. Indeed, it assumes that all the x_i are independent which is not the case due to the geometric distortion.

4.1.2 Geometric distortion

The perspective of the 3D image results in different sizes for heads located at different places. As visible in figure 4.1, we can see that the head at the bottom has a size of 100x120 pixels while the one at the top left has a size of 25x24 pixels. Now that we know this, we need to take it into account when estimating the crowd density.

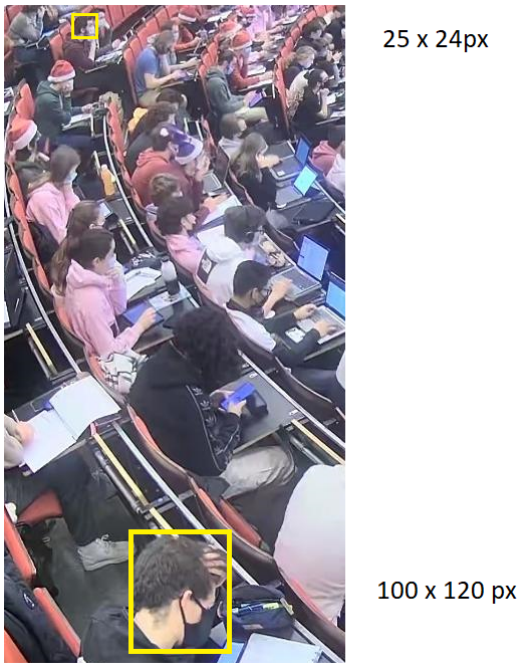


Figure 4.1: Example of geometric distortion

The problem in this case is that we have no clue about the distribution of students in the auditoriums and it is hard to precisely determine the size of each head because we sometimes see only a part of the head. However, taking the average distance between a head and the k-neighbouring ones can give a quite good estimation about this geometric distortion because neighbours of someone standing in the background will be closer than for someone in the foreground.

Now that we have a method to estimate the spread parameter σ to apply to each head, we have :

$$F(x) = \sum_{i=1}^{N} \delta(x - x_i) \times G_{\sigma_i}(x), \qquad \text{where} \quad \sigma_i = \beta \bar{d}^i \tag{4.3}$$

σ computation :

As we can see, σ is composed of 2 terms and is different for each head. To construct $\bar{d}^i$, we need the distances from a head i to its k nearest neighbours that we can write as $\{d_1^i, d_2^i, ..., d_m^i\}$ with an average distance corresponding to $\bar{d}^i = \frac{1}{m} \sum_{j=1}^{m} d_j^i$.

The estimate density around pixel x_i is thus the delta function convolved with a Gaussian kernel having variance σ_i which depends on $\bar{d}^i$.
It has been found empirically that the best choice are $\beta = 0.3$ and k = 3 [36]

It is interesting to note that for non-crowded images an alternative method can be used. It is possible to use a fixed value for σ and it depends on the case and will correspond to the average head size. We do not use this method because at the UCLouvain, students in a crowded auditorium are not spaced.

4.1.3 Example

Words are good but a concrete example is always better. In this section, we will start from an image and see all the steps leading to the ground truth.

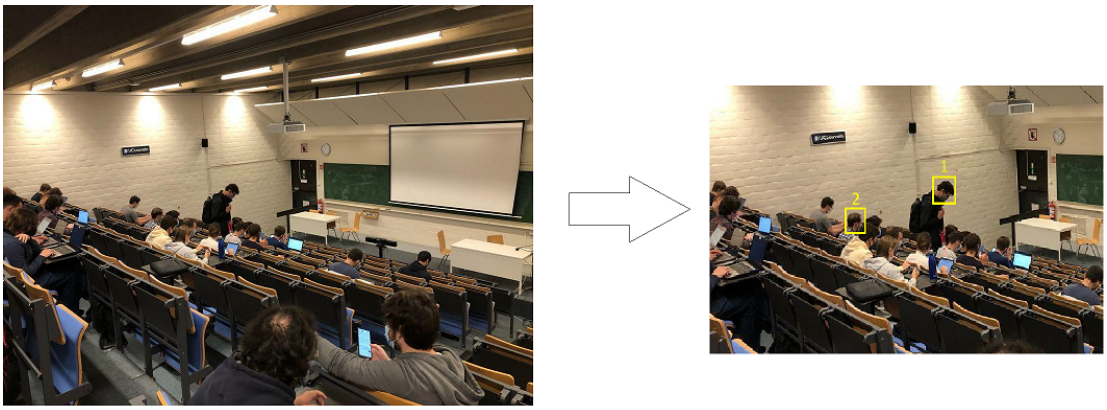


Figure 4.2: Example image

The initial image is visible in figure 4.2 and we will, for the example, focus on two heads that are framed.

Step 1

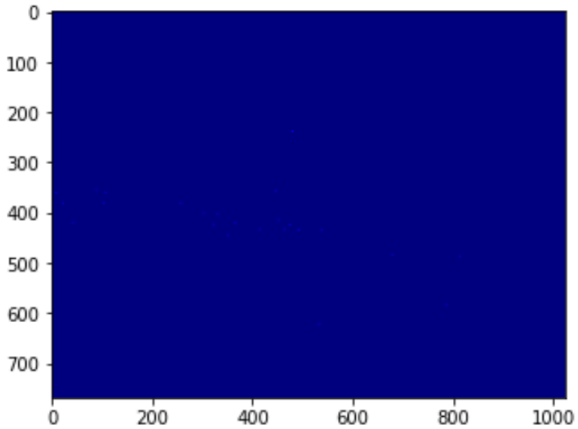
We use formula 4.1 and we retrieve information about the locations of the heads. If we take a look at the values of the maps around each head, we can see in figure 4.3a that the value of the coordinates representing a head has a non-null value.

In figure 4.3b where the generated map is displayed we can not see any information because among all the values, only 23 positions have a value (which is one). Now, we are beginning to understand why the Gaussian kernel is useful.

```
For head 1
[0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0.]

For head 2
[0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0.]
```

(a) Values around heads



(b) Ground truth

Figure 4.3: Step 1 of the density map generation

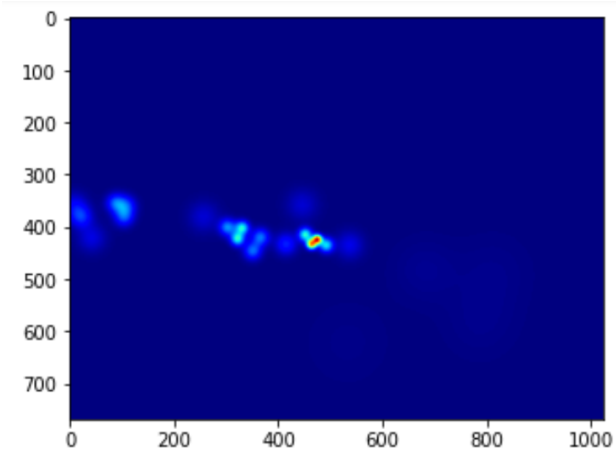
Step 2

We found that formula 4.1 is not sufficient so we will see what is the impact of the Gaussian kernel by using formula 4.3. If we look again at the values around the heads in figure 4.4a, there are some changes.
On the one hand, for head "1", which correspond to the guy who is standing up, we can see that the value at the coordinates of the head has the higher value and that the others decreases according to the same amount. Indeed, this head is not close to the other ones so the values only represent the impact of head 1.
On the other hand, we have a look at the head "2", we observe that the values on the right are higher than values on the left. It can be explained by looking at the picture because there are 2 heads close to this one on the right so the values for each of the heads are overlapping.

In the end, the generated density map has visible points corresponding to the values seen before as shown in figure 4.4b

```
                    For head 1
[0.00036966 0.00037097 0.0003714  0.00037097 0.00036966]
[0.00037097 0.00037227 0.00037271 0.00037227 0.00037097]
[0.0003714  0.00037271 0.00037315 0.00037271 0.0003714 ]
[0.00037097 0.00037227 0.00037271 0.00037227 0.00037097]
[0.00036966 0.00037097 0.0003714  0.00037097 0.00036966]
                    For head 2
[0.00137269 0.00139183 0.00140097 0.00140034 0.00139059]
[0.00139048 0.0014104  0.00142026 0.00142033 0.00141127]
[0.00139685 0.00141744 0.00142804 0.00142892 0.00142077]
[0.00139169 0.00141288 0.00142424 0.00142608 0.00141909]
[0.00137524 0.00139696 0.00140913 0.00141209 0.00140653]
```

(a) Values around heads



(b) Ground truth

Figure 4.4: Step 2 of the density map generation

4.2 Loss function

The loss function is the Euclidean distance and it will compute the differences between the ground truth and the generated density map as follows [3] :

$$L(\Theta) = \frac{1}{2N} \sum_{i=1}^{N} ||Z(X_i; \Theta) - Z_i^{GT}||_2^2 \tag{4.4}$$

where N is the batch size : the number of images in the training set. $Z(X_i; \Theta)$ corresponds to the CSRNet output with Θ representing the parameters of the model. Finally, we have Z_i^{GT} : the ground truth of image X_i.

4.3 Training details

The network is trained as an end to end structure. While starting the training from scratch, the weights of the frontend are initialized from a well-trained VGG 16. For the other layers, the creators of CSRNet initialized them from a Gaussian with a 0.01 standard deviation. However, in our experience this led to a really huge loss at the beginning that prevented from a correct training of the network. We therefore decided to initialize them at 0 and were able to achieve good results. The network is trained using Stochastic gradient descent, with a fixed learning rate of 1e-6. It is important to know when to stop the training of the model : it is not always better to train a model again and again because overfitting can happen.

Overfitting

Overfitting occurs when a model fits its training data exactly. In fact, since the minimized loss function is only dependent of the training data, the weights can become too dependent of the training data and therefore have bad results on new data.

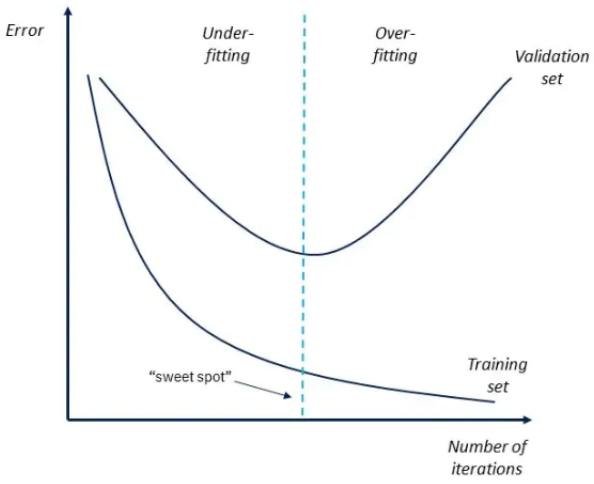


Figure 4.5: Illustration of overfitting [15]

As shown in figure 4.5, the training error is then still decreasing while the validation error is increasing. It is more likely to happen when the model train for a large number of epochs or if there is not enough training data.

To avoid overfitting, several techniques are used such as early stopping and data augmentation [15]. In the case of CSRNet the dataset is divided in two parts : the training set and the validation set. The models only trains on the training set and after each epoch the MAE of the validation set is calculated. Only the model with the best MAE is saved as the good model. This permits to detect overfitting if the MAE on the validation set start increasing for several consecutive epochs.

Training environment

It should also be noted that all the training were done using *Colaboratory* also known as *Google Colab*. It is an online resource that allows to run python codes and that gives free access to GPU (2-core Xeon 2.2GHz, 13GB RAM).

Chapter 5

Data collection

Now that we have a model to count the people on an image, we need a dataset of images to train the model. There are general data which are used in many crowd counting applications and specific data. In our case, the specific data concerns the auditoriums.

5.1 General data

The idea of crowd counting is not new and there are already a lot of experiments which also needed data to create their models. The two main datasets for crowd counting are the ShangaiTech [1] and the UFC datasets [2]. These two datasets have the same structure : each crowd image has a corresponding .mat file with the coordinates of the head of each person in the image. The ShangaiTech dataset consist of 1198 annotated images divided in two parts. Part A consists of 482 images with a high crowd density while the 782 images of Part B contain fewer people, as shown in figure 5.1.

(a) Image from Part A

(b) Image from Part B

Figure 5.1: Samples from the ShangaiTech dataset

We first trained the model on the Shanghai dataset in order to reproduce the results of the CSRNet creators [3]. It was also a way to check that the modifications we did in the code to adapt the model from python 2 to python 3 were correct.

[1] https://github.com/desenzhou/ShanghaiTechDataset
[2] https://www.crcv.ucf.edu/data/ucf-cc-50/

This basic model can be used as a base model to compare the results of our different models and see if training the models specifically for auditoriums or for the A10 (One of the largest auditorium at the UCLouvain) really improve the efficiency of the counting.

5.2 Auditorium data

In deep learning, the models learn from their datasets. The cameras in the auditorium are often at the back, which implies that people are mainly from the back or the side. Moreover, due to the Covid-19 pandemic, students wear masks during the courses which is not the case of most of the available datasets. It is thus interesting to train the model specifically on these auditoriums. Moreover, training the models with images from similar auditoriums always taken in the same place could lead to even better results. Our first way to collect images was to go by ourselves and take pictures with our phone during courses but due to time constraints, it was hard to make the size of the dataset grow fast enough.

5.2.1 Access to the camera

Since training a neural network requires a lot of data, we asked to have access to the images of the security camera of the A10 auditorium, one of the biggest auditoriums at UCLouvain. In a concern of security, we used the set-up described in figure 5.2. We had to first connect to a PC from the UCL using the remote desktop functionality. From there, we had access (with remote desktop again) to a VM that was not connected to the internet and only had access to the camera from the A10 auditorium.

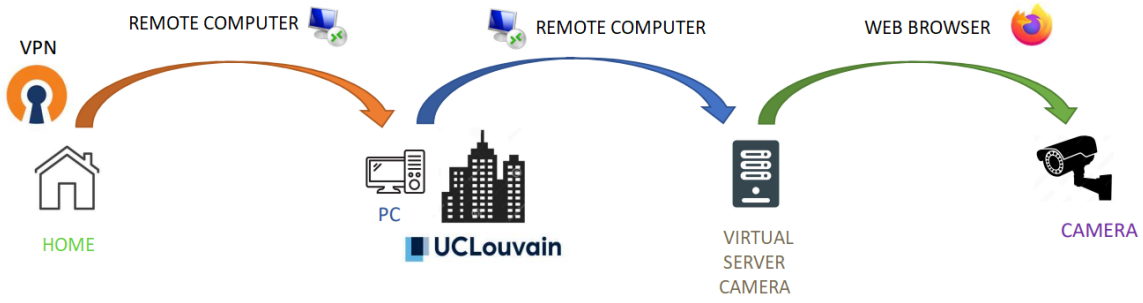


Figure 5.2: Way to access the auditorium camera

5.2.2 Data collection

To efficiently collect the images we created a python script that was automatically executed by the windows task scheduler twice at each course in the A10 auditorium, once a bit before the course and once during the course to have a different number of people. The images collected were images from the camera of size 1440x1080, similar to the one in figure 5.3.

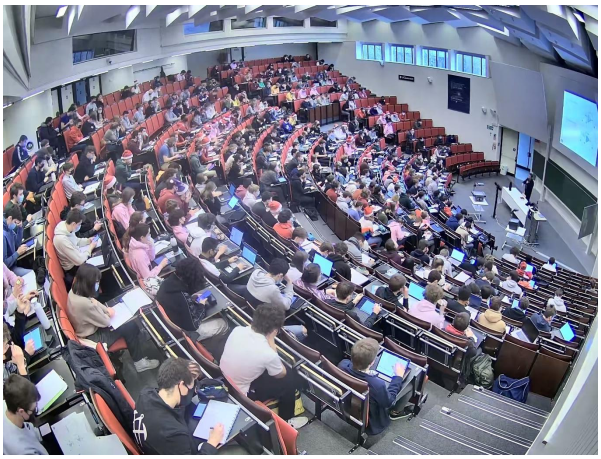


Figure 5.3: One of the largest auditorium at the UCLouvain

In the end, we created a dataset with a total of 152 annotated images.

5.2.3 Data annotation

To annotate the data, we used a Matlab script [35] that allows to manually click in the centre of each head via the Matlab image processing toolbox as shown in figure 5.4. The script then create a .mat file with the coordinates of each pointed head similar to the ones from the ShanghaiTech and UCF datasets.

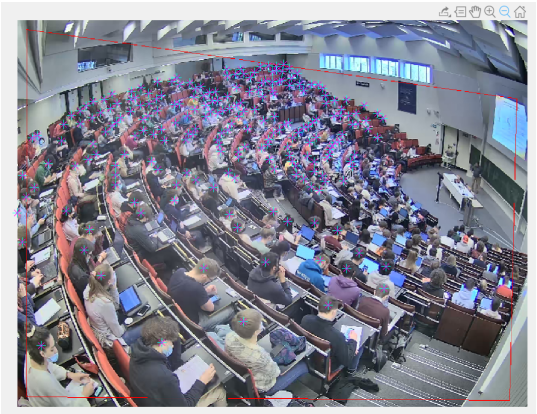


Figure 5.4: Overview of the way to annotate the data

5.3 Data augmentation

An idea to improve the quality of the model when there are few data is to use **data augmentation**. It permits to have more data for training the model and to reduce overfitting. In our case, we flipped the pictures and the ground truth horizontally such that the model can learn new associations of head. It is also possible to change the brightness of the pictures to represent all the moments of the day. It is an easy way to produce more data from an image and its ground truth. In addition, the operations applied on the ground truth are simple and the OpenCV[3] Python package supplies functions for the image's modifications. An example of data augmentation is available in figure 5.5

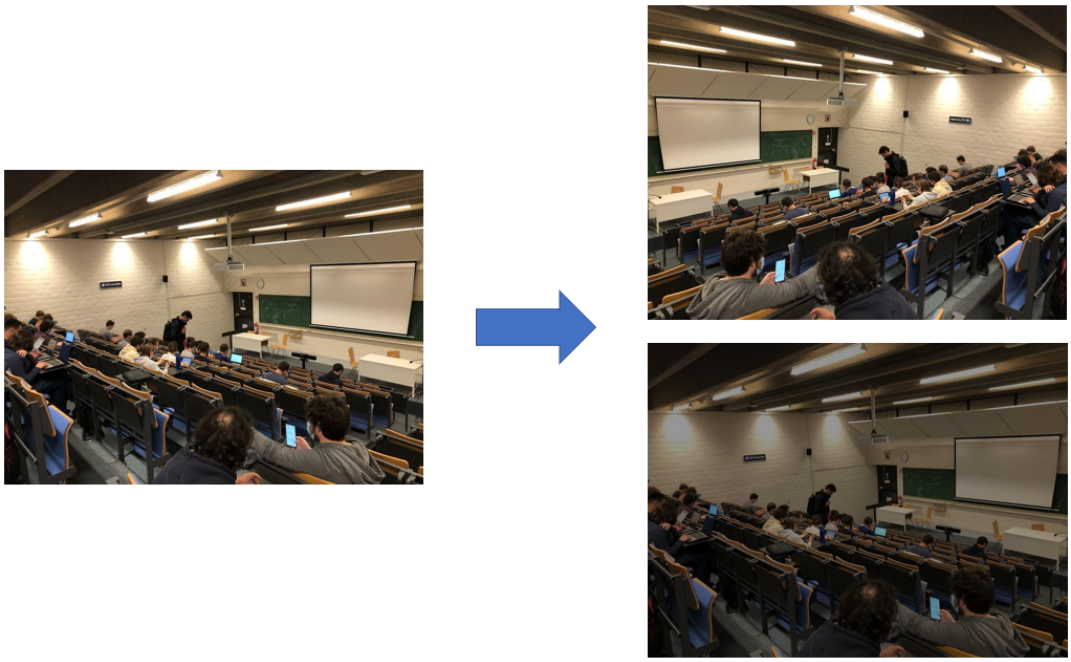


Figure 5.5: Example of data augmentation

[3]https://opencv.org/

Chapter 6

Performance and experiments

This section present the main results of the adaptation of CSRNet to the counting in UCLouvain auditoriums. The first subsection details the different metrics used to assess the performance of each model. The second section compares the performance of the different models and discuss the utility of the variation of different parameters.

6.1 Evaluation

In order to analyse the performance of the model, we need some metrics to compare them. They will also be useful to compare models with each other when the parameters of the models differ. The two metrics that are used are the Mean Absolute Error (MAE) and the Mean Squared Error (MSE) [3]

$$MAE = \frac{1}{N}\sum^{N} i = 1|C_i - C_i^{GT}| \tag{6.1}$$

$$MSE = \sqrt{\frac{1}{N}\sum^{N} i = 1|C_i - C_i^{GT}|^2} \tag{6.2}$$

Where N is the number of images, C_i is the predicted count of people in image i and C_i^{GT} is the real number of people in image i. The method for computing C_i is the following :

$$C_i = \sum_{l=1}^{L}\sum_{w=1}^{W} z_{l,w} \tag{6.3}$$

The computation of the estimated count needs the length and the width of the density map (respectively L and W) and $z_{l,w}$ which represent the pixel at position (l,w) in the generated density map.

6.2 Results

Now that we have trained models and metrics to analyse its performances, we need a test set. The test set is a new set of images that have not been used during the training so that the model does not know them. This dataset is composed of 31 representative images of the different cases such as crowded/non-crowded scenes of different brightnesses in the A10 auditorium. All the following results can be reproduced from the notebooks on the GitHub repository at Appendix B.3.3

Note that the images are confidential and the datasets are not present in the GitHub repository. If someone wants to reproduce the results or to use the datasets, it is possible to request them to Pierre Schaus which is Professor at UCLouvain.

As a first attempt, we took the model already trained on the Shanghai dataset. This model that we called "SH_C" has the following results on the validation set :

- MAE = 22.47
- MSE = 664.89

These are not bad results but it is normal that they are not very good given that the model has never been trained on auditorium images.

6.3 Experiments

In order to have the best performances, we tried to determine the influence of several parameters, following several questions :

- Does training the model on general datasets (such as ShanghaiTech) improve the performances ?
- Does the specific training on the auditoriums improve the performances ?
- Does the model perform better when it is trained with data augmentation ?
- Is a model specific to an auditorium or does it have good performance on other auditoriums ?
- Is the model equally performing on really crowded and nearly empty images ?

Notations

To easily recognize the characteristics of the different models we introduce a notation consisting of several prefixes separated by the "_" character. Each prefix contained in the name of a model correspond to one of the characteristic of this model. The different prefixes are :

- **SH** means that the model was first trained on the ShangaiTech dataset part A (section 5.1) before any other training, with the exact same parameters as the creators of CSRNet [3].

- **A10** means that the model was trained on the images from the A10 auditorium (section 5.2).
- **DA** means that the model was trained on the images from the A10 auditorium and the images created from them based on our data augmentation methods (section 5.3)
- **C** means that the ground truth of the training images was generated using the geo-adaptive kernel and not a Gaussian with a static standard deviation (section 4.1)

6.3.1 Impact the training set

Let's first see if the specific and general training have an impact on the performances. We introduce two new models that are "A10_SH_C" and "A10_C" so that we have the different configurations.

Impact of general training

Regarding the impact of the general training, we can see at Table 6.1 that with or without it, the performance of the models trained with a specific training are almost the same when tested on new images of the auditorium.

Impact of specific training

If we look again at Table 6.1, we can see that for a specific application, the specific training is important to have better results. Indeed, the error on the model trained only on the ShanghaiTech dataset is 4-5 times higher than the ones trained on the auditoriums.

Model	MAE	MSE
SH_C	22.47	664.89
A10_C	6.17	72.74
A10_SH_C	5.55	65.46

Table 6.1: Comparison of models on A10 dataset

After these observations, we now have two models that seem to be working well.

6.3.2 Impact of data augmentation

As explained earlier, we first had to collect data in order to build our dataset and to be able to train our model. We collected data during a few months and we thus not have a lot of images. We wondered if performance could be improved with more images so we did some data augmentation (section 5.3).

We introduce 2 new models which are "A10_DA_C" and "A10_SH_DA_C". Our first assumption was that data augmentation would improve the models but if we look at Table 6.2 we see that it does not improve the models results.

Model	MAE	MSE
A10_C	6.17	72.74
A10_SH_C	5.55	65.46
A10_DA_C	16.24	341.63
A10_SH_DA_C	10.76	213.5

Table 6.2: Comparison of models on A10 dataset

The authors of CSRNet also proposed a method of data augmentation. It consists of cropping the images in 9 parts but they found out that the results were better without using it.

This new experience does not lead to new performant models and our two best models are still in the competition.

6.3.3 Inference on other auditoriums

We have two models that perform well on an auditorium, but it is better if they also work well on all the others so that you do not have to train a different model for each auditorium.

Test set

In order to assess the performance on other auditoriums, we needed to build a test set composed of images from smaller auditoriums than the A10. On this purpose, we received 13 images of a middle-sized auditorium (SUD 11 at UCLouvain) from our supervisor (Pierre Schaus) and we took 11 pictures from a small auditorium (Barb94 at UCLouvain). All these pictures have different crowd densities and luminosities. Typical images from the test set are shown in figure 6.1

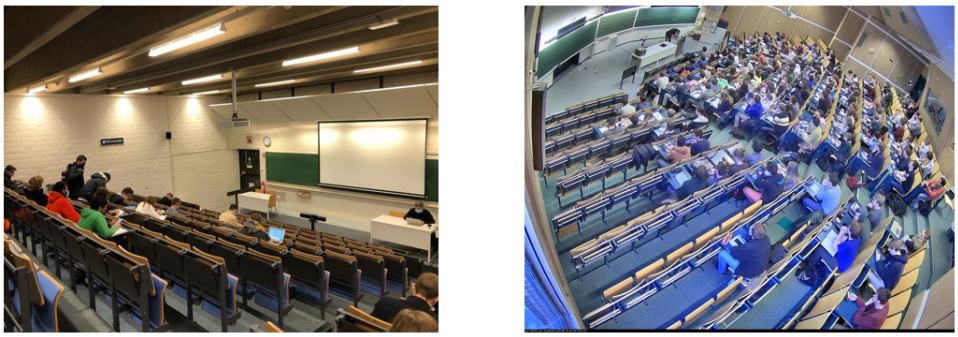


Figure 6.1: Images from the test set

Performances

Table 6.3 shows the MAE and MSE of the different models on the new datasets. One can notice that our two models perform well on the Barb images but on the Sud ones, the model non-trained on the Shanghai images is less good. Even if at first sight, training the model on the Shanghai images did not give much difference, we can see here that this training allows the model to be more versatile and consistent.

However, the "A10_C" model performs very well on the Barb dataset which is from the same point of view as the A10 images and badly on the Sud images which are not from a similar point of view. The point of view probably has an influence on the performance, the same experiment should be done on other auditoriums in order to validate this hypothesis.

Model / Dataset	Barb		Sud	
	MAE	MSE	MAE	MSE
SH_C	5.6	34.94	12.69	252.07
A10_C	1.93	8.33	15.81	382.29
A10_SH_C	4.75	27.68	7.42	105.6

Table 6.3: Comparison of models on Barb and Sud datasets

Since we want to be able to count people in all kinds of auditoriums in UCLouvain, the best model so far is the "A10_SH_C".

6.3.4 Impact of these modifications on general data

After the previous experiments, we were able to determine which model was the most efficient and versatile. This model is the one that was trained on the ShanghaiTech and A10 datasets. We then asked ourselves if this model still performs well on the ShanghaiTech images. In order to see how this evolves with the models, we also put the model that was discarded in the previous step.

Model	MAE	MSE
SH_C	65.96	11023.54
A10_C	234.63	125013.06
A10_SH_C	137.46	44621.56

Table 6.4: Comparison of models on ShanghaiTech datasets

As visible at Table 6.4, compared to the model trained only on the ShanghaiTech dataset, our two models performs badly. However, we can see that the model trained, in a first time on Shanghai images, is quite better than the one only trained on the auditoriums. So, training the model first on the Shanghai images and then on the auditoriums images leads to bad results for images on the first dataset in the end. Is it possible to mix the two dataset in order to improve the global accuracy of our model ?

6.3.5 Combination of general and specific training

For this experiment, we trained a new model but the dataset used is a mix of the Shanghai and A10 images. We called it "A10xSH_C". Its performances are shown at Table 6.5.

Dataset	MAE	MSE
A10 images	7.93	79.11
Barb images	8.71	82.37
Sud images	10	140.9
Shanghai images	70.13	12483.35

Table 6.5: Performances of A10xSH_C

This model is the most polyvalent model : it has good performances on all kinds of images. However, it is never the best model for any of the particular task since the influence of all kinds of images was the same during its training.
This model can be useful if we want to generalize to really different auditoriums. Especially if the cameras have a really different point of view than the training images (for example in the front of the auditorium).

6.3.6 Number of people

The goal of this section is to measure the performance of previously introduced models in function of the number of people on the images. In order to do that we separated the test set composed of A10 images in 3 categories : non-crowded images (less than 50 people), middle-crowded images (between 50 and 200 people) and crowded images (more than 200 people). The characteristics of these test sets are available at Table 6.6.

	nbr of images	mean nbr of people
non-crowded	15	20
mid-crowded	11	127
crowded	5	268

Table 6.6: characteristics of the datasets

Table 6.7 shows the results of the analysis. For all models, the error percentage is better when the number of people on the picture is high. We can also notice that the models pretrained on the Shanghai dataset generally perform better on crowded scenes but their performance is less good on images with few people. It is probably because all images from the Shanghai A dataset are crowded images.

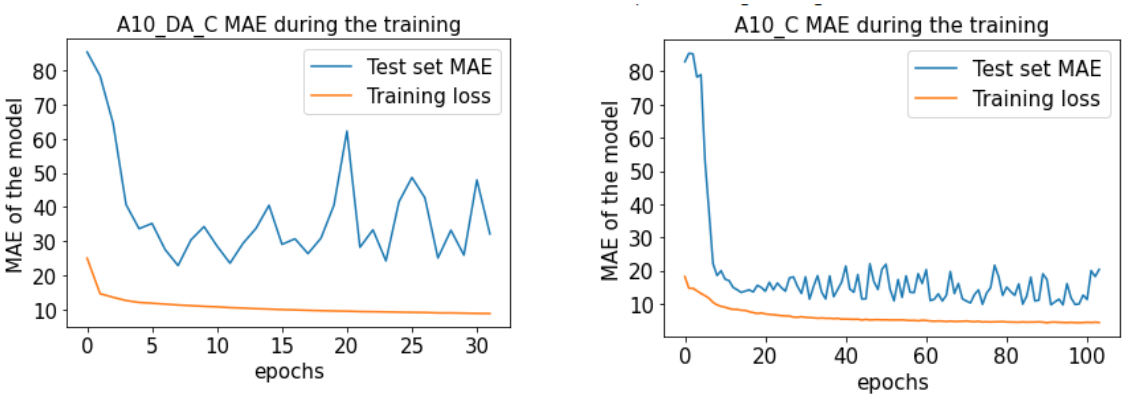
6.3.7 Training stop criterion

In order to know when to stop the training of our models to find the good balance between good training and overfitting, we carefully analysed the MAE on the validation set during the training. Figure 6.2 compare the MAE of the A10_C and A10_DA_C models, the two models trained without shanghai data. It is interesting to notice that the model trained using data augmentation seems to converge faster to its minimum MAE than the other model. Also,

Models	MAE (mean error percentage)		
	non-crowded	mid-crowded	crowded
SH_C	8.7 (43.5 %)	36.3 (20.7 %)	30 (11.2 %)
A10_C	4.5 (22.2 %)	6.1 (4.8 %)	11.1 (4.2 %)
A10_SH_C	4.5 (22.3 %)	7.5 (5.9 %)	7.1 (2.6 %)
A10xSH_C	5.8 (29 %)	9 (7.04 %)	11.9 (4.44 %)

Table 6.7: Results of the crowd analysis

since the test set is really similar to the training set (both A10 images), the models don't show clear signs of overfitting even when the test set MAE is not improving. However, it could happen if we continue the training for more epochs.



(a) With data augmentation

(b) Without data augmentation

Figure 6.2: MAE of the models trained from scratch

We can also analyse the impact of the Shanghai pretraining on the convergence of the model. Figure 6.3 shows the training MAE of the A10S model trained for the interface (See section 7.1.4).The model takes more epochs to converge to its MAE. It is probably because it was trained on the Shanghai images that are really different from the images from the test set.

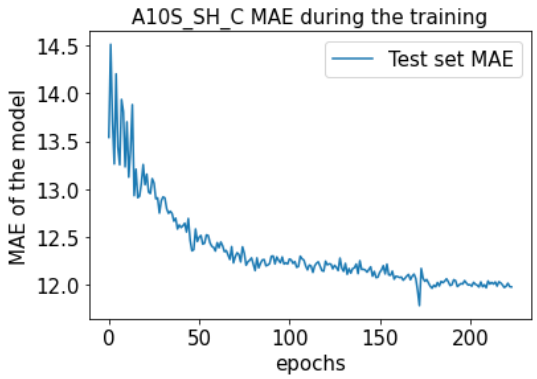


Figure 6.3: MAE of the A10S model

6.3.8 Results

The best models are “A10_SH_C” and “A10xSH_C”. The first one performs the best on auditorium images while the second one seems to be the most versatile. A versatile model can be very useful if you are dealing with many different auditoriums and points of view.

There is a summary of the performances of all trained models on the different datasets at Table 6.8.

Model / Dataset	A10		Barb		Sud		Shanghai	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
SH_C	22.47	664.89	5.6	34.94	12.69	252.07	65.96	11023.54
A10_C	6.17	72.74	1.93	8.33	15.81	382.29	234.63	125013.06
A10_SH_C	5.55	65.46	4.75	27.68	7.42	105.6	137.46	44621.56
A10_DA_C	16.24	341.63	25.3	661.56	30.22	1214.74	250.68	141467.97
A10_SH_DA_C	10.76	213.5	13.55	192.12	24.07	990.26	206.43	80263.06
A10xSH_C	7.93	79.11	8.71	82.37	10	140.9	70.13	12483.35

Table 6.8: Comparison of models on Barb and Sud datasets

Chapter 7

Accessibility of the model

Another goal of this master thesis is to make the model easily available. For doing this, we created an online website interface and a Python library.

7.1 Online interface

In order to have the model available from nowhere, we deployed a web interface that allows the user to predict how many people are on a picture. We experimented two approaches. The first one works as a client-server approach so that the server receives the images and perform the computation. The second one is a client-only approach where the computation is made on each client's machine.

Both approaches have been developed in NodeJS and the source codes are accessible at Appendix B.3.1.

7.1.1 User interface

Whatever the approach used, the user will see the same interface and will have several possibilities :

- The user can choose a model among the three available. There are two (general) models trained on Shanghai dataset for crowded pictures, and our model trained on the auditoriums. For the general models, one is faster than the other but is thus less accurate so that the user can choose if he prefers the speed or the accuracy.
- It is possible to download a picture directly from the machine.
- It is possible to take a live picture.

An overview of the final interface is shown in figure 7.1 and it can be accessed at Appendix B.2.

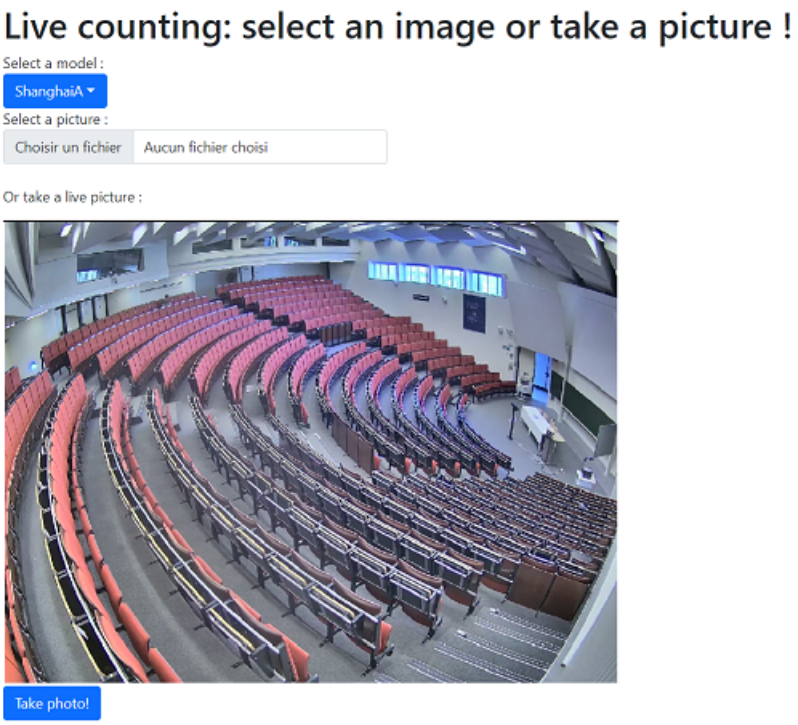


Figure 7.1: Overview of the interface

7.1.2 Computation method

Client-Server approach

In this approach, the client interface sends a request to the server that does all the computations. Once the server has received the image through HTTP, it will load the model and launch a python process. This python process will call the predict method of the model to obtain the prediction (the same method as the one available in the python package).

Client only approach

The idea behind this approach is that the model is on the client side, the prediction can be computed by the client device. This means that even if several users use the site at the same time, the server will not be overloaded by thousands of predictions at the same moment. If we want to dispense with the server, we need to find another way to predict the number of people in a photo. Here are the different steps to follow.

- Conversion of the model into JavaScript : To make the model usable by a browser, we need to make it usable in JavaScript. We used the TensorFlowJS library[1], following the method described in this article [38]. The conversion can be easily done following the notebook available at Appendix B.3.3. The final form of the model is obtained using the TensorFlowJS library. The model, which is composed of a JSON file and binary files that will be used to load the weights of the model, is usable in JavaScript.

- Importation of the model in JavaScript : To be able to use the model, we have to import it in JavaScript and we were limited by the importation options in the TensorFlowJS library. The solution that we use is to launch a small HTTP server containing the model. The program will download the model from this server when loading the page.

- Prediction computation : Make a prediction from a given image is a process in three steps :

  - Choose a resolution (see section 7.1.4) and resize the image to this resolution since the model can only make prediction within a given resolution under its TensorflowJS form.
  - Normalize and standardize the tensor of pixels. It improves the performances of the predictions.
  - Give the tensor to the model and wait for the prediction.

    It is interesting to notice that the few first computations take longer to compute than the next ones because the model have to warm up.

7.1.3 Deployment of the application

In the end, we chose to deploy the client-only application mainly because it is easier to manage given that all the computations are done locally on the user's machine.

The interface has been deployed using GitHub pages[2]. It is a static site hosting service that takes HTML, CSS, and JavaScript files straight from a repository on GitHub.
The server hosting the different used models is 000webhost[3]. It is a free web hosting that allow to get the models files via an HTTP request.

[1] https://github.com/tensorflow/tfjs-converter/blob/master/tfjs-converter/README.md
[2] https://pages.github.com/
[3] https://www.000webhost.com/

7.1.4 Performance of the application

The time to obtain a prediction on a 1024x768 images was really long (about 10 minutes). However, one of the goal of the application is to have a live prediction of the number of people on an image. In order to understand why it was so slow, we measured the CPU and memory consumption during the prediction and got the following results :

- The memory use was approximately the same for all models and didn't explode due to predictions.
- The available CPU was always fully used during all the time of a prediction. Between 70 and 80 % of the time spent in calculations was spent in the convolutional layers.

Based on that, we concluded that the limitation factor is the CPU consumption. In order to reduce the time spent in calculations in the convolutional layers and by the same occasion the total prediction time, we tried to make predictions with smaller image resolutions. Figure 7.2 shows the mean prediction time of the model for different images resolution. The measures were taken with the interface opened in Google Chrome, on a computer with Intel Core i7-7500U CPU and 8 GB of RAM.

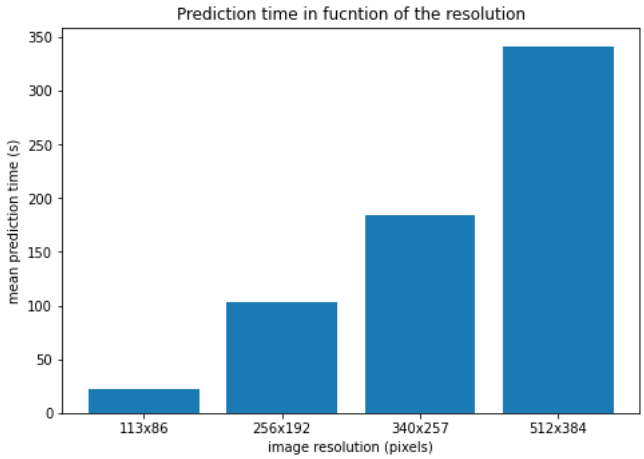


Figure 7.2: Mean computation time in function of the image resolution

However, if the images have a lower resolution, there is less information in the image and the prediction are less accurate. The challenge is then to have the best accuracy with a reasonable prediction time. In order to find the most suitable model, we tested four different models on the different resolutions :

- **Shanghai A** This model is trained on the Shanghai A dataset, the exact same way as the creators. This model is taken as a basis to compare and interpret the results of the other models.
- **Shanghai_A_small** This model is trained on the images of the Shanghai A dataset, but resized to a size 340x257 pixels. The aim of this training is to see if training the model on smaller images would improve the quality of the predictions on images with a smaller resolution.

- **A10_SH_C** This model is the model chosen in the section about experiments 6.3. The aim of this model is to see if training models for specific situations would improve the model accuracy on images with small resolutions.
- **A10_SH_C small** This model is first trained on the Shanghai A dataset with the normal resolution. Then it is trained on the A10 images but with a resolution of 340257 pixels. The aim of this training is to see if the specific training in a small resolution would improve the results of the prediction.

Influence of training on small images

In order to evaluate the relevance of the trained models, we evaluated their MAE on different sets used in section 6.3. Table 7.1 shows the MAE of the basic and small Shanghai model on the ShanghaiA test set, with different resolutions. It is interesting to notice that the small models performs better on small images and might therefore be interesting.

Model / resolution	Basic	340x257
Shanghai A	66	123.5
Shanghai_A_small	260	92

Table 7.1: MAE of the models in function of the resolution of the test set

To clearly understand the advantages and disadvantages of each model, we also analysed the results of three images with different characteristics. One of the Shanghai Test dataset (90 peoples), one of a small auditorium (14 peoples) and one of a big auditorium (281 people).

Figure 7.3 shows the prediction of the models on the three testing images. Even though the small model have better results on the small resolution images, it seems to be more dependent on the training set. In fact, it predicts a large number of people even on the non-crowded images, probably because there is always a lot of people on all the images of the training set. It is also interesting to notice that, as we could expect, results on images with a higher resolution are in general better than results on a smaller resolution.

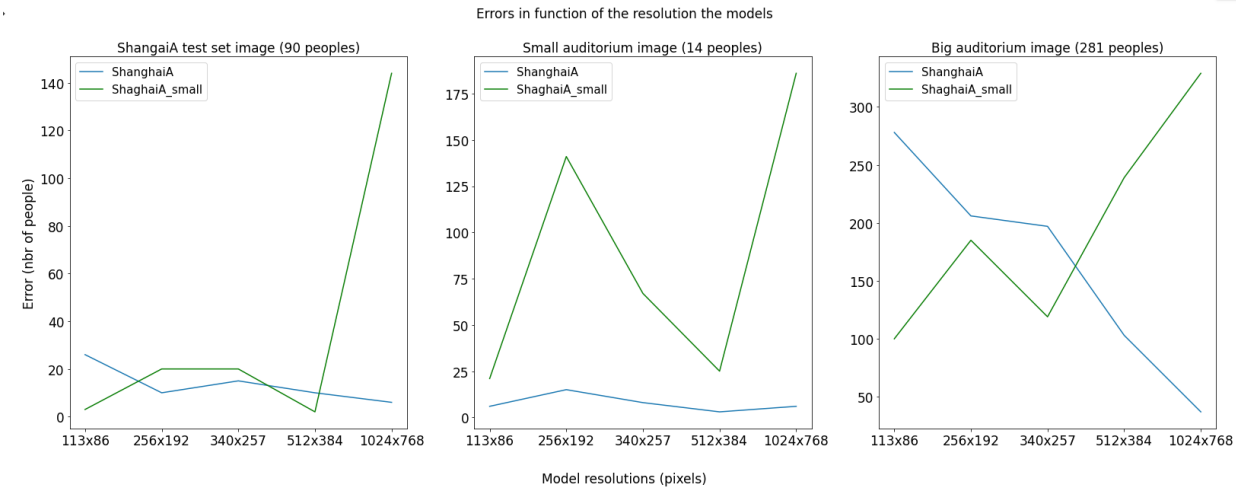


Figure 7.3: Performance of the models on the testing images

Influence of specific training

Table 7.2 shows the MAE of the different models on the A10 test set with different resolutions. Based on the results, we can draw several types of conclusions :

- Even though specific training in the basic resolution improve the accuracy of the prediction for normal images, it doesn't improve the accuracy on smaller resolutions
- Specific training in small resolution improves the results on small images

Model / resolution	1024x768	340x257
Shanghai A	18.4	52.1
A10_SH_C	6	73.7
Shanghai_A_small	623	65
A10_SH_C small	106.1	13.6

Table 7.2: MAE of the models in function of the resolution of the test set

Figure 7.4 shows the prediction of the models trained on the basic resolution on the same three testing images as in the previous section. One would notice that the results of these models in resolutions smaller than 512x384 are not really accurate (more than 50 % error), even if the model was pretrained on auditoriums.

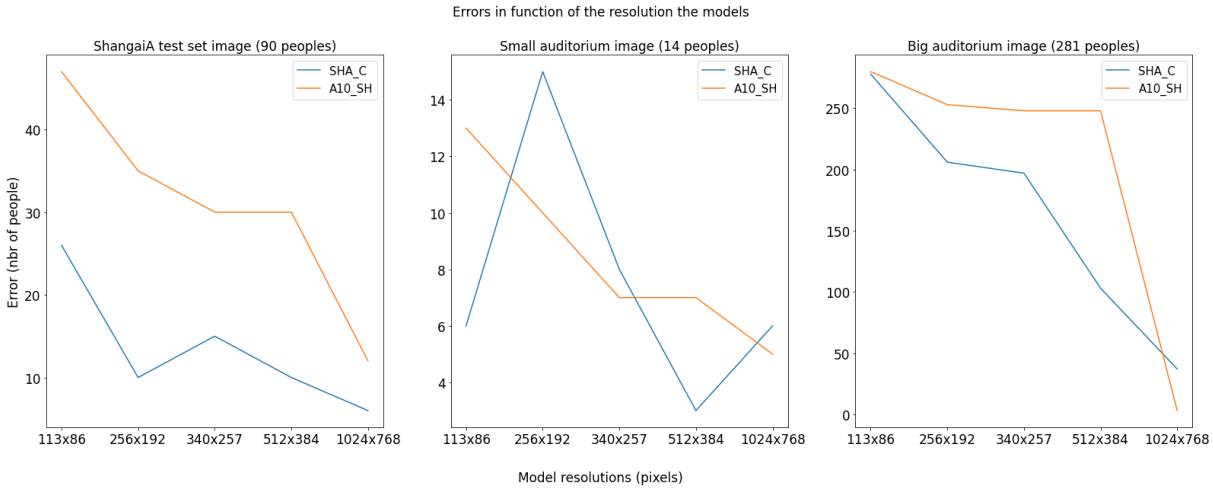


Figure 7.4: Performance of the models on the testing images

7.1.5 Conclusion

This in-depth analysis of the performances showed that it is impossible to obtain as good results for small resolutions as for big resolutions. That is why we decided to make three models available with different characteristics on the interface. It permits the user to choose by himself the balance between waiting time and precision.

7.2 Python crowd counting library

Objective

The goal is to allow people interested in crowd counting to easily have access to trained models and to functions to train their own. Moreover, one of the models already loaded in the library is our model trained on the auditoriums. Since then, this could be used by people from UCLouvain that would need to count the number of people in the auditorium whatever their goal.

Content

We created a python package called Crowd-counting that can be easily installed using the 'pip' command. The package fulfils both its objectives and allows the possible actions :

- Make a prediction using already well-trained models, especially on the auditoriums or new loaded model.
- Train new models with annotated data
- Create the ground truth in the right format from images and their annotations.
- Do some data augmentation on new images.
- Evaluate the performance of models using new images. The function returns the MAE and MSE.

Documentation

In order to easily get to grips with the library, we made a documentation using Sphinx[4], a tool for creating documentation. The full documentation of the library is available on ReadTheDocs (see Appendix B.1) and the code of it can be accessed at Appendix B.3.2.

[4]https://www.sphinx-doc.org/en/master/

Conclusion

The initial objective of this thesis was to obtain performing models for crowd counting on auditoriums, based on the images from the security cameras, and to make them easily accessible. After learning about the different existing methods, we looked in more detail at the one that seemed the most relevant, the CNN-based approach. In this method, it creates a density map that can be summed to know the number of people. Then, we took an existing architecture, called CSRNet, to train our own models.

However, we needed images of the auditoriums for this. On the one hand, we contacted the UCLouvain in order to have access to the images of the security cameras. On the other hand, we had to annotate the collected images so that the model would know how many people are in the pictures. These pictures can be easily annotated using Matlab.

When all this was done, we trained and analysed different models that had been trained in different ways. This allowed us to determine which models are the most efficient and which are the most versatile, i.e. best suited to situations other than the one intended. The results show that a specific training is necessary to obtain good results on a specific case. In this case, training the models on images from a specific auditorium helps to improve their performances on all kinds of auditoriums. Moreover, a model will be more versatile if it is also trained on general data such as the ShanghaiTech dataset. Our latest experiment in which we train a model on both types of images is quite versatile while keeping a good accuracy.

Finally, in order to make the work we have done more accessible, we have created a python library which we hope will one day be useful to someone who wants to start crowd counting. We also deployed an interface that allows anyone to get an estimate of the number of people in a picture.

Bibliography

[1] S. Di Domenico, G. Pecoraro, E. Cianca, and M. De Sanctis, "Trained-once device-free crowd counting and occupancy estimation using wifi: A doppler spectrum based approach," in *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 1–8, 2016.

[2] C. C. Loy, K. Chen, S. Gong, and T. Xiang, *Crowd Counting and Profiling: Methodology and Evaluation*, pp. 347–382. New York, NY: Springer New York, 2013.

[3] X. Z. Yuhong Li and D. Chen, "Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes.," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] J. Bryden, "Biologically inspired computing: The neural network." https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.4731&rep=rep1&type=pdf. Accessed: 22-04-18.

[5] L. Chu, "Human neuron vs. artificial neuron: Similarities and discrepancies." https://lanstonchu.wordpress.com/2021/09/06/human-neuron-vs-artificial-neuron-similarities-and-discrepancies/, 2021. Accessed: 22-04-18.

[6] L. Pauly, H. Peel, S. Luo, D. Hogg, and R. Fuentes, "Deeper networks for pavement crack detection," 07 2017.

[7] M. Verleysen and J. Lee, "Lelec2870 - machine learning," 2020 -2021. UCLouvain.

[8] S. Ojha and K. Santos, "Parallelizing gradient descent on different architectures." https://shashank-ojha.github.io/ParallelGradientDescent/proposal.pdf, 2018. Accessed: 22-04-18.

[9] IBM education, "What is gradient descent?." https://www.ibm.com/cloud/learn/gradient-descent, October 27, 2020. Accessed 01/05/2022.

[10] G. Castellano, "Intelligent interactive systems," 2021-2022. Uppsala university.

[11] IBM education, "What are convolutional neural networks?." https://www.ibm.com/cloud/learn/convolutional-neural-networks, October 20, 2020. Accessed 01/05/2022.

[12] IBM education, "What is padding?." http://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html, August 17, 2020. Accessed 01/05/2022.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Apr 2015.

[14] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," 2015.

[15] IBM education, "What is overfitting." https://www.ibm.com/cloud/learn/overfitting, March 3, 2021. Accessed 21/05/2022.

[16] "Regression vs. classification in machine learning." https://www.javatpoint.com/regression-vs-classification-in-machine-learning.

[17] H. Zou, Y. Zhou, J. Yang, and C. J. Spanos, "Device-free occupancy detection and crowd counting in smart buildings with wifi-enabled iot," *Energy and Buildings*, vol. 174, pp. 309–322, 2018.

[18] S. Depatla and Y. Mostofi, "Crowd counting through walls using wifi," in *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–10, 2018.

[19] S. Liu, Y. Zhao, F. Xue, B. Chen, and X. Chen, "Deepcount: Crowd counting with wifi via deep learning," 2019.

[20] C. Jiang, M. K. Masood, Y. C. Soh, and H. Li, "Indoor occupancy estimation from carbon dioxide concentration," *Energy and Buildings*, vol. 131, pp. 132–141, 2016.

[21] S. K, "Non-maximum suppression (nms)." https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c, 2019. Accessed: 22-04-18.

[22] M. Li, Z. Zhang, K. Huang, and T. Tan, "Estimating the number of people in crowded scenes by mid based foreground segmentation and head-shoulder detection," *2008 19th International Conference on Pattern Recognition*, pp. 1–4, 2008.

[23] A. B. Chan and N. Vasconcelos, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, pp. 1627–1645, 2010.

[24] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision volume*, vol. 57, pp. 137–154, 2004.

[25] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[26] A. B. Chan and N. Vasconcelos, "Bayesian poisson regression for crowd counting," *2009 IEEE 12th International Conference on Computer Vision*, pp. 545–551, 2009.

[27] H. Idrees, I. Saleemi, C. Seibert, and M. Shah, "Multi-source multi-scale counting in extremely dense crowd images."

[28] "Crowd counting benchmarks." https://paperswithcode.com/task/crowd-counting. Accessed: 22-04-28.

[29] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *towards data science*, vol. 6, no. 12, pp. 310–316, 2017.

[30] J. Brownlee, "A gentle introduction to transfer learning for deep learning." https://machinelearningmastery.com/transfer-learning-for-deep-learning/, 2017. Accessed: 22-04-18.

[31] D. B. Sam, S. Surya, and R. V. Babu, "Switching convolutional neural network for crowd counting," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4031–4039, 2017.

[32] L. Boominathan, S. S. S. Kruthiventi, and R. V. Babu, "Crowdnet: A deep convolutional network for dense crowd counting," *CoRR*, vol. abs/1608.06197, 2016.

[33] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013.

[34] A.N.Rajagopalan, "Lec 08 - bilinear interpolation." https://www.youtube.com/watch?v=UhGEtSdBwIQ. Accessed: 22-04-12.

[35] princenarula222, "Crowd_annotation." https://github.com/princenarula222/Crowd_Annotation.

[36] Y. Zhang, D. Zhou, S. Chen, S. Gao, and Y. Ma, "Single-image crowd counting via multi-column convolutional neural network," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 589–597, 2016.

[37] V. Lempitsky and A. Zisserman, "Learning to count objects in images," in *Advances in Neural Information Processing Systems* (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), vol. 23, Curran Associates, Inc., 2010.

[38] N. P. Hadash, "Exporting and running a deep learning model in the browser (including lstm?) - a straightforward guide." https://bit.ly/3Nvc9ct. Accessed: 22-03-25.

[39] J. Brownlee, "Difference between classification and regression in machine learning." https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/. Accessed: 22-06-01.

Appendix A

Classification and regression

Regression and classification are two kinds of problems that can be solved using machine learning [39].

Classification models are used to predict discrete class/values (for example : True or false, blue or red). The algorithm learn a function that will divide the dataset into classes based on different parameters.

Regression models are used to predict continuous values such as a future salary etc. The algorithm learn a function that reflects the correlation between the input variables and the output variable. Figure A.1 shows a graphical representation of the difference between classification and regression.

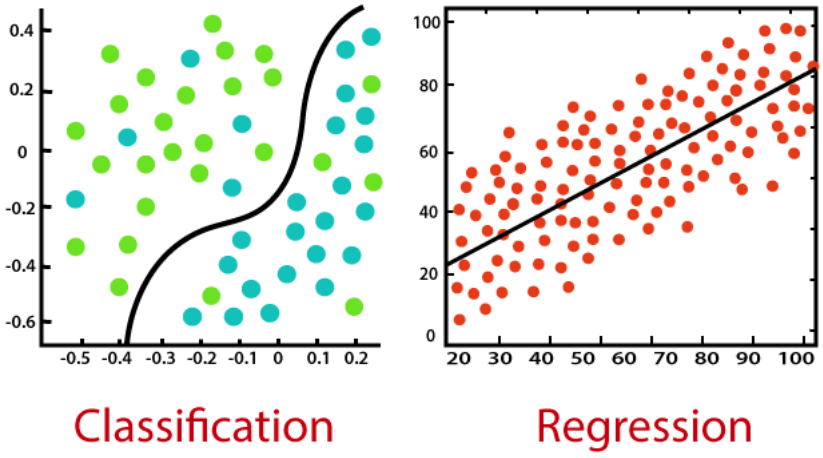


Figure A.1: Difference between classification and regression [16]

Appendix B

Online resources

B.1 Library documentation

The package documentation is available here : https://crowd-counting.readthedocs.io/en/latest/

B.2 Web interface

The online interface is available here : https://juski07.github.io/livecounting/

B.3 GitHub repositories

B.3.1 Online interface

Available at https://github.com/Juski07/livecounting

B.3.2 Library

Available at : https://github.com/lrobins1/crowd_counting

B.3.3 Main repository

The repository containing the models, the files to reproduce the results, ... is available at : https://github.com/Juski07/TFE-Crowd-counting.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**