Computer Science and Information Systems 17(3):959–982 https://doi.org/10.2298/CSIS200115029P

Crowd counting á la Bourdieu ⋆

Automated estimation of the number of people

Karolina Przybylek[1] and Illia Shkroba[2] ⋆⋆

[1] University of Warsaw, Warsaw
00-721 Podchorazych 20, Poland
karolina.m.przybylek@gmail.com
[2] Polish-Japanese Academy of Computer Technology, Warsaw
02-008 Koszykowa 86, Poland
is@pjwstk.edu.pl

Abstract. In recent years, sociologists have taught us how important and emergent the problem of crowd counting is. They have recognised a variety of reasons for this fact, including: public safety (e.g. crushing between people, trampling underfoot, risk of spreading infectious disease, aggression), politics (e.g. police and government tend to underestimate the number of people, whilst protest organisers tend to overestimate it) and journalism (e.g. accuracy of the estimation of the ground truth supporting an article).
The aim of this paper is to investigate models for crowd counting that are inspired by the observations of famous sociologist Pierre Bourdieu. We show that despite the simplicity of the models, we can achieve competitive result. This makes them suitable for low computational power and energy efficient architectures.

Keywords: crowd counting, deep learning, mall dataset, habitus.

1. Introduction

Due to the increasing degree of urbanisation crowd management and control is a key issue for human life and security. For example, stampedes at the Kumbh Melas used to kill hundreds of people each event, until appropriate crowd management policy has been applied. In fact, many such problems related to crowds can be prevented or completely solved if aspects of crowd management and control are well organised. Crowd estimation is the first and foremost task in every crowd management process. Other crucial applications of crowd counting include politics and journalism: the number of people that take part in a given event indicates the strength and the impact of the event. Moreover, in our democratic world the number of people is the chief argument in every discussion. For this reason, crowd counts in presidential inauguration ceremonies (e.g. Obama in 2009 vs. Trump in 2017), demonstrations (e.g. "Black march" in 2018 in Poland) or festivals (e.g. LGBT Film Festival in 2011) are highly disputed. Automated crowd counting methods provide objective and indisputable estimation of the size of crowd.

⋆ This paper is an extension of a paper initially published in ADBIS 2019 Workshops proceedings (Springer CCIS 1064).

⋆⋆ The names of the authors are arranged in alphabetical order.

The aim of crowd counting methods is to provide an accurate estimation of the number of people (the crowd) presented on a given picture. This task is extremely challenging for a number of reasons: scaling factor and perspective (people that are nearer to the camera appear much bigger), resolution of fragments containing a single person (i.e. few pixels per person), occlusion and clutter, illumination issues, distinct ambient environments, to name a few. Over last decades we witnessed a significant progress in automated crowd counting.

Two most successful classes of methods for crowd counting are: regression based and detection based methods. Regression based methods aim at estimating the density of the crowd by computing (either directly or indirectly) the heat-map of a picture. For example:

- A state-of-the-art regression model is described in [29].
- A very prominent approach to crowd counting by regression is described in [11]. The authors propose a multi-output regression model that learns how to balance local with global features (i.e. so called "spatially localised crowd counting").
- A variant of the previous method was proposed in [10]. The authors base their approach on the notion of "cumulative attributes", whose aim is to exploit deep correlations between different features.
- The first method capable of reliably counting thousands of people was proposed in [14]. The authors combine three sources of information to construct a regression model: Fourier transformation, interest points identification and head detection.
- A deep convolutional model to estimate the density map of an image is described in [5]. Actually, the main idea of the paper is to combine both deep and shallow convolutional neural networks to achieve good scaling effect.

On the other hand, detection based methods try to recognise some identifying features of each single person on a picture and sum up the number of recognised in this way people. For example:

- State-of-the-art pedestrian detection based on multiple features detection is described in [18]. The authors focuses on the problem of overlapping people as one of the main challenge to successful object-based detection. The method is optimised to count low density crowd and whose individuals have enough detail.
- The authors in [28] describe an end-to-end method of identification of pedestrians on images by using a novel architecture based on LSTM recurrent neural network. The results are competitive on TUDCrossing and Brainwash datasets. One may hope to further improve the method by using dynamic RNNs and Luong attention model as indicated in [7].
- It is well-known that naive transfer-learning does not work well in crowd counting. This issue is investigated in [30]. The authors proposes some methods to overcome such problem.

Our approach is a detection based method with explicit segmentation. We split an image into small segments, build a neural network to estimate the probability that a given segment contains a person, apply a cut-off point value and sum up all of the predictions. This approach was implicitly suggested by Pierre Bourdieu in [6], where he studied the concept of a *habitus*. The concept of habitus in terms of sociological theory of Pierre

Bourdieu is similar to the concept of personality. In fact, habitus could be defined as "social personality" as it especially focuses on the social aspects of the formation of personality. Like personality in psychological terms, habitus is a permanent "generative structure" that organises collective and individual experience. Moreover, it is also a structure that orients people's attitudes and behaviours. There are different type of habitus — e.g. artistic habitus, legal habitus, prison habitus. The shape of these structures depends on the environmental conditions in which an individual functioned in the past and functions now. In other words, habitus depends on the social capital (including economic capital, cultural capital, symbolic capital, etc.). The sociologist argued that habitual complexity of individuals is present at different levels of scale, and, therefore, one may identify a person (or even the social class of a person) by looking only at some of her tiny details in isolation. We show that the models that we build, despite their simplicity, are competitive on Mall Dataset [1] (also see: [9], [10], [19] and [11]). The simplicity of our models makes them suitable for mobile and embedded architectures, where computational power and space are limited. We note that other methods inspired by social observations were successfully used in the past, for example: [4] and its extensions [23], [22].

The source code of our methods is available at a public repository:

https://github.com/s14028/engineer

The paper is the extension of paper [21] initially published in ADBIS 2019 Workshops proceedings. The organisation of the paper is as follows. In the next section we describe the mathematical model behind our estimation method. The section covers the following concepts: the basic idea of a neural network, together with convolutional neural networks, binary cross-entropy loss function, max pooling, and common problems that one may encounter while training neural networks, or selecting models. The experimental environment is described in Section 3. The dataset is described in Subsection 3.1 and our transformations of datasets together with necessary preprocessing (i.e. data segmentation) in the following subsection. Subsection 3.3 describes a perspective map. Subsection 3.4 describes our approach to image augmentation. Subsection 3.5 describes the hardware that we used to train our models. Subsection 3.6 describes the error metrics that we use to assess the quality of the models. In Section 4 we describe three models. the first one is described in Subsection 4.1 and operates on raw images, the second of the models, described in Subsection 4.2, utilises additional information from the perspective map. These two models use the standard binary cut-off point method to decide whether a given segment of a picture contains a person. The third model, described in Subsection 4.3, also utilises additional information from the perspective map, but uses a more advanced continuous cut-off point method. We summarise the results and conclude the paper in Section 5.

2. Neural Networks

As mentioned in the introduction, all our models are represented as, so called, neural networks. A neural network is a mathematical model of computation based on layers of neurons connected together. The aim of this section is to provide basic information about neural networks that are necessary to understand and replicate our results. We will begin this section with the explanation of a model of a single neuron. Then we tell how multiple neurons can be connected to form a layer. Finally, we describe how multiple layers can

be stacked on top of each other to form a neural network. We will also describe some common pitfalls in choosing an appropriate model and in training the models.

2.1. Artificial neuron

A neuron in a neural network is a function $f\colon \mathbb{R}^k \mapsto \mathbb{R}$ of the form as in Equation (1), where $w \in \mathbb{R}^k$ is a vector called the *weights* for neuron f :

$$f(x_1, x_2, \ldots, x_k) = \sum_{j=1}^{k} w_j x_j \tag{1}$$

The function f is be usually written in a "dot product" notation as shown in Equation (2).

$$w \cdot x = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_k \cdot x_k = f(x_1, x_2, \ldots, x_k) \tag{2}$$

Groups of neurons in a neural network are arranged together in "layers". A layer takes some input vector x and forwards it to each of its neurons. The outputs of the neurons are connected together to form a vector y. If a layer has n neurons, vector y has dimension n. Then, vector y is passed to an "activation function" $g\colon \mathbb{R}^n \mapsto \mathbb{R}^n$. A layer can be represented as a pair: a matrix A and an activation function g, as shown in Equation (3), where each row represents a neuron in the layer.

$$A = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1k} \\ w_{21} & w_{22} & \cdots & w_{2k} \\ \vdots & \vdots & \cdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nk} \end{bmatrix} \tag{3}$$

Therefore, a layer $\langle Ag \rangle$ linearly transforms vector x to vector: $g(Ax)$ — i.e. we can think of a layer as a linear transformation composed with some non-linear activation function. If we combine several layers together, we will obtain a neural network. Of course, the dimensions of the consecutive layers have to agree. If one layer is represented as a matrix A of dimension $n \times k$, the following layer represented as a matrix B must have dimension $m \times n$. For example, if the layers have activation functions $g\colon \mathbb{R}^n \mapsto \mathbb{R}^n$ and $h\colon \mathbb{R}^m \mapsto \mathbb{R}^m$ respectively, we get the following network $M\colon \mathbb{R}^k \mapsto \mathbb{R}^m$, that is shown in Equation (4).

$$M(x) = h(B(g(Ax))) \tag{4}$$

2.2. Bias

A *bias* is an additional vector of weights $b \in \mathbb{R}^n$ that is added to the output vector $y \in \mathbb{R}^n$ of a layer just before applying the activation function $g\colon \mathbb{R}^n \mapsto \mathbb{R}^n$. This additional vector

allows the network to "shift" vector y. Equation (5) shows the transformation made by a layer with a bias.

$$g(y + b) = g(Ax + b) \tag{5}$$

Which is equivalent to Equation (6).

$$g(y') = g(A'x') \tag{6}$$

Where $A' \in \mathbb{R}^{n \times k+1}$ is presented in Equation (7).

$$A' = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1k} & b_1 \\ w_{21} & w_{22} & \cdots & w_{2k} & b_2 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nk} & b_n \end{bmatrix} = \begin{bmatrix} A & b \end{bmatrix} \tag{7}$$

And where $x' \in \mathbb{R}^{k+1}$ is presented in Equation (8).

$$x' = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{8}$$

Before moving to a possible interpretation of the concept of bias, let us recall (Equation (9)) another representation for the "dot product" of vectors.

$$w \cdot x := ||w||||x|| \cos \alpha \tag{9}$$

Where $||-||$ denotes the length of a vector, and α is the angle between vectors w and x. If the length of each vector is equal 1, then the dot product is equal to $\cos \alpha$. If the angle between w and x is approaching 0, then the dot product is approaching $||w|| \cdot ||x|| \cdot 1$. If the angle between w and x is approaching π, then the dot product is approaching $||w|| \cdot ||x|| \cdot (-1)$. And if the angle between w and x is approaching $\frac{\pi}{2}$ or $\frac{3\pi}{2}$, then the dot product is approaching $||w|| \cdot ||x|| \cdot 0$.

Let us consider a vector $w \in \mathbb{R}^k$ and assume that $w_i \neq 0$ for some $i \leq k$. Then $w \cdot x$ gives the Equation (10).

$$x_j = -\frac{1}{w_j}(w_1 \cdot x_1 + \cdots + w_{j-1} \cdot x_{j-1} + w_{j+1} \cdot x_{j+1} + \ldots + w_k \cdot x_k) \tag{10}$$

Where $x_1, ..., x_{j-1}, x_{j+1}, ..., x_k \in \mathbb{R}$ are arbitrary real numbers, and x_j can be calculated from the equation. This means, that vector x could be represented by some vector $(x_1, ..., x_{j-1}, x_{j+1}, ..., x_k) \in \mathbb{R}^{k-1}$ together with Equation (10). The subspace $\mathbb{R}^{k-1}$ spanned by all such vectors is called the hyperplane of vector w (it is the hyperplane orthognoal to w).

To get some intuitions about the concept of bias, let us consider the following example. Suppose that we are dealing with a single neuron represented as function $f \colon \mathbb{R}^2 \mapsto \mathbb{R}$.

The input to the neuron consists of vectors in $\mathbb{R}^2$ that belong to two different classes: $\triangle$ or $\bigcirc$. For the vectors of class $\triangle$, the neuron has to assign a value $y \in (-\infty, 0)$, and for vectors of class $\bigcirc$, the neuron has to assign a value $y \in \langle 0, \infty)$ — i.e. the neuron has to separate the classes (this is a typical classification problem). Some sample input vectors are presented on Figure 1.

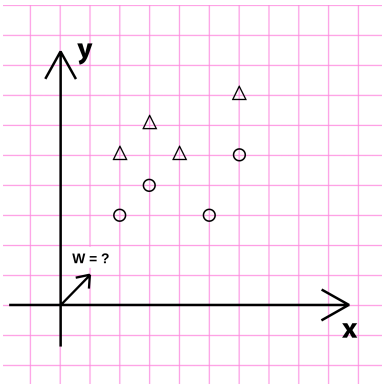


Fig. 1. Vectors of classes $\bigcirc$ and $\triangle$

The weight vector w associated with a neuron for this classification problem, should have an angle greater than $\frac{\pi}{2}$ and smaller than $\frac{3\pi}{2}$ to classify $\triangle$, and should have an angle smaller or equal than $\frac{\pi}{2}$ or bigger or equal than $\frac{3\pi}{2}$ to classify $\bigcirc$. Unfortunately, no vector w can satisfy these requirements.

Bias solves this problem, by extending the weight vector w and the input vector x by an additional dimension. Each input vector x has assigned a constant value 1 in that dimension. This means that each of these vectors live in some $\mathbb{R}^2$ subspace of the extended $\mathbb{R}^3$ space. In contrast, weight vector w has some value $b \in \mathbb{R}\colon b \neq 0$ in that dimension. This allows vector w to have a hyperplane $\mathbb{R}^2$ to differentiate between input vectors.

Consider the intersection of the hyperplane of vector w with the space spanned by the input vectors. Assume that w is extended to an additional dimension with $w_{k+1} = b$ and x with $x_{k+1} = 1$. For each vector $v \in \mathbb{R}^{k+1}$, from the hyperplane of vector w , we can define the value v_{k+1} (11) by using Equation (10).

$$v_{k+1} = -\frac{1}{b}(w_1 \cdot v_1 + w_2 \cdot v_2 + \cdots + w_k \cdot v_k) \tag{11}$$

To obtain the intersection we have to consider the situation when vectors v and x are equal. For some $i \in \{1, 2, \ldots, k\}$ the value $v_i \in \mathbb{R}$ could be equal to value $x_i \in \mathbb{R}$, thus find the intersection, we have to check when $v_{k+1} = 1$, as shown in Equation (12) and Equation (13).

$$-\frac{1}{b}(w_1 \cdot v_1 + w_2 \cdot v_2 + \cdots + w_k \cdot v_k) = 1 \tag{12}$$

$$-\frac{1}{w_j}(w_1 \cdot v_1 + \cdots + w_{j-1} \cdot v_{j-1} + w_{j+1} \cdot v_{j+1} + \cdots + w_k \cdot v_k + b) = v_j \tag{13}$$

Where $v_1, v_2, ..., v_{j-1}, v_{j+1}, ..., v_k \in \mathbb{R}$ could be any real numbers, and v_j depends on v_i for $i \neq j$. Therefore, that vector v can be represented as $(v_1, v_2, ..., v_{j-1}, v_{j+1}, ..., v_k) \in \mathbb{R}^{k-1}$ and Equations (11) together with Equation (13). This shows that the intersection of the hyperplane of w and the space spanned by the input vectors is a subspace $\mathbb{R}^{k-1}$. In our case, when $k = 2$, the intersection is a subspace $\mathbb{R}^1$, which means, that the intersection is just a line.

This leads us to the conclusion, that bias allows us to think of a neuron as a shifted hyperplane dividing the space spanned by the input vectors, as shown on Figure 2.

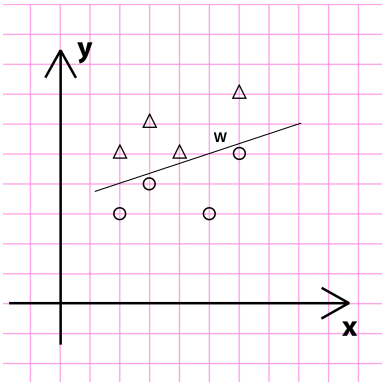


Fig. 2. Separating vectors for classes ◯ and △ with line

2.3. Binary Cross-Entropy Loss

In the process of training, we minimise a loss function E associated with the model by adjusting the weights of the model. A single weight adjustment is called a "step". A single run through the whole training set is called an "epoch". An epoch consists of several steps. A set of inputs associated with the step is called a "batch", and length of these inputs set is called a "batch size".

For example, let us assume, that our training set X along with its true labels D looks as shown on Equation (14) and Equation (15):

$$X = \{I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9, I_{10}, I_{11}, I_{12}, I_{13}\} \tag{14}$$

$$D = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}, d_{11}, d_{12}, d_{13}\} \tag{15}$$

For batch size $m = 2$, some batches are shown in Equation (16) and Equation (17).

$$X/\sim = \{\{I_1, I_2\}, \{I_3, I_4\}, \{I_5, I_6\}, \{I_7, I_8\}, \{I_9, I_{10}\}, \{I_{11}, I_{12}\}, \{I_{13}\}\} \tag{16}$$

$$D/\sim = \{\{d_1, d_2\}, \{d_3, d_4\}, \{d_5, d_6\}, \{d_7, d_8\}, \{d_9, d_{10}\}, \{d_{11}, d_{12}\}, \{d_{13}\}\} \tag{17}$$

For each of these batches, predictions will be made by the model and true labels along with predictions will be used for calculating a binary cross-etropy loss E. The Binary

cross-entropy loss is widely used for training binary classifiers. Its definition is presented as Equation (18).

$$E(d,y) = -lr \sum_{i=1}^{n} d \cdot log_2(y) + (1-d) \cdot log_2(1-y) \tag{18}$$

Where $lr \in \mathbb{R}$ is a hyperparameter called the "learning rate". What distinguishes hyperparameter of a model from parameter, is that hyperparameter is not being trained. We have to tune a value of a hyperparameter ourselves. By tuning hyperparameter lr we can control how fast loss function converges to minimum. If a learning rate is too high, model can bypass minimum of loss function. If a learning rate is too low, model will train slowly and could stuck in a local minimum.

2.4. The idea of Convolutional Neural Network

The convolutional neural network is a type of a neural network model widely used for image recognition. The reason for that, is that convolutional neural network is capable of learning and recognising "similar patterns" on far different parts of an image. Before we explain convolutional neural network, we have to recall the idea of the convolution in mathematics.

Let us assume, that we have a relay-race with 2 members. The whole distance of the race is equal to $t \in \mathbb{R}$. The function $f\colon \mathbb{R} \mapsto \langle 0,1 \rangle$ is the probability of running some distance $x \in \mathbb{R}$ by the runner who starts the race. The covered distance x could, of course, be smaller than the whole distance t. If the first runner stops at some point x then the second runner must start from the same point x and has to finish the race by running distance $t-x$ with probability function given by $g\colon \mathbb{R} \mapsto \langle 0,1 \rangle$.

If we consider every possible position x, where the first runner stops, and calculate the probability of "winning" by reaching the finish line (i.e. covering the whole distance t), we will get the convolution of functions f with g as shown in Equation (19).

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x) \cdot g(t-x) dx \tag{19}$$

The discrete version of the convolution for discrete probability functions has following definition (20):

$$(f * g)(t) = \sum_{x} f(x) \cdot g(t-x) \tag{20}$$

In convolutional neural network, the role of function f is played by the input image, and the role of function g is played by the kernel (also known as the filter). Assume, that the input image is represented as a matrix $I \in \mathbb{R}^{n \times m}$, and the kernel is a matrix $K \in \mathbb{R}^{r \times k}$, where $n, m \in \mathbb{N}$, and $r \leq n \wedge k \leq m$. The kernel, starting from the top left corner moves through entire x-axis with some stride until it reaches the right corner. After reaching the

right corner, the kernel is placed at the beginning of x-axis, then slightly shifted down through y-axis with some stride, and the process repeats, until the kernel reaches the bottom right corner. During each shift, the convolution of the kernel with the covered part of the image is calculated, and the result is stored as the "output image" O.

An example of convolution with stride 1×1 is shown in Equations (21), (22) and (23):

$$I = \begin{bmatrix} 5 & 3 & 2 \\ 1 & 5 & 6 \\ 3 & 4 & 1 \end{bmatrix} \tag{21}$$

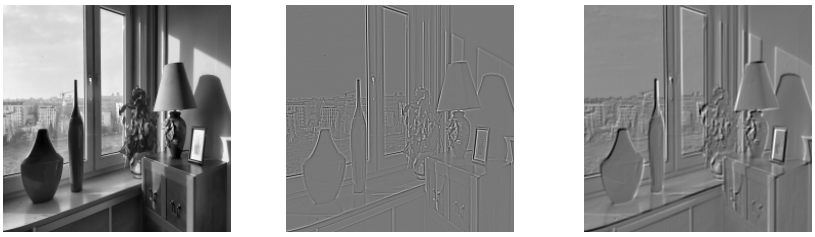
$$K = \begin{bmatrix} -2 & -1 \\ -1 & 1 \end{bmatrix} \tag{22}$$

$$O = \begin{bmatrix} (-2 \cdot 5 + -1 \cdot 3 + -1 \cdot 1 + 1 \cdot 5) & (-2 \cdot 3 + -1 \cdot 2 + -1 \cdot 5 + 1 \cdot 6) \\ (-2 \cdot 1 + -1 \cdot 5 + -1 \cdot 3 + 1 \cdot 4) & (-2 \cdot 5 + -1 \cdot 6 + -1 \cdot 4 + 1 \cdot 1) \end{bmatrix} \tag{23}$$

A layer in the convolutional neural network is represented by a sequence of kernels. Each kernel in the process of training learns to "highlight" different patterns on an image. For example, kernel K_1 (24) is capable of detecting edges and kernel K_2 (25) embosses image. The resulting convolutions are presented along with the original image on Figure 3.

$$K_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{24}$$

$$K_2 = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix} \tag{25}$$



(a) Original image (b) edge detection with kernel K_1 (c) emboss with kernel K_2

Fig. 3. The result of the convolution process applied to image (a) with kernel K_1 (b) and K_2 (c)

The process of training of a convolutional neural network is a described in the above. The loss function is minimised and, consequently, the weights of the kernels are modified.

If an image consists of $c \in \mathbb{N}$ color channels, then the kernel is represented as c matrices instead of a single one. Each of these kernels is applied accordingly to the color channels and the convolution is calculated. The resulting convolutions for each of the color channels are summed together into a single color channel of the resulting image.

Therefore, if the input image to the layer has *RGB* colors, the output image has as much color channels as the number of kernels.

2.5. Max Pooling

In order to reduce the memory consumption and the model complexity, *max pooling* operation is applied to the images. Max pooling splits its input image into small segments and substitutes these segments by the "representative pixels". The representative pixel is the maximum of the values of the pixels in a given segment. An example of max pooling is presented on Figure 4.

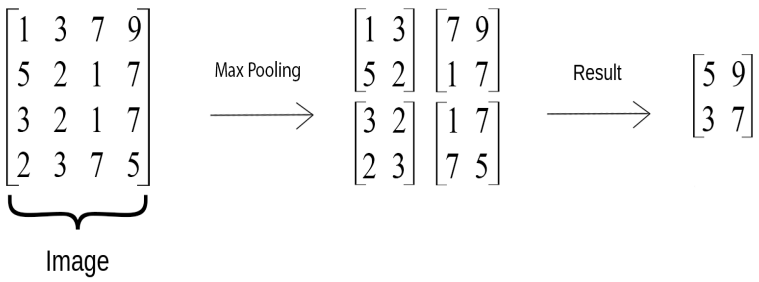


Fig. 4. An example of max pooling with segment shape 2×2

Max pooling also works with shapes of segments that cannot be fitted into the whole image. For example, if we split the original image $I \in \mathbb{R}^{5\times5}$ into segments with the highest dimension 2×2, then 4 segments will be created with dimension 2×2, 2 segments with dimension 2×1, 2 segments with dimension 1×2 and a single segment with dimension 1×1. Each of those segments will be substituted by its representative pixel. Further description of max pooling could be found in article [12].

2.6. Flatten

The flatten layer is used when a sub-neural network is stacked on top of a convolutional neural network to form a combined model. The sub-neural network takes a 1-dimensional vector $x \in \mathbb{R}^k$ as its input, however the convolutional network outputs a 3-dimensional matrix $O \in \mathbb{R}^{n\times m\times r}$. The flatten layer connects these two networks by flattening matrix O to vector x.

2.7. Common pitfalls

In this subsection we describe some common problems that one may encounter when selecting models for a given problem, or when training the models. We give some insights how to solve or avoid the potential problems.

Overfitting When the loss function E on the training set approaches zero, but on the validation and test sets is relatively high, a phenomenon called "model overfitting" appears. During the training process, the model starts to learn the noise that is present in the data and miss-interpret it as some important feature of the input. The overfitted model is a model with too high complexity for a given problem.

As an example let us consider a model that recognises different animals on images. Overfitted model could create association, that if some pixels on the image has some specific colors, then there is a polar bear on the image.

Underfitting An opposite phenomenon to the "model overfitting" can also occur. A model could be too simple to create some complicated associations and, consequently, learns overly generalised concepts. For example, an "underfitted model" can learn that if an image has a high white color balance, then we are dealing with polar bear on an image. This mistaken association happens, because images containing polar bears are often being made in snow environments, so the image mostly consists of white colors. Thereofre, an underfitted model can have problems in distinguishing a polar bear from a polar wolf.

Regularisation One way of dealing with overfitting is by using "regularisation" to reduce the complexity of the models. The complexity of a neural network grows with the increased number of weights and layers. Techniques used for the reduction of model complexity are called "regularisation techniques". In the below we will mention a couple of the most common regularisation techniques, especially focusing on these that we have used in our models.

Dimensionality Reduction. If we consider a vector $x \in \mathbb{R}^k$ as an input of a layer $A \in \mathbb{R}^{n \times k}$, the reduction of the dimension of vector x will also reduce the number of columns in matrix A. In this way, the complexity of a neural network can be reduced. It is also possible to reduce the number of neurons in matrix A. Consider a neural network M (26).

$$M(x) = h(B(g(Ax))) \tag{26}$$

If we reduce the number of neurons in layer A to some value $t \in \mathbb{N} : 0 < t < n$, the number of columns of matrix B will be accordingly reduced to t.

Regularisation l_1 and l_2. Often we do not know what is a good way of reducing the dimensionality of the input features and the number of neurons. To remove unnecessary complexity without modifying the structure of the model (i.e. without removing neurons or layers) we can substitute unnecessary weights with zeros. Unfortunately, we do not know which weights are important and which are not. The process of finding and removing unnecessary weights can be achieved by providing additional loss function to minimise the total size of the weights. More specifically, we can add a loss function whose aim is to reduce weights like on Equation (27), where $w \in \mathbb{R}^n : n \in \mathbb{N}$ is a sequence of the weights of the whole model and $\alpha \in \mathbb{R}$ is a hyperparameter.

$$l_1 = \alpha \sum_{i=1}^{n} |w_i| \tag{27}$$

Regularisation function l_1 is also commonly known as "Least Absolute Shrinkage and Selection Operator (LASSO)". Minimising several loss functions is called "multicriteria optimisation" and in our case is achieved by adding them together (28).

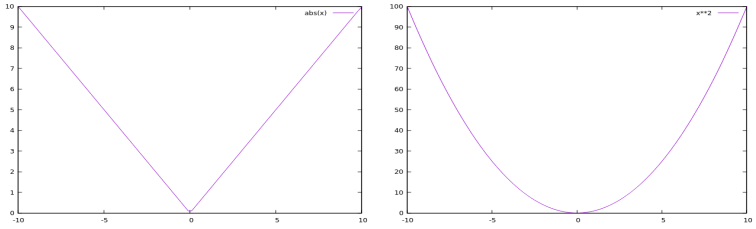
$$E' = E + l_1 \tag{28}$$

By tuning hyperparameter α, we can control the level of regularisation. If the level of regularisation is too high, all weights of the model could reach values close to zero. If a regularisation level is too low, the model can still overfit.

A similar regularisation function l_2, also known as "ridge" is defined by Equation (29).

$$l_2 = \alpha \sum_{i=1}^{n} w_i^2 \tag{29}$$

Comparing function l_1 and l_2, we can observe that l_1 function is more "aggressive". By looking at the shapes of these functions (see Figure 5), we can see that l_1 has the same speed of convergence, independently of the values of the weights. In contrast, l_2 function converges slower, when the values of the weights are small.



(a) l_1 regularisation (b) l_2 regularisation

Fig. 5. Functions applied by regularisers l_1 and l_2

Data Augmentation. If an overfitted model creates artificial associations, we can provide to the model additional slightly augmented images. The reason behind data augmentation is that collecting enough training samples can be often problematic or even impossible. The idea of data augmentation is as follows: instead of collecting new training samples, the samples are generated as a slight modification of the current training samples. Some of the possible modifications are:

- rotation
- shift by x axis
- shift by y axis
- vertical flip
- horizontal flip
- shear
- zoom

– change of the color map

Our method of image augmentation is based on rotations with filling segment gaps by segment surrounding and is described in details in Section 3.4.

Balance between Overfitting and Underfitting Sometimes reducing the complexity of a model can lead to underfitting. Dimensionality reduction may remove some important features from the data that cannot be easily noticed or seems to be unimportant only in a specific training samples. A neural network with some neurons and layers removed may not be able to learn complicated patterns in the data. Finally, after providing new samples with data augmentation it may be too difficult for a model to learn other associations than the primitive ones.

When a model starts to underfit after applying complexity reduction, we have to increase its complexity, by adding additional layers, neurons or reducing the level of regularisation. After increasing the complexity of the model, we can face again the problem of overfitting. Such a situation leads to a loop called "balance between overfitting and underfitting", which is explained in article [2]. There is no simple solution to this problem. During our research, we used the following technique. We started with building a simple model with only two layers — one hidden layer and the ouptut layer, and used this model as the "baseline model". This baseline model gave some results. Then we applied regularisation to it. After applying the regularisation the model gave worse results, so we significantly increased model complexity, by adding more layers and neurons. This model started to overfit quickly, so we applied a higher level of regularisation (i.e. we used $\alpha = 0.001$ for l_1 regulariser and added data augmentation). In the result, the model started to underfit, so we simply lowered the regularisation hyperparameters obtaining our final results.

3. Experimental Environment

We evaluate our models and compare against the state-of-the-art model on Mall Dataset from [1].

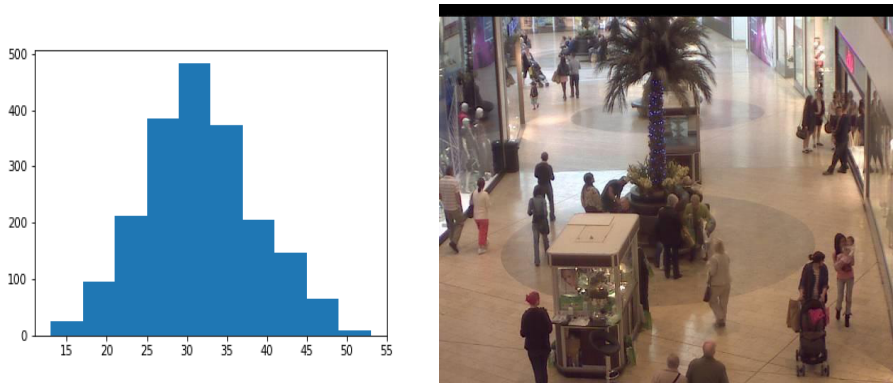
3.1. Dataset

The data from Mall Dataset consists of:

– 2000 images taken from the same camera with the same perspective and with resolution 640×480 pixels each;
– the coordinates of people's faces on images;
– the perspective map of the images (see Subsection 3.3).

The distribution of the number of people in the database is shown on Figure 6(a). The distribution clearly resembles the Gaussian distribution with mean 31.16 and standard deviation 6.94. Basic statistics for the whole dataset are as follows:

– maximum number of people on an image: 53;
– minimum number of people on an image: 13;



(a) Distribution of crowd sizes across samples (b) Sample image

Fig. 6. Distribution of crowd sizes along with sample image from Mall dataset

- average number of people on an image: 31.16;
- median number of people on an image: 31;
- standard deviation of the number of people on an image: 6.94.

A sample image from the dataset is shown on Figure 6(b).
The dataset was randomly split into:

- the training set consisting of 1500 images;
- the validation set consisting of 300 images;
- the test set consisting of 200 images.

3.2. Data Segmentation

Since we were interested in low memory cost of crowd counting, we could not work with full images. We decided to split each image into smaller segments of the same shapes and with each segment associate a binary value indicating whether or not a person is present on the segment. We accomplished this by creating a map that to the coordinate of the centre of a person's face assigns the segment that contains it. In this process we lost some information, but the accuracy of the final classifiers shows that the loss was not that big.

We examined several splitting methods: we split images into 100, 400 and 1600 non-overlapping rectangular segments of the same shape[3]. Figure 7 shows segments of an image after splitting it on a various number of segments. It is worth noticing, that a person's sweater is barely visible when we split the image on 1600 segments. We have found that splitting into 400 segments performs best for our models. One may argue that this splitting method has a reasonable ratio of segments with a person to segments without a person at around 0.0679, and at the same time it has a decently low loss rate. Table 1 summarises the statistics for other splitting methods.

[3] Notice, that we did not use content-based image segmentation.

Table 1. Statistics for different number of segments for image splitting

Segments	Ratio	Mean loss	Std loss
100	0.2105	−10.1035	3.7835
400	0.0679	−3.9995	2.1153
1600	0.0188	−1.0040	1.0144

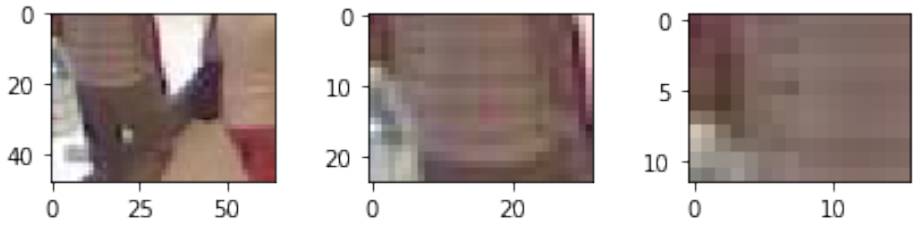


Fig. 7. Segments after splitting an original image on 100, 400 and 1600 fragments

3.3. Perspective Map

In Mall Dataset the camera that took the pictures was not positioned perpendicularly to the scene, therefore objects on the pictures may have significant distortion. Mall Dataset provides this information in the form of perspective map. The perspective map indicates the real size of each pixel of the images. Figure 8 shows a sample image from the dataset whose fragments are adjusted according to the perspective map.

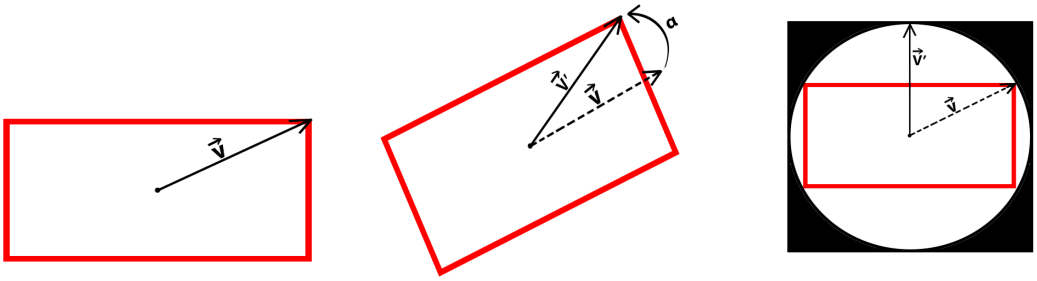
Fig. 8. An image with adjusted perspective using perspective map

3.4. Data Augmentation

Due to the fact that the number of segments with a person is much smaller than the number of segments without a person (the ratio is equal to 0.0679), the two classes are unbalanced and a classifier may favor the larger class. Therefore, some augmentation techniques had to be used to produce more balanced classes.

The segments were transformed using random rotations of images by an angle smaller or equal to $\frac{\pi}{12}$. The problem with the standard method of data augmentation is, that after a rotation the segment is cropped to the shape of (a non-rotated) rectangle, whereas new fragments which occur after the rotation are filled out with meaningless values. To prevent this and to create completely augmented segments, a larger surrounding fragment was rotated. Thanks to this approach we did not lose any data after rotation. Figure 9 shows how a rotation of a segment should be performed to not lose information from the surrounding image.

There are also segments for augmentation that lie on the edges of an image — they were augmented in the same way, except that areas of the square which were out of the image were filled with zeros.



(a) Vector v (b) Vector v after rotation by α (c) A new image segment

Fig. 9. A segment within the surrounding image

3.5. Hardware

We trained our models on an Intel Xeon workstation with 256GB RAM and equipped with four NVIDIA Tesla K80 units (a single K80 unit consists of two interconnected K40 units).

3.6. Error Metrics

In order to compare the models, we use two fundamental error metrics — Mean Square Error (MSE) (30), and Mean Absolute Error (MAE) (31) (see for example: [31]), which are calculated according to the following formulas:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \tag{30}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{31}$$

Where $\langle x_1, x_2, \ldots, x_n\rangle$ are the predicted values, whereas $\langle y_1, y_2, \ldots, y_n\rangle$ are the reference (i.e. true) values. In addition, we calculate the accuracy of a prediction x_i with Equation (32)

$$acc(x_i, y_i) = \begin{cases} 1 - \frac{|x_i - y_i|}{y_i} & \text{if } x \in \langle 0, 2y_i\rangle \\ 0 & \text{if } x \in (2y_i, +\infty) \end{cases} \tag{32}$$

and the accuracy of a model as the average over all its predictions.

4. Models

As mentioned earlier we focused on building possibly simple models that can be run on low computational power/memory devices, without sacrificing much on the accuracy. Two major limiting factors for our architectures were: a) we could not afford working with full images, therefore the images had to be split into smaller segments, b) in deep neural network architectures, we could not afford any variant of skipping connections (because they force the device to cache the data along the forward-pass of the network). We experimented with variety of architectures of neural networks to predict if a single segment of an image contains a person. For the results presented at the conference We trained 17 different models in total. Two best models are presented in Subsection 4.1 and Subsection 4.2 (see Figure 10). The first one operates on raw (segments) of images, whereas the second makes use of perspective map. Other models mostly differ in the values of regularisation coefficients and the number of segments each image was splitted[4]. To get the final prediction of the size of crowd, we used a cut-off point of each prediction, and then sum over all values of segments that constitute the image.

The model described in Subsection 4.3 enhances the above approach in the following way. Instead of the binary cut-off point, we use a cut-off point based on continuous logic: a value above the cut-off point is truncated to 1, whereas values below the cut-off point are scaled linearly. We trained 12 models in total according to this approach. The best model is described in Subsection 4.3 (see Figure 11). By using this approach we were able to greatly simplify the overall structure of the model and improve its accuracy.

4.1. Convolutional Model v1

The architecture of Convolutional model v1 is presented on Figure 10(a). The building blocks of the model are as follows: 2-dimensional convolutional layers [26] [16] with

[4] We also tested a split into 1600 segments, which gave much worse results

receptive field size of 3×3, stride of 1×1, $(2, 2)$-max pooling layers [12] [24], Batch normalization layers [15], Dropout layers [27] [3] with parameter 0.5 and dense (i.e. fully connected) layers. Next to the name of the layers, the numbers specify the size of the input and the size of the output respectively (width, height and depth). We used segmentation technique as described in Subsection 3.2 (each image was split into 400 segments) and data augmentation as described in Subsection 3.4 to balance the classes of positive and negative appearances.

Because we used a convolutional layer as the first layer in the network, the system had to store segments as full tensors rather than mere vectors. In the result the underlying filters had to be represented as full tensors as well and used in convolution process. Such situation resulted in high memory consumption, and we were forced switch in the training process from batches to mini-batches. The neurons from the hidden layers used Rectified Linear Unit as the activation function, whereas the output layer used the sigmoid function. We used a batch normalization behind each layer which allowed us to set higher values of learning rate at the beginning of neural network learning process, and also prevented Internal Covariate Shift process (the issue is discussed in [15]). The training process utilises two regularisation techniques: dropout connections as described in the above, and l2 regularisation. We tested several regularisation coefficients for the learning process, but found that the value of 10^{-7} performs best.

We trained the model with Adam optimiser initially setting the learning rate to 10^{-1} to speed up training process. High value of learning rate was acceptable due to Batch Normalization usage. We trained the model for about 75 epochs till accuracy for validation set started getting worse. Then we applied a learning rate value 10^{-3} and continued training for next 30 epochs. At that point we exceeded previous results. We decided to settle much lower learning rate - 10^{-5} and continue the process for next 20 epochs. At that point we reached the best weights. Further training caused overfitting.

The cut-off point has been chosen to 0.3. The overall accuracy of the model is presented in Table 3. We achieved Mean Absolute Error (MAE) of 3.44 and Mean Squared Error (MSE) of 18.87.

4.2. Convolutional Model v2

Convolutional model v2 (see Figure 10(b)) is an enhanced version of the model from the previous subsection. Together with a 3-channel image it additionally utilises the fourth channel with the perspective map of the image. Because the size of the input has significantly grown, a deeper model than the previous one performed better. All of the parameters of the architecture remained unchanged. Data augmentation had to be modified to incorporate the perspective map.

The training process of convolutional model v2 was similar to the training process of convolutional model v1. At the beginning we settled learning rate at value 10^{-3} and started training for 30 epochs. After that, we lowered learning rate to value 10^{-4} to move slowly to local minimum and continued for the next 20 epochs. Then we lowered learning rate to the value 10^{-5} and continued for the next 80 epochs. At that point we reached local minimum and further training had not give any better results.

We experimented with several l2 regularisation hyperparameters for this model as well. Value 10^{-6} performed best for this model.

The cut-off point has been chosen to 0.6. The overall accuracy of the model is presented in Table 2. We achieved Mean Absolute Error (MAE) of 3.35 and Mean Squared Error (MSE) of 18.33, which is slightly better than with the previous model.

4.3. Continuous convolutional model

Continuous convolutional model (see Figure 11) is a new model based on enhanced cut-off point approach. Just like Convolutional model v2, together with a 3-channel image it additionally utilises the fourth channel with the perspective map of the image. Because the model can use a continuous spectrum of information, we could simplify its architecture by reducing the convolutional blocks to a single layer and remove two dense layers.

The training process of continuous convolutional model was similar to the training process of convolutional model v1 and v2. We started with learning rate at value 10^{-4} and trained model for 5 epochs. After that, we lowered learning rate to value 10^{-5} and continued for the next 50 epochs. Then we again lowered learning rate to the value 10^{-6} and continued for the next 20 epochs. Then we switched back to 10^{-5} learning rate value and continued for the next 5 epochs. Then we settled learning rate at value 10^{-6} and continued training for the next 11 epochs. During last 4 epochs of the training process model was oscillating close to some local minimum, so we decided to finish the training process.

Several l2 regularisation hyperparameters were tested. Value 10^{-7} performed best for this model.

The continuous cut-off point has been chosen to 0.9. The overall accuracy of the model is presented in Table 2. We achieved Mean Absolute Error (MAE) of 2.46 and Mean Squared Error (MSE) of 9.6. These values are far better than the best values achieved by our previous models.

5. Conclusions and Future Work

The accuracy and the errors of each of our models are summarised in Table 2. Convolutional model v2 slightly outperforms Convolutional model v1 both in terms of accuracy, MSE and MAE at the cost of a slight increase of model complexity. Nonetheless, it is the continuous convolutional model that beats Convolutional model v1 and v2 in terms of accuracy, MSE, MAE and model simplicity. Table 3 compares our best model to seven

Table 2. Accuracy and error of our models

Model	Accuracy	MSE	MAE
Convolutional model v1	88.73	18.87	3.44
Convolutional model v2	88.90	18.33	3.35
Continuous convolutional model	92.00	9.61	2.46

most promising models found in literature: the state-of-the art solution by Walach and Wolf [29], the model based on cumulative attributes [10], the model of random projection

Table 3. Model comparison

Model	MSE	MAE
Convolutional model v1	18.87	3.44
Convolutional model v2	18.33	3.35
Continuous convolutional model	9.61	2.46
State-of-the-art [29]	NA	2.01
Cumulative Attributes [10]	17.7	3.43
Count forest [20]	10.0	2.50
Random projection forest [32]	15.5	3.22
Mixture of Counting CNNs [17]	13.4	2.75
Weighted VLAD [25]	13.05	2.86
Localized crowd counting [11]	15.7	3.15

forest [32], mixture of counting CNNs [17], weighted VLAD [25] and localized crowd counting [11].

As it turns out, all of our models show competitive results, despite their simplicity and low-computational-cost of their architecture. Our continuous convolutional model is second-best falling behind the state-of-the art solution only. Although our MAE of 2.46 is worse than MAE of 2.01 (the state-of-the art solution), this is negligible from the perspective of crowd-sizes of about 50 persons (i.e. the difference between the errors is less than 1%). Therefore, it may be successfully run on devices that have relatively small resources available. Moreover, we did not use any ensemble methods to tune up our models. By substituting the usual cut-off point by continuous cut-off point we were able to outperform not only our previous models: convolutional model v1 and v2, but also all of the other models evaluated on MALL dataset.

More optimisation techniques may be applied to our models to obtain even lighter architectures (e.g. reduce the size of the color channels, MobileNet architectures [13, 8]). We leave this for the future work. From the perspective of crowd management another interesting direction of research is to identify more details about crowd beside the mere number of the participants. The theory developed by Pierre Bourdieu in [6] tells us that the aspects like gender, race or social class of the individuals may play crucial role in crowd management. We also leave this for future work.

References

1. Mall dataset. http://personal.ie.cuhk.edu.hk/~cloy/downloads_mall_dataset.html (2014)
2. Van der Aalst, W.M., Rubin, V., Verbeek, H., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software & Systems Modeling 9(1), 87 (2010)
3. Ba, J., Frey, B.: Adaptive dropout for training deep neural networks. In: Advances in Neural Information Processing Systems. pp. 3084–3092 (2013)
4. Bonyadi, M.R., Michalewicz, Z., Przybylek, M.R., Wierzbicki, A.: Socially inspired algorithms for the travelling thief problem. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. pp. 421–428. ACM (2014)
5. Boominathan, L., Kruthiventi, S.S., Babu, R.V.: Crowdnet: A deep convolutional network for dense crowd counting. In: Proceedings of the 24th ACM international conference on Multimedia. pp. 640–644. ACM (2016)

6. Bourdieu, P.: Distinction: A social critique of the judgement of taste. Routledge (2013)
7. Brzeski, A., Grinholc, K., Nowodworski, K., Przybylek, A.: Evaluating performance and accuracy improvements for attention-ocr. In: 18th International Conference on Computer Information Systems and Industrial Management Applications (2019)
8. Brzeski, A., Grinholc, K., Nowodworski, K., Przybylek, A.: Residual mobilenets. In: Workshop on Modern Approaches in Data Engineering and Information System Design at ADBIS (2019)
9. Change Loy, C., Gong, S., Xiang, T.: From semi-supervised to transfer counting of crowds. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2256–2263 (2013)
10. Chen, K., Gong, S., Xiang, T., Change Loy, C.: Cumulative attribute space for age and crowd density estimation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2467–2474 (2013)
11. Chen, K., Loy, C.C., Gong, S., Xiang, T.: Feature mining for localised crowd counting. In: BMVC. vol. 1, p. 3 (2012)
12. Graham, B.: Fractional max-pooling. arXiv preprint arXiv:1412.6071 (2014)
13. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
14. Idrees, H., Saleemi, I., Seibert, C., Shah, M.: Multi-source multi-scale counting in extremely dense crowd images. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2547–2554 (2013)
15. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
16. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
17. Kumagai, S., Hotta, K., Kurita, T.: Mixture of counting cnns: Adaptive integration of cnns specialized to specific appearance for crowd counting. arXiv preprint arXiv:1703.09393 (2017)
18. Leibe, B., Seemann, E., Schiele, B.: Pedestrian detection in crowded scenes. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). vol. 1, pp. 878–885. IEEE (2005)
19. Loy, C.C., Chen, K., Gong, S., Xiang, T.: Crowd counting and profiling: Methodology and evaluation. In: Modeling, simulation and visual analysis of crowds, pp. 347–382. Springer (2013)
20. Pham, V.Q., Kozakaya, T., Yamaguchi, O., Okada, R.: Count forest: Co-voting uncertain number of targets using random forest for crowd density estimation. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 3253–3261 (2015)
21. Przybylek, K., Shkroba, I.: Crowd counting á la bourdieu. In: European Conference on Advances in Databases and Information Systems. pp. 295–305. Springer (2019)
22. Przybylek, M.R., Wierzbicki, A., Michalewicz, Z.: Decomposition algorithms for a multi-hard problem. Evolutionary computation 26(3), 507–533 (2018)
23. Przybylek, M.R., Wierzbicki, A., Michalewicz, Z.: Multi-hard problems in uncertain environment. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 381–388. ACM (2016)
24. Scherer, D., Müller, A., Behnke, S.: Evaluation of pooling operations in convolutional architectures for object recognition. In: International conference on artificial neural networks. pp. 92–101. Springer (2010)
25. Sheng, B., Shen, C., Lin, G., Li, J., Yang, W., Sun, C.: Crowd counting via weighted vlad on a dense attribute feature map. IEEE Transactions on Circuits and Systems for Video Technology 28(8), 1788–1797 (2016)
26. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
27. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929–1958 (2014)

28. Stewart, R., Andriluka, M., Ng, A.Y.: End-to-end people detection in crowded scenes. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2325–2333 (2016)
29. Walach, E., Wolf, L.: Learning to count with cnn boosting. In: European Conference on Computer Vision. pp. 660–676. Springer (2016)
30. Wang, M., Li, W., Wang, X.: Transferring a generic pedestrian detector towards specific scenes. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition. pp. 3274–3281. IEEE (2012)
31. Willmott, C.J., Ackleson, S.G., Davis, R.E., Feddema, J.J., Klink, K.M., Legates, D.R., O'donnell, J., Rowe, C.M.: Statistics for the evaluation and comparison of models. Journal of Geophysical Research: Oceans 90(C5), 8995–9005 (1985)
32. Xu, B., Qiu, G.: Crowd density estimation based on rich features and random projection forest. In: 2016 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 1–8. IEEE (2016)

Karolina Przybyłek obtained her master's degree in social sciences (with honors) from University of Warsaw in 2018. Since then she has been dealing with scientific and social activities: she co-organized and participated in international scientific conferences. She is currently preparing her PhD dissertation on the decision-making process of judges. Her research interests include: theories of education, the sociology of Pierre Bourdieu, sociology of art, criminal law, criminal trial, issues connected with the status of courts and judges.

Illia Shkroba is a teaching assistant at Polish-Japanese Academy of Information Technology in Warsaw. He obtained his master's degree in Information Technology with Data Science specialization (with honors) from Polish-Japanese Academy of Information Technology in 2020. Since 2019 he is actively participating in government research projects. His research interests include: deep neural networks, data mining, process mining.

Received: January 15, 2020; Accepted: September 1, 2020.

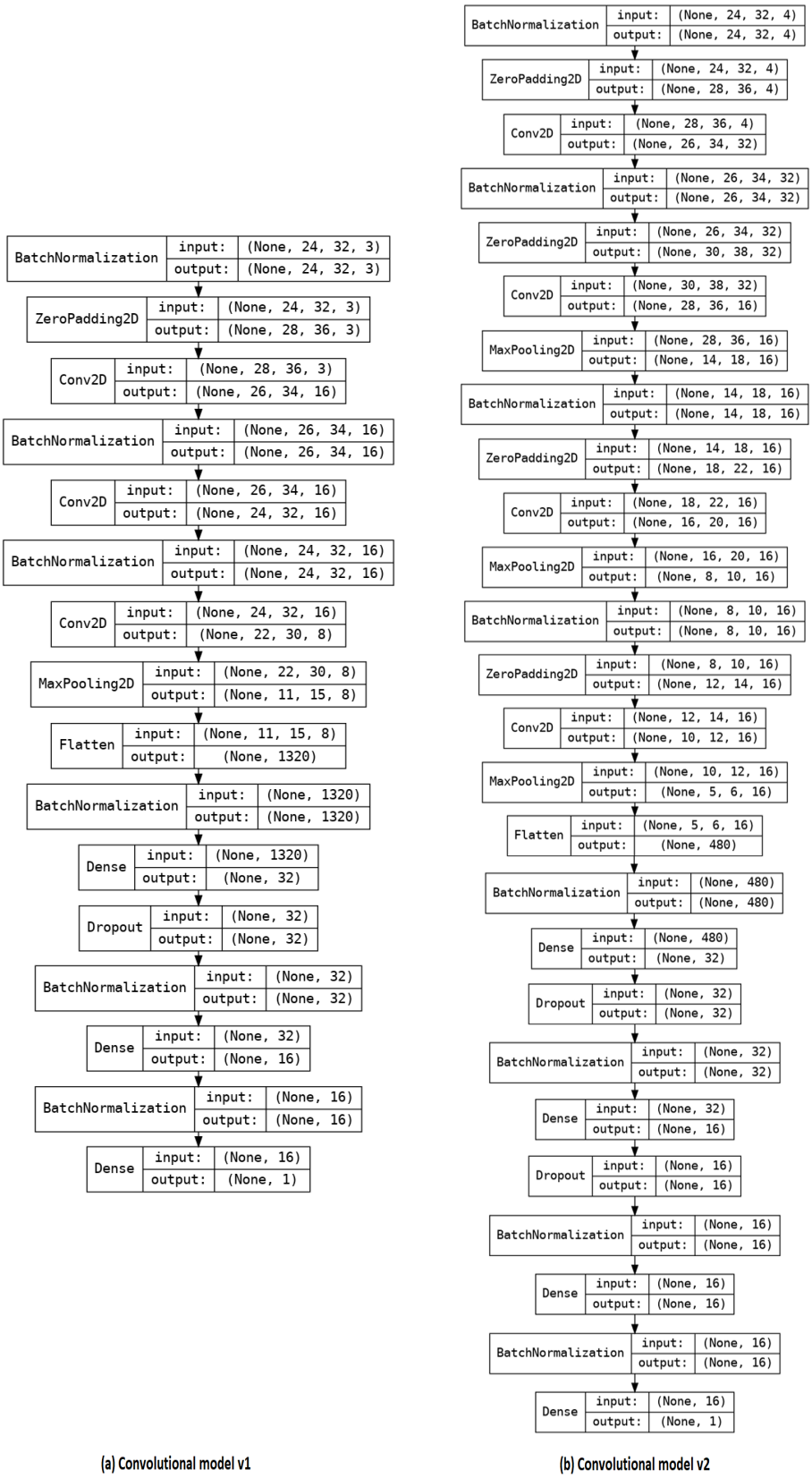


Fig. 10. CNN models

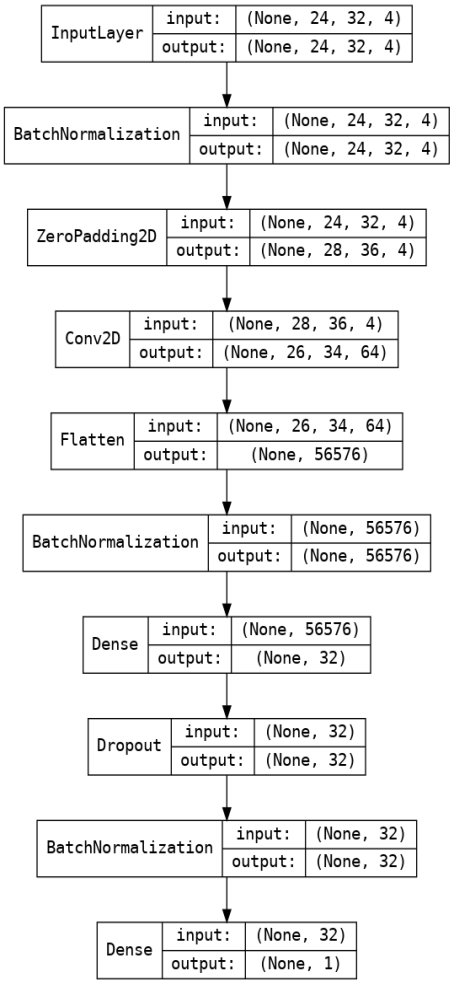


Fig. 11. Continuous convolutional model