BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

COUNTING VEHICLES IN IMAGE AND VIDEO

POČÍTÁNÍ VOZIDEL V OBRAZE A VIDEU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **DOMINIK GABZDYL**
AUTOR PRÁCE

SUPERVISOR **Ing. JAKUB ŠPAŇHEL**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Bachelor's Thesis Specification

Student: **Gabzdyl Dominik**
Programme: Information Technology
Title: **Counting Vehicles in Image and Video**
Category: Image Processing

Assignment:

1. Learn how to count vehicles in an image and video.
2. Collect the appropriate experiment data set.
3. Find literature on the use of convolutional neural networks for counting vehicles in the image and focus on the "Counting by Regression" principle
4. Implement them with available toolkits and experiment with methods that are appropriate for vehicle counting problem.
5. Evaluate the selected methods properly and discuss the results achieved.
6. Create a poster and video presenting this work, its goals and results.

Recommended literature:

- Onoro-Rubio et al. "Towards perspective-free object counting with deep learning." *ECCV,* 2016.
- Goldman et al. "Precise Detection in Densely Packed Scenes." *CVPR,* 2019.
- Further as directed by the supervisor

Requirements for the first semester:

- Completed first three items of the assignment
- Fourth item in progress
- Parts of the final report

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Špaňhel Jakub, Ing.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: May 28, 2020
Approval date: November 1, 2019

Abstract

Traffic analysis is still a challenging task. During such task there are many pitfalls to be aware of. Such as small image resolution, high number of overlapping objects, angle of camera, blurred objects due to their motion or weather conditions. This thesis addresses these issues by using the convolutional neural network approach. In this thesis I propose a new architecture which adheres to Counting by Regression principle. The proposed architecture is inspired by some state-of-the-art architectures and improves accuracy on various datasets. For instance on the very small PUCPR+ dataset the Root Mean Square Error between expected and predicted vehicle counts was reduced from 34.46 to 6.99 vehicles (measured on the test set). Results achieved showed that there is still space for improvements and a possible further research in Counting by Regression principle.

Abstrakt

Analýza silničního provozu je stále náročnou úlohou. V průběhu této úlohy se vyskytují mnohá úskalí, která je třeba brát na vědomí. Například malé rozlišení obrazu, vysoký počet překrývajících se objektů, úhel kamery, rozmazání objektů v důsledku jejich pohybu nebo povětrnostní podmínky. Tato práce adresuje tato úskalí použitím konvolučních neuronových sítí. V této práci představuji novou architektu založenou na principu počítání regresí (Counting by Regression). Navržená architektura je inspirována některými state-of-the-art architekturami a vylepšuje přesnost na různých datasetech. Například na velmi malém PUCPR+ datasetu byla odmocnina ze střední kvadratické chyby (RMSE) snížena z 34.46 na 6.99 vozidel (měřeno na test setu). Dosažené výsledky ukázaly, že je zde stále prostor ke zlepšení a možný další výzkum v oblasti počítání regresí.

Keywords

vehicle counting, counting by regression, convolutional neural networks

Klíčová slova

počítání vozidel, počítání regresí, konvoluční neuronové sítě

Reference

GABZDYL, Dominik. *Counting Vehicles in Image and Video.* Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jakub Špaňhel

Rozšířený abstrakt

Úvod

Před vzestupem umělé inteligence se detekce vozidel realizovala hardwarovými řešeními. Jedním z příkladů této technologie je detekční smyčka. Vozidlo projíždějící takovou smyčkou způsobuje změnu v magnetickém poli a takto signalizuje svoji přítomnost. Avšak taková hardwarová řešení jsou nákladná. Aktuální vzestup na poli počítačového vidění představil přístupy s vysokou přesností a nízkými implementačními náklady.

Korektní detekce objektů v obraze je silně ovlivněna mnohými faktory. Takové faktory zahrnují extrémně překrývající se objekty, zkreslený obraz, nepříznivé povětrnostní podmínky, vysoká hustota objektů, atd. Analýza dat získaná z automatizovaných detekcí se může použít k vylepšení řízení dopravy. Frekventované dálnice nebo křižovatky se mohou sledovat pro svůj vývoj při dopravních zácpách. Tyto informace by mohly být užitečné pro veřejné orgány zodpovědné za údržbu a plánování silničních infrastruktur.

V této práci řeším obtížný problém přesného počítání objektů v obraze (nebo ve videu). Tato práce se soustřeďuje na použití konvolučních neuronových sítí (dále jen CNN) k detekci vozidel v obraze. Pro počítání objektů v CNN existuje mnoho principů. Jedním z principů je počítání regresí (Counting by Regression), což je i zaměřením této práce, v níž experimentuji se současnými state-of-the-art architekturami na různých datasetech. Inspirován těmito architekturami jsem navrhl novou CNN architekturu pro vylepšení současné state-of-the-art přesnosti na těchto datasetech.

Popis řešení

Počítání regresí (Counting by Regression) patří do kategorie učení s učitelem (supervised learning). Při tomto učení musí být poskytnutý očekávaný výstup (dále jen "ground truth"). Jednou z technik tohoto učení je použití teplotních map (dále jen "heatmapy"). Při této technice se očekává poskytnutí modelu dvojice vstupního obrázku a jeho očekávaného výstupu. Trénování modelu je poté v podstatě učení se mapování mezi vstupními obrázky a jejich ground truths.

Následováním idey Lempitského a kolektivu [22] se stane tato problematika počítání objektů procesem obnovování funkce hustoty, jakožto reálné funkce pixelů vstupních obrázků. Výstupem sítě tohoto přístupu je/jsou dlaždice (patch/patches) s heatmapou či heatmapami s predikovanými objekty. Ground truth dlaždice se vytváří pomocí gaussovského rozostření (Gaussian blur) s rozumnou hodnotou standardní odchylky σ (v této práci $\sigma = 15$) centrovaného na každou tečku v anotovaném obrázku.

Avšak tímto vytvoříme úplný ground truth obrázek. Vstupní obrázky spolu s jejich ground truths je zapotřebí rozdělit na dlaždice. Je potřeba si povšimnout, že každá dvojice vstupního obrázku a ground truth musí mít společné souřadnice (souřadnice dlaždic si musí vzájemně odpovídat).

Gaussovské rozostření používá kernel, jehož hodnoty v součtu dávají 1. Toto zajišťuje, že z obrázku po vykonání operace není žádná energie ani přidána ani odebrána. Tím pádem, je součet pixelů ve výsledném ground truth obrázku nezměněn (počet vozidel je zachován). K získání finálního počtu vozidel z heatmapy stačí jednoduše sečíst všehcny hodnoty pixelů v heatmapě.

Navržená architektura Navržené řešení využívá VGG-16 [40] front-endu pro extrakci nízko-úrovňových rysů obrazu. Context-Aware Network [23] a Pyramid Density-aware Attention Net for Accurate Crowd Counting (dále jen "PDANet") [2] používají jen prvních deset vrstev z VGG-16, stejně jako navržené řešení. Front-end je poté připojen ke stromově vypadající struktuře, která je inspirována Multi-column Network [4] architekturou.

Jedna vrstva (struktura) celé této stromové struktury je složena následovně. Vstupu do takové vrstvy je n-krát přiřazena váha (kde n je počet paralelních konvolučních vrstev v odpovídající vrstvě). Každá váha w_i představuje jeden neuron, jehož hodnota je určena během trénování modelu. Tyto váhy používají Softmax aktivační funkce. Na začátku je hodnota každé váhy w_i nastavena na zvyšující se násobek $\frac{1}{n}$ (n je počet paralelních konvolučních vrstev v odpovídající struktuře). Tedy například poslední vrstva má počáteční hodnoty $w_7 = \frac{1}{6}$, $w_8 = \frac{2}{6}$, $w_9 = \frac{3}{6}$, $w_{10} = \frac{4}{6}$, atd.

Každý neuron představující tuto váhu je připojen k dilatační konvoluci. Všechny dilatační konvoluce mají stejnou dimenzi jádra (kernelu) a to 3×3, ale liší se hodnotou dilatace a hloubkou.

Dilatační konvoluce v takové vrstvě jsou poté zprůměrovány. Zavěrečná 1×1 konvoluce zakončuje jednu takovou strukturu. Všechny konvoluce jsou v každé takové struktuře následovány ReLU aktivačními funkcemi.

Poslední struktura je zakončena dodatečnou 1×1 konvolucí a produkuje heatmapy pro vstupní dlaždice obrázku.

Nápad s váhovými neurony pochází od PDANet klasifikátoru, který je zodpovědný za unikátní váhy pro obě větve (dense a non-dense branches) v PDANet. Dilatační konvoluce zvyšují receptivní pole sítě a pomáhají jí s pochopením celkového obrazu namísto jemných detailů [27].

Navržená architektura je implementována v Keras frameworku [20]. Vytvořený model používá střední kvadratickou chybu (MSE) a optimizér Adam k výpočtu nových vah.

Nastavení trénování a predikce Extrakce vstupních RGB dlaždic probíhá rozdělením obrázku na čtyři ekvivalentní oblasti (kvadranty) a náhodným výběrem jedné z nich. Tato dlaždice získá náhodně vygenerovanou kombinaci augmentace (přiblížení, horizontální obrácení, změna gamma, přidání gaussového šumu). Front-end navržené sítě obsahuje dvě max-pooling vrstvy, což má za následek osminásobnou redukci dimenzí pro vstupní dlaždici. Ground truth obrázek musí být patřičně zmenšen, aby reflektoval tuto změnu. Prosté natahování nebo zmenšování dlaždice by potencionálně mohlo změnit počet (proto je potřeba dlaždici normalizovat). V článku *Towards perspective-free object counting with deep learning* [34] je pro tento případ navržená rovnice (rovnice č.1).

$$\hat{D}^{P+}_{pred} = \frac{\sum_{\forall p} {D^P}_{pred}(p)}{\sum_{\forall p} \hat{D}^{P}_{pred}(p)} \cdot \hat{D}^{P}_{pred}\,, \tag{1}$$

kde:

- $\hat{D}^{P+}_{pred}$: je normalizovaná heatmapa s s odpovídajícím počtem
- ${D^P}_{pred}$: je originální (ne přeškálovaná) heatmapa
- $\hat{D}^{P}_{pred}$: je přeškálovaná heatmapa *bez* odpovídajícího počtu
- p: je pixel

Model podporuje dvě metody pro vytvoření predikce. První z nich je kvadrantová extrakce dlaždic. Vstupní obrázek je rozdělen do čtyř ekvivalentně velkých nepřekrývajících se dlaždic (kvadrantů). Výsledná heatmapa se vytvoří zpětným poskládáním těchto čtyř dlaždic dohromady.

Druhá metoda (upřednostněna v této práci) je metoda husté extrakce dlaždic (dense patch extraction). V tomto případě má každá extrahovaná dlaždice velikost $\frac{1}{4}$ vstupního obrázku. V této práci se používá 10pixelový krok (horizontální i vertikální) k extrakci dlaždic z obrázku. Výsledná heatmapa musí být normalizována z důvodu překryvu extrahovaných dlaždic. Každá pozice (pixel) ve výsledné heatmapě je průměrem dlaždic, které se podíleli na predikci na dané pozici.

Zhodnocení výsledků

V průběhu implementace byla přesnost modelu upřednostněna před jeho rychlostí. Z tohoto důvodu byla upřednostněna hustá extrakce dlaždic před rychlejší kvadrantovou extrakcí. Model využívá techniky přenosu učení (transfer learning), který znatelně zredukoval trénovací dobu modelu (kupříkladu, trénování na TRANCOS datasetu trvalo jen 1.17 hodin). Stažené předtrénované váhy jsou součástí VGG-16 front-endu v modelu. Front-end je napojen na multi-column struktury s dilatačními konvolucemi. Tato sestava experimentálně zlepšila současné state-of-the-art výsledky na testovaných datasetech.

Navržený model dále vylepšuje state-of-the-art výsledky na všech testovaných datasetech (TRANCOS, PUCPR+, CARPK). Například na velmi malém PUCPR+ datasetu byla odmocnina ze střední kvadratické chyby (RMSE) snížena z 34.46 na 6.99 vozidel (měřeno na test setu). Podle mého názoru má na tom transfer learning nemalou zásluhu. Všechny naměřené hodnoty jsou výsledkem trénování s limitem 100 epoch.

Přesnost by mohla být dále vylepšená, pokud by byl nastavený větší limit pro epochy během trénování. Hustá extrakce dlaždic je přesnější než kvadrantová extrakce. Avšak, hustá extrakce má vyšší nároky na paměť i na čas v porovnání s kvadrantovou extrakcí. Tyto obtíže se dají redukovat nalezením optimální hodnoty pixelového kroku nebo nalezením nějaké pokročilejší metody. Zmíněná použitá extrakce se uplatňuje i při detekcích ve videu. Zde video představuje snímkovou sekvenci zakódovanou do video souboru. Real-time detekce se v této práci neuvažuje a může tak být předmětem pro další výzkum. Dosažené výsledky ukázaly, že je zde stále prostor ke zlepšení a možný další výzkum v oblasti počítání regresí.

Counting Vehicles in Image and Video

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Jakub Špaňhel. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Dominik Gabzdyl
May 28, 2020

Acknowledgements

I would like to thank my supervisor Ing. Jakub Špaňhel for his patient guidance, encouragement and advice he has provided throughout writing this thesis.

Contents

Chapter 1

Introduction

Before the rise of Artificial Intelligence, vehicle detection task was carried out via hardware solutions. Inductive loop solution, for instance, is one of them. Vehicle passing through such a loop causes a change in magnetic field, thus signalizing its presence. However, such hardware solutions are costly. Recent major rise in computer vision presented approaches with high accuracy and low implementation cost.

The correct object detection in image is severely impacted by many factors. Some of which include extremely overlapping objects, distorted image, bad weather conditions, high density of objects, etc. Data analysis acquired from automated object detection can improve management of traffic. Busy highways or road intersections can be monitored for traffic congestions to see how they evolve. Public authorities in charge of the maintenance and planning of road infrastructures can make use of such information.

This thesis addresses the difficult problem of precisely counting object instances within an image (or a video). This thesis focuses on the convolutional neural network (CNN for short) approach to perform vehicle detection in image. There are many principles to count objects instances in CNNs. One of them is Counting by Regression principle which is also the aim of this thesis. A new CNN architecture inspired by current state-of-the-art architectures is proposed. The goal of this new architecture is to further improve current state-of-the-art accuracy on various datasets.

In Chapter 2 some traditional approaches to count vehicles are introduced. The examples here are described only briefly as they are not the aim of the thesis. Chapter 3 is split into four sections. Firstly, an introduction to a related theory of CNNs is provided. Secondly, vehicle counting task in the context of CNNs is described. Counting by Regression principle is described here, along with other principles. Thirdly, related work architectures which inspired my architecture are discussed. Lastly, the proposed architecture is described. Chapter 4 provides implementation details of the proposed architecture. In addition to this, existing vehicle datasets are also introduced here along with some of their sample images. Chapter 5 contains an experimental evaluation of the model. It specifies how the accuracy is evaluated and what metrics are used. Evaluation configuration data (like GPU used, training times, learning rate,...) are also mentioned here. This chapter also contains the evaluation of the datasets mentioned in Chapter 4.

Chapter 2

Traditional Approaches to Count Vehicles

In this chapter I will go through some traditional approaches to count vehicles. This means only approaches that do not make use of Convolutional Neural Networks are described here. These traditional approaches will not be described in detail since it is not the aim of this thesis. The *Traffic Detector Handbook* [21] is the only source used here to introduce some traditional methods. The sections below have references to this book in the format (Chap *M*, p. *N*), where *M* is the chapter and *N* is the page number within the book.

Traditional approaches heavily rely on mechanical and hardware elements. Acquisition of electrical circuitry, maintenance of the mechanical parts or possible changes in road infrastructure render these solutions relatively costly. Recent major rise in computer vision presented approaches with high accuracy and low implementation cost.

2.1 Inductive-loop Detector

An example of in-roadway sensor (placed in a sawcut slot in the pavement) is inductive-loop detector. The inductive-loop system is essentially a tuned electrical circuit where loop wire and lead-in cable are the inductive elements. A conductive material (a vehicle) is induced by electrical current. The overall inductance of the loop is changed (decreased) and the controller interprets this as a presence of a vehicle [21] (Chap 2, p. 1). Figure 2.1 shows the architecture of this approach. It should be noted that the detector efficiency is degraded by pavement deterioration or weather conditions [21] (Chap 1, p. 5).

2.2 Magnetic Sensors

Magnetic Sensors are of passive type (they do not emit energy on their own) [21] (Chap 1, p. 13). The idea is based on a magnetic anomaly caused by ferrous objects. In this anomaly a ferrous object causes perturbation in the Earth's magnetic field. One way to implement the sensor is to use a search coil magnetometer (single coil winding on a permeable magnetic material rod core) [21] (Chap 1, p. 14). A moving vehicle causes a perturbation in the Earth's magnetic field and as a result a generated voltage in the coil is detected. Figure 2.2 shows magnetic flux lines in case of both present vehicle and both absent vehicle.

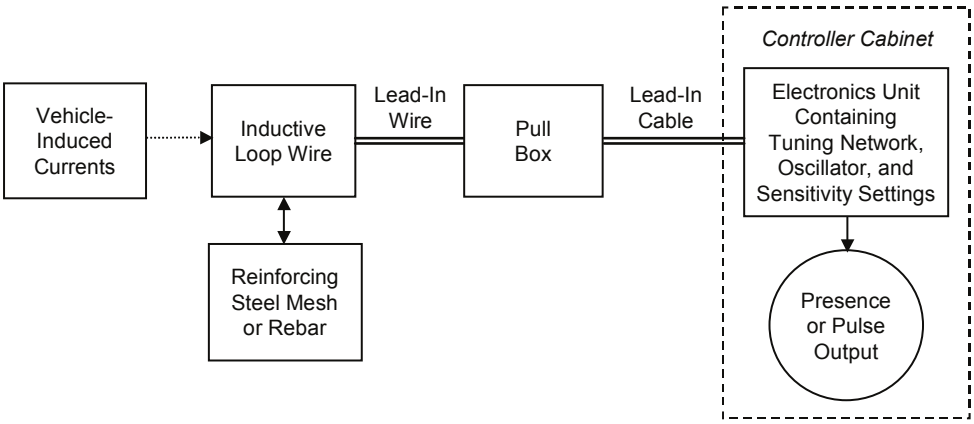


Figure 2.1: Inductive-loop detector system overview [21]. A vehicle within the loop causes eddy currents and decreases the loop inductance. A pulse is then sent to the Controller Cabinet to signify the passage or presence of a vehicle.

2.3 Microwave Radar Sensors

RADAR (originally acronym for RAdio Detection And Ranging) is generally defined as a device which transmits electromagnetic signals and receives echoes back from objects within its volume coverage [21] (Chap 1, p. 15). The word 'Microwave' refers to the wavelength of the transmitted signal. This wavelength is between 1 and 30 centimeters, which corresponds to a frequency range between 1 GHz and 30 GHz. Vehicle passing through the antenna beam reflects some of the transmitted energy back towards the antenna. A receiver then computes traffic data from the received signal (like speed, length of the vehicle, etc.) [21] (Chap 1, p. 16). Figure 2.3 shows the principle of microwave radar operation.

2.4 Laser Radar Sensors

These infrared sensors are of active type (they transmit energy on their own) [21] (Chap 1, p. 18). The transmitted energy is in the near infrared spectrum. Laser diodes emit a fixed number of beams that cover a desired area (for example lane width). A portion of this emitted energy is either reflected back towards the sensor or scattered by vehicles. The energy is captured by an infrared sensitive material and converted into electrical signals. Signal-processing techniques are used to further analyze these signals for the presence of vehicles. Figure 2.4 shows the geometry of a laser radar with two beams.

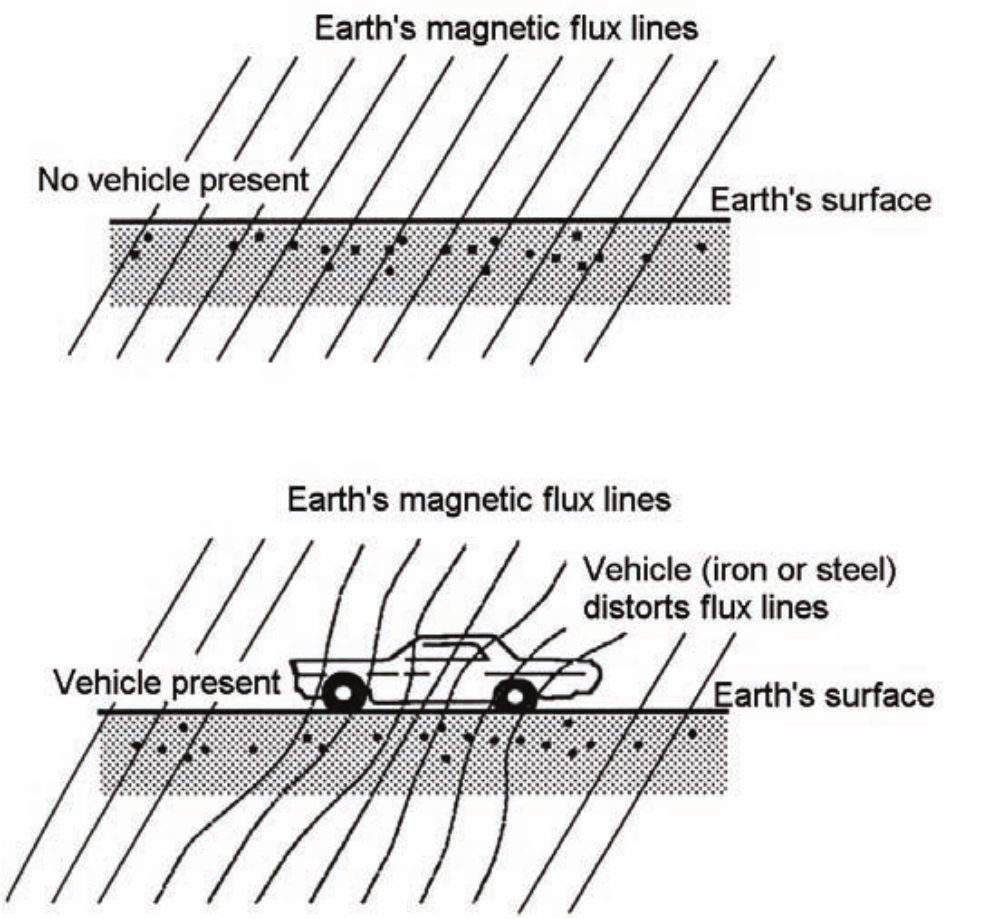


Figure 2.2: Magnetic anomaly example [21]. **Top** Earth's magnetic flux lines with no vehicle present. **Bottom** Distorted Earth's magnetic flux lines by a ferrous object (a vehicle).

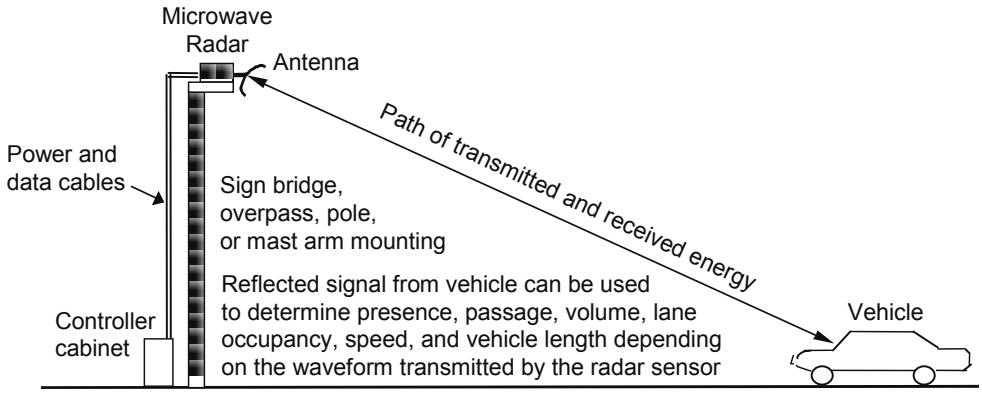


Figure 2.3: Microwave radar operation principle [21]. Radar transmits electromagnetic signals within its volume coverage. Passing vehicle reflects some of the transmitted energy back towards the antenna. A receiver further analyses this reflected energy.

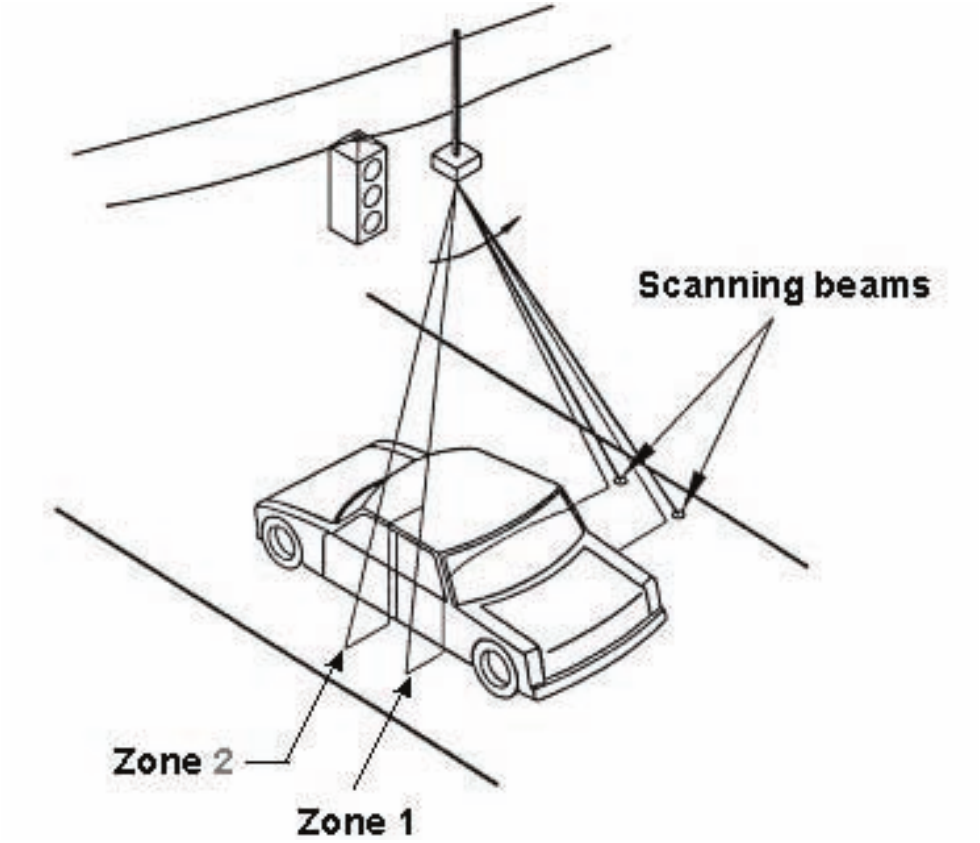


Figure 2.4: Laser radar beam geometry (figure taken from [21], source: drawing courtesy of Schwartz Electro-Optics, now OSI Laserscan, Orlando, FL). Two beams are separated by several degrees. Two beams allow for better collection of the scattered energy and also allow for speed computation since the beams are known distance apart from each other.

Chapter 3

Convolutional Neural Network Approach to Count Vehicles

A Convolutional Neural Network (CNN for short) is a subset of deep learning category [5]. It is an alternative solution to existing algorithms for feature detection. CNN is used in imagery, where classic algorithms are not suitable for detecting complex objects (like animals, vehicles, crowds,...). Figure 3.1 depicts an example of such CNN architecture.

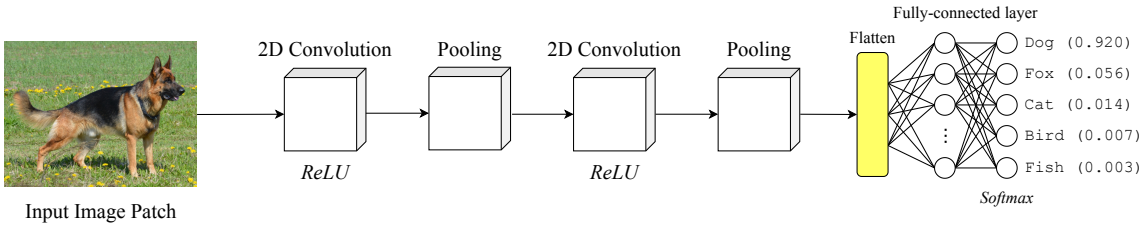


Figure 3.1: An example of a Convolutional Neural Network architecture. The image is taken from [35].

3.1 Convolutional Neural Network Layers

The most important part of a CNN is a convolution layer (Subsection 3.1.1) which is the basic building block. Along with this layer, there are several other layers which are in charge of:

- introducing non-linearities (Subsection 3.1.2)
- reducing computational overhead (Subsection 3.1.3)
- preventing overfitting (Subsection 3.1.3)
- category classification (Subsection 3.1.4)

3.1.1 Convolution Layer

Digital image is represented by a matrix with dimensions (height, width, channels). Image can be either greyscale (only pixel intensities) or coloured (red, green and blue channels).

This thesis takes into consideration only colour images, thus $channels = 3$. Coloured images provide more information about scenery and are less susceptible to inclement weather conditions or to shadow artefacts in comparison to their greyscale counterparts.

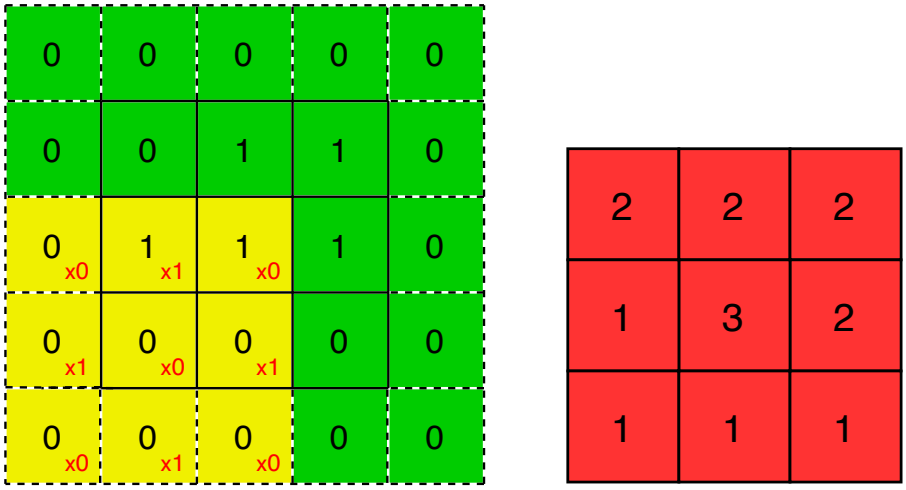
Convolution is a mathematical operation which combines two signals to produce a third one. In this context the first signal is an input image and the second one is a 2D filter (also known as a kernel). Equation 3.1 describes the mathematical operation of these two signals.

$$y[i,j] = \sum_{m=0}^{M_a-1} \sum_{n=0}^{N_a-1} h[m,n] * x[i-m, j-n]\,, \tag{3.1}$$

where:

- x: is the input 2D matrix (one channel of the RGB image)
- h: is the kernel (filter)
- M_a, N_a: are the kernel dimensions
- M_b, N_b: are the input 2D matrix dimensions
- i: $0 \leq i < M_a + M_b - 1$
- j: $0 \leq j < N_a + N_b - 1$

Assuming 0 is the first pixel, then initially the kernel is placed at the top left corner of the input image. Convolving in this initial setup is sliding the window (kernel) over the image from left to right from top to bottom direction. New pixel is computed by multiplying kernel values with input image values (within the window) and summing them up. Visualisation of this operation is shown in Figure 3.2.



(a) 2D convolving (b) Resulting feature map

Figure 3.2: 2D convolution process with zero-padding.

Notice, resulting signal is smaller than its input. This should be kept in mind when designing a CNN architecture. One way to change the output dimension of convolution

is to use zero-padding technique. In this technique a column or a row of zeros is either prepended or appended to the input image (could be repeated many times). Equation 3.2 describes the dimensions after convolution.

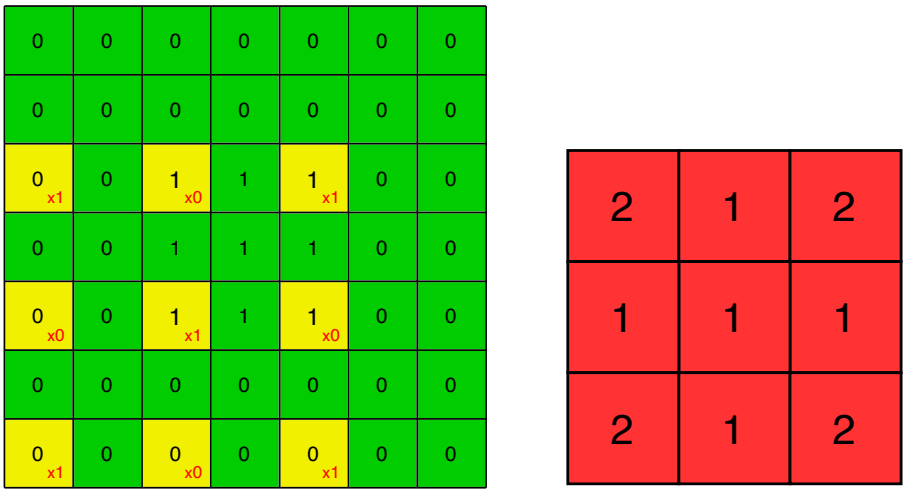
$$c_d = \frac{x_M - h_M + z}{s} \times \frac{x_N - h_N + z}{s} \times n_c \,, \tag{3.2}$$

where:

- c_d: are the resulting image dimensions after convolution
- x_M: is the input image height
- x_N: is the input image width
- h_M: is the kernel height
- h_N: is the kernel width
- z: is the number of additional rows/columns of zero-padding
- s: is the kernel stride
- n_c: is the number of channels

Some of the kernel's values have been revealed for common image manipulations, like image sharpening, blurring, edge detection. But these are just generic filters as we may want to detect more complex and sophisticated features of images such as precise object localization. Training CNN adjusts kernel values (weights), so that complex object detection (vehicles) is possible.

The proposed model uses along with normal convolutions dilated convolutions (also known as atrous convolutions). Dilated convolutions increase the receptive field of the network and help to understand the overall picture instead of finer details [27]. Figure 3.3 shows a visualization of the procedure.



(a) 2D dilated convolving

(b) Resulting feature map

Figure 3.3: 2D dilated convolution. In this example dilatation rate = 2.

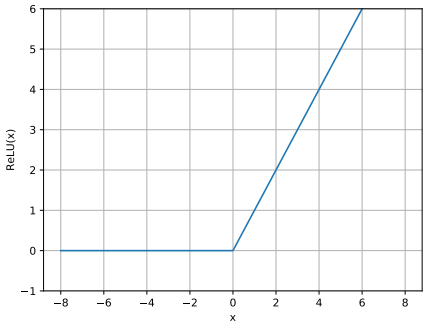


Figure 3.4: ReLU activation function

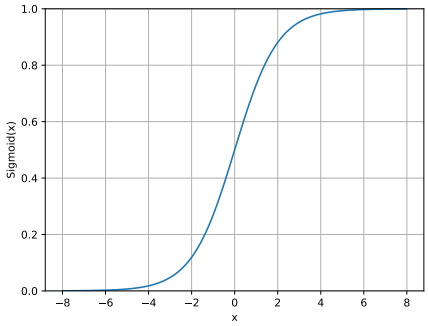


Figure 3.5: Sigmoid activation function

3.1.2 Activation Layer

Artificial neuron computes the weighted sum of its inputs and adds a bias. Activation function transforms this input and decides whether the neuron is fired (activated) or not - whether its value is significant or negligible.

Activation function introduces non-linearity on input signal. Not using non-linear activation functions essentially leads to a linear regression model [44]. The purpose of a non-linear activation function is to apply a non-linear transform to its input. The proposed model in this thesis uses two non-linear activation functions. Namely, Softmax and ReLU.

Softmax is useful in a multi-classification network architecture. Raw (non-normalized) predictions of the last linear layer (also known as logits) are passed to this activation function. This function turns them into a vector representing probability distributions of labels. The sum of the vector is equal to 1 (all probabilities add up to 1). Notice, in the Figure 3.1 probabilities also sum up to one. Equation 3.3 describes the activation function.

$$\text{Softmax}(y_i) = \frac{e^{y_i}}{\sum_{j=0}^{N} e^{y_j}}, \tag{3.3}$$

where:

- e: is the Euler's number
- y_i: is the raw input (logit)
- N: is the raw input vector size

Using exponentials squashes values inside input vector to be between 0 and 1. There is a similar activation function to Softmax and it is Sigmoid (see Figure 3.5). Sigmoid also outputs values between 0 and 1 but its sum need not to add up to one. High values have high probability, but not higher than others unlike in Softmax case. Sigmoid is a useful activation function in binary classification, whilst in multi-class classification Softmax is preferred.

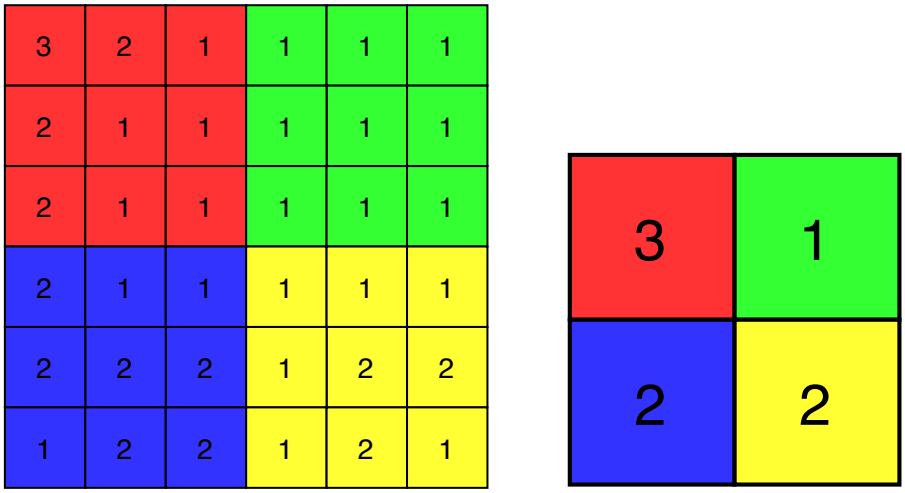
ReLU stands for Rectified Linear Units. Despite its name the function is actually non-linear. This function is used in the proposed model in all 2D convolution layers. It is chosen for two reasons. It is non-linear and it is fast for computation (see Equation 3.4).

$$\text{ReLU}(x) = max(0, x) \tag{3.4}$$

The equation is really straightforward and simple as it outputs values between 0 and x. Figure 3.4 depicts the graph of ReLU function.

3.1.3 Pooling Layer

Pooling layer is used to reduce the spatial dimensions (down-sample input signal). The layer acts as a filter which slides over the input image and summarizes features in it. Down-sampling input feature map leads to less parameters in a neural network model and thus to smaller computational effort. Kernel is a window defining a pooling strategy (for example: average pooling, max-pooling, ...). Kernel slides over a feature map with a specified stride and applies the chosen strategy on values located within the current window. Figure 3.6 shows max-pooling with $stride = 3$ on a 3×3 feature map.



(a) 3×3 max-pooling with stride $= 3$ (b) Feature map after max-pooling

Figure 3.6: 2D max-pooling process.

Max-pooling is also used in the proposed model (see Section 3.4) in the VGG-16 front-end. Choosing an optimal down-sampling strategy of feature maps leads to better generalization as well as less computational overhead.

An extreme variation of pooling is global pooling. Global pooling uses a window with the size equal to the input feature map to perform a downsampling [3]. Therefore, input feature map size with dimensions $h \times w \times d$ is reduced to $1 \times 1 \times d$. For each dimension d there is only one value to represent down-sampled features within that dimension. An example of this pooling is global average pooling (GAP for short). GAP is useful in multi-classification tasks. In cases where fully-connected layers cause over-fitting and/or computational overhead [36]. GAP on the other hand has no parameters to tune (since it is a pooling strategy). PDANet [2] (see also Subsection 3.3.4) makes a use of such pooling.

He et al. [13] used another type of pooling strategy - Spatial Pyramid Pooling (SPP for short). The strategy addresses the need of having an input image of fixed size in some architectures. Specifically, Fully-connected layers require input of a fixed-size. This technique partitions image from finer to coarser levels and outputs a fixed-size vector. Figure 3.7 describes the procedure on an example. Using this technique avoids cropping and distorting the input image. Pyramid Feature Extractor in the aforementioned PDANet is inspired by SPP.

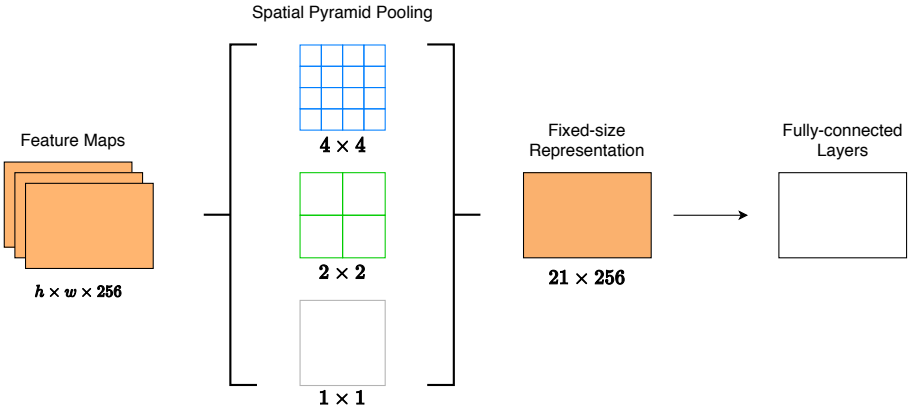


Figure 3.7: 2D Image Spatial Pyramid Pooling example. Each one of the 256 feature maps is pooled with three sets of pooling filters with the same pooling strategy (max-pooling for example). At the end there is $(4 \cdot 4 + 2 \cdot 2 + 1 \cdot 1) \times 256 \Rightarrow 21 \times 256$ fixed-size vector of pooled features. The output vector can be directly sent to fully-connected layer, since they already have the needed fixed size dimensions. There are two things worth noticing. Firstly, height and width dimensions values are arbitrary. Secondly, the 1×1 pooling acts like Global Max Pooling.

3.1.4 Fully-connected Layer

Fully-connected layer uses the output of the convolution/pooling layers. These features are flattened into a single vector of values [28].

Neurons in this layer are fully-connected. Each neuron's activation function output is the input for all neurons in the following layer. In a multi-classification task Softmax activation function could be applied to the last layer of neurons. These neurons would represent a probability for the input image to belong to a certain class/label. Figure 3.8 demonstrates a use of Softmax function.

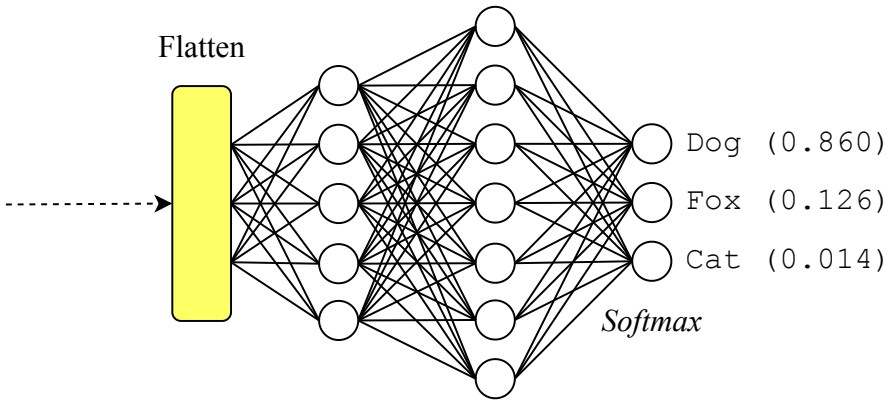


Figure 3.8: Fully-Connected layer with Softmax - classification task.

Given a small dataset with a very limited data, training can potentially lead to over-fitting and thus worse generalization of the model. A common technique to reduce the odds of over-fitting is to use a Dropout. The idea is to randomly drop neurons along with their connections during training. Dropping neurons basically introduces many variations

of a given model during training (as it alters the structure of neurons). Using Dropout typically makes the training 2-3 times longer [42]. The major reason behind this are very noisy weights updates (training on frequently altered model).

3.2 Vehicle Counting

Traffic analysis is still a challenging task. During such task, there are many pitfalls to be aware of. Such as small image resolution, high number of overlapping objects, angle of camera, blurred objects due to their motion or weather conditions. This thesis addresses the issue of counting vehicles instances in images and videos.

Data acquired from the automated traffic flow analysis can improve management of traffic. Busy highways or road intersections can be monitored for traffic congestions to see how they evolve. Public authorities in charge of the maintenance and planning of road infrastructures can make use of such information.

There are many approaches to count objects instances in imagery. According to Loy et al. [25], crowd counting (could be also applied to vehicle counting) can be divided into three paradigms:

- Counting by Detection (Subsection 3.2.1)
- Counting by Clustering (Subsection 3.2.2)
- Counting by Regression (Subsection 3.2.3)

This thesis is focused on the third paradigm – Counting by Regression.

3.2.1 Counting by Detection

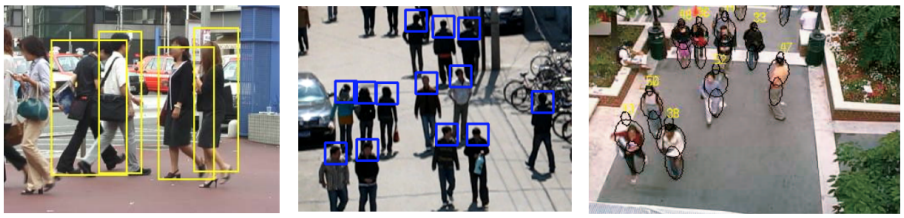


Figure 3.9: Counting by Detection methods on pedestrian detection example [25]. From left to right: Monolithic Detection, Part-based Detection, Shape Matching Detection.

This approach can be divided into three main types based on features identified within the scenery. Figure 3.9 shows these three types on a detection example.

The first type being Monolithic Detection. In this approach classifier is trained in a whole-object fashion. Monolithic Detection is preferred in a scene with sparse crowd count to densely crowded scene. Dense images present issues for classifier, since objects are overlapping or the image borders truncate some parts of the object.

Turning the monolithic based classifier into part-based is the second type. Part-based Detection classifier is trained to look after object parts rather than a whole object. For example head and shoulders of people are detected to estimate the count.

The last is Shape Matching Detection. A set of defined body shapes with ellipse boundaries is used along with a stochastic process to estimate the shape configuration and the count.

3.2.2 Counting by Clustering

(a) Good example of clustering.

(b) Sometimes persons are merged in detection.

Figure 3.10: Examples of clustering paradigm (both images are from the paper [37]).

The paradigm is based on an assumption that a motion of individual or visual features are relatively uniform. A coherent trajectory features are then grouped and may represent each individual object. As mentioned, the motion stands behind this idea, hence the paradigm cannot be applied across static images. Another issue arises when the assumption of motion coherency does not hold true. This happens in cases where the objects are remaining static or two objects are sharing common feature trajectories over time [25]. An example of clustering paradigm is shown in the Figure 3.10.

3.2.3 Counting by Regression

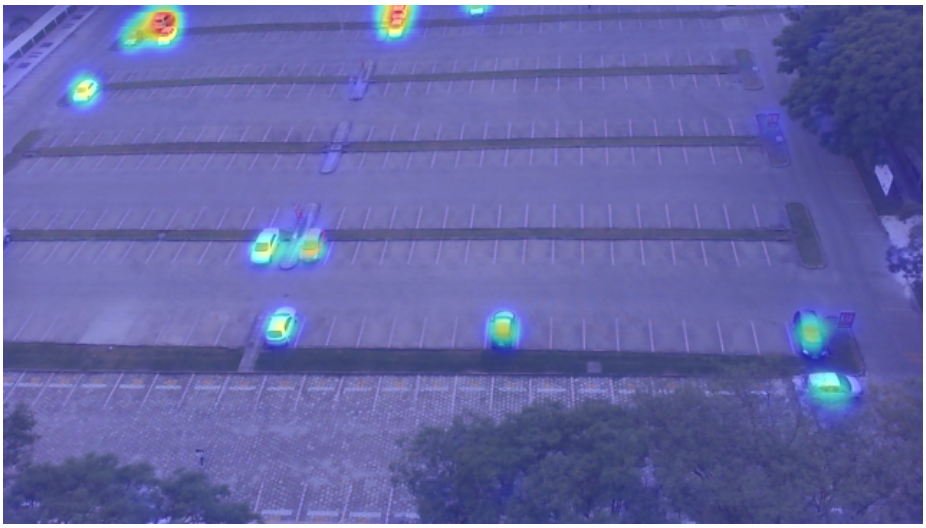


Figure 3.11: Example of Counting by Regression paradigm. Vehicles detection estimation on the image from PUCPR+ dataset (see Subsection 4.4.2).

The two aforementioned paradigms have their limitations. Counting by Detection is inefficient in dense crowds. Counting by Clustering requires a motion to take place. Counting by Regression addresses both of these issues. No individual detection and no motion are involved. The approach will be described in more detail than its aforementioned paradigms,

since this paradigm is the aim of this thesis. A detection example of the paradigm is shown in the Figure 3.11.

Counting by Regression belongs to a supervised learning category. In supervised learning a ground truth (expected output) must be provided (this is also a limitation of the approach). One technique of the Counting by Regression approach is to use heatmaps. In this technique a pair of input image and its expected output heatmap image is provided to the model. Training phase of the model 4.3.1 is then basically learning mapping between ground truth images and input images.

Following the idea of Lempitsky et al. [22] the counting problem is then a process of recovering a density function as a real function of pixels of input images. A predicted heatmap patch of objects is the output from a network of this technique. The expected heatmap patch is made by applying Gaussian blur with a reasonable standard deviation value σ (in this work $\sigma = 15$) centred on each dot in the annotated image. Figure 3.12 shows the conversion of annotated image to ground truth image.

(a) Annotated image

(b) Resulting Ground Truth image

Figure 3.12: Conversion example of annotated image to ground truth image. Gaussian blur with $\sigma = 15$ is applied to the annotated image (greyscale image with white dots representing the position of each vehicle). In this figure dots are represented by red crosses to see annotated vehicles more clearly. The input image is from TRANCOS dataset (see Subsection 4.4.1).

However, this will get us a full ground truth image. We need to split input images and ground truth images into patches. Note that each input and ground truth patch pair must have matching coordinates (their patch coordinates must correspond).

Gaussian blur uses a kernel whose values add up to 1. This ensures no energy is added or removed from the image after this operation. Thus, the sum of convolved dots (pixels) in the annotated image is unchanged (the vehicle count is preserved). To get the final count we simply sum up the density map. Equation 3.5 describes how the count is retrieved from density map. The equation is proposed in the paper *Towards perspective-free object counting with deep learning* [34]. Notice that Gaussian blur considers overlapping vehicles and also supports partial-count. When an image is truncated it makes sense to count a car as a fraction of 1 (like 0.778) instead of 1.

$$N_I = \sum_{p \in I} D_I(p)\,, \tag{3.5}$$

where:

- N_I: is the object count in a density image I
- D_I: is the density image I
- p: is the pixel

Final prediction density map is created by assembling patches together. There might be a pitfall when some down-sampling or up-sampling strategies are used in a CNN architecture (such as pooling - see Subsection 3.1.3). Resized input patch has to be resized back to its original resolution. Simply stretching or shrinking the patch could change the count (hence patch normalization is needed). In the *Towards perspective-free object counting with deep learning* paper [34] a normalization formula is proposed to solve such issue (see Equation 3.6).

$$\hat{D}^{P+}_{pred} = \frac{\sum_{\forall p} {D^P}_{pred}(p)}{\sum_{\forall p} \hat{D}^{P}_{pred}(p)} \cdot \hat{D}^{P}_{pred}\,, \tag{3.6}$$

where:

- $\hat{D}^{P+}_{pred}$: is the rescaled density patch *with* a matching count
- ${D^P}_{pred}$: is the original (not rescaled) density patch
- $\hat{D}^{P}_{pred}$: is the rescaled density patch *without* a matching count
- p: is the pixel

Dense patch extraction is a chosen (default) method to produce a final density estimation. In this thesis the stride is set to 10 pixels in both vertical and horizontal directions. Each patch has $\frac{1}{2}$ of height and $\frac{1}{2}$ of width of its input image (a quarter of it). Since patches are overlapping the final density map must be normalized. Each position (pixel) in the final density map is an average of patches that cast a prediction in it.

3.3 Related Work

In this section I will briefly introduce CNN architectures which inspired my work. I will provide a brief summary of each and also mention the advantages and disadvantages where possible.

3.3.1 Multi-column Deep Neural Networks for Image Classification

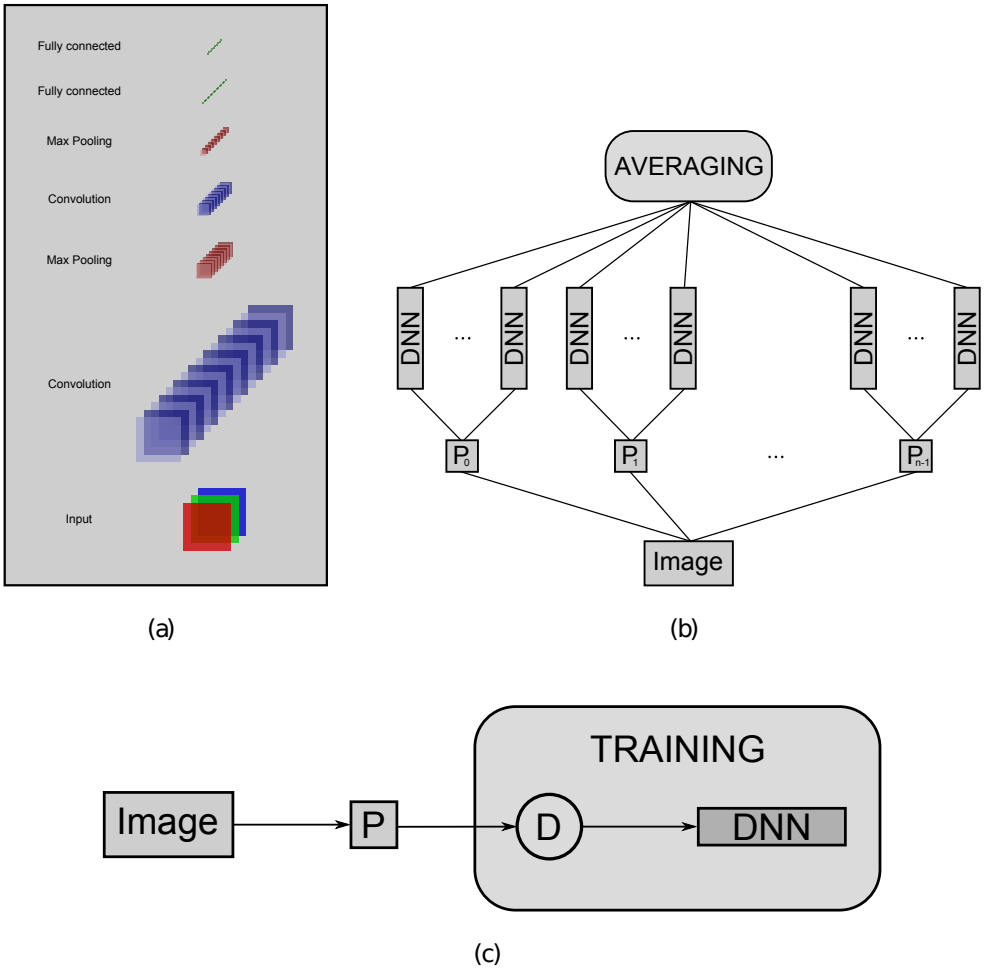


Figure 3.13: Multi-column Deep Neural Network architecture overview [4]. (a) Deep Neural Network (DNN) architecture. (b) Multi-column Deep Neural Network (MCDNN) prediction phase. Input image can be preprocessed ($P_0 - P_{n-1}$ blocks) and is sent to DNN columns in parallel. Final predictions are made by averaging DNN columns. (c) Training a DNN column. Here P and D are representing preprocessing and image distortion blocks respectively.

This network architecture [4] is inspired by vertical column-like arrays of neurons (located in the temporal cortex of human). The model is made of vertically stacked columns to learn image features independently and in parallel. Each column consists of the same combination of convolutions, pooling layers and fully connected layers.

In prediction phase input image gets a unique preprocessing before it is passed into a column. Image features are extracted in parallel in each column. At the end all columns are averaged and final output is produced. Training is almost same as prediction phase. What is different is a DNN getting a distortion (at the beginning of every epoch) on a preprocessed input image before it is sent to the column. Figure 3.13 shows the architecture along with its training and prediction phases.

3.3.2 Counting CNN

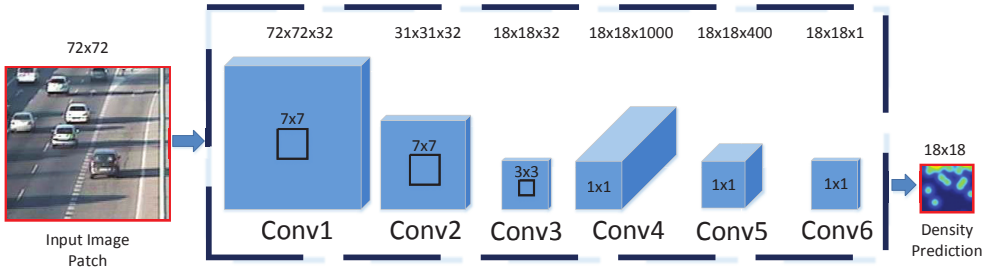


Figure 3.14: Counting CNN architecture [34]. A very straightforward architecture comprising of only 6 convolutional layers. An input image patch of fixed-size is passed forward to the network and corresponding density image patch is the direct output of the architecture.

One of the proposals of the paper *Towards perspective-free object counting with deep learning* [34] is the Counting CNN. The model falls into Counting by Regression category, which is the category covered in this thesis.

In training phase 800 RGB patches are randomly cropped with a fixed size from the input image. All patches are augmented by vertical flipping making it 1,600 patches in total. Each patch has dimensions $72 \times 72 \times 3$ and $18 \times 18 \times 1$ patch is the direct output from the network. The prediction is done via dense patch extraction (see Subsection 3.2.3) and the output patches are assembled together to generate the final ground truth estimation. Counting CNN is also trained on the TRANCOS dataset (see TRANCOS) and reported an improved accuracy on this dataset [34]. Figure 3.14 depicts the architecture.

Their architecture is rather small than large (6 convolutional layers in total). This makes it relatively fast to predict an output patch from the input one. If the number of training patches is smaller, training will be a lot faster with not so big accuracy impact.

3.3.3 Context-Aware Crowd Counting Network

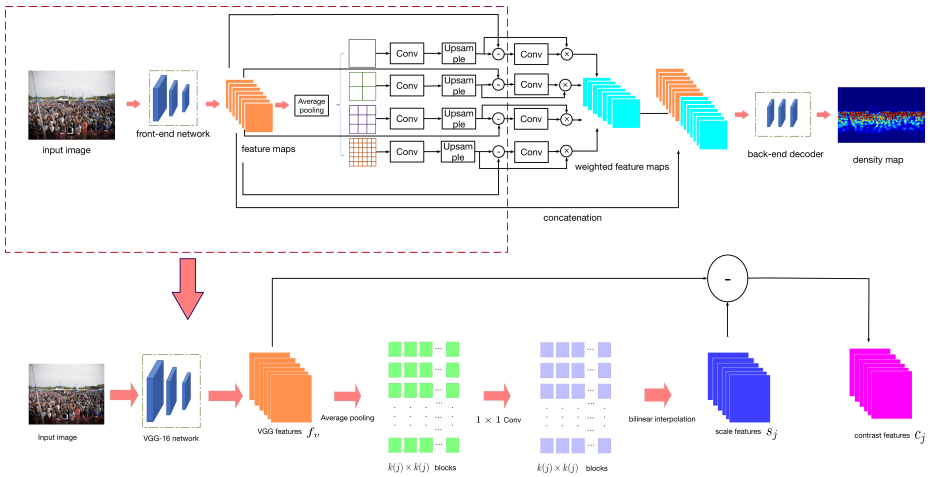


Figure 3.15: Context-Aware Crowd Counting Network [23]. **(Top)** Architecture overview. **(Bottom)** Detailed view of the **Top** figure section (red dashed box) which captures computation of the contrast features C_j.

Even if the Counting CNN tries to solve the perspective distortion by random patch extraction, its results indicate severe inaccuracies since the scale continuously varies across the whole image [23].

The Context-Aware Crowd Counting Network [23] (CAN) adaptively encodes a multi-level contextual information into features it produces. The starting point of the architecture is VGG-16 [40] front-end (first 10 layers of the VGG-16 model). The front-end uses a method known as transfer learning. This means pre-trained weights are downloaded and used instead of initializing the weights by some distribution, thus significantly reducing the training time. CAN model downloads pre-trained weights from the ImageNet dataset [6] and adjusts (trains) them to fit a particular dataset. The model performs so-called Spatial pyramid pooling [13] (see also Subsection 3.1.3) to compute scale-aware features S_j. Each scale-aware feature has its relative influence at each spatial location determined by weight maps (learnt values during training). Contrast features (C_j) are the difference between VGG-16 features and S_j and they help to understand the local scale of each image region. Architecture is finished with the Back-end decoder which produces a final density map estimation.

Training phase of CAN is different than in Counting CNN (in words of patch extraction). The input image is divided into four regions of equal size (non-overlapping). One of these four regions is randomly chosen and passed to the net.

3.3.4 Pyramid Density-aware Attention Net for Accurate Crowd Counting

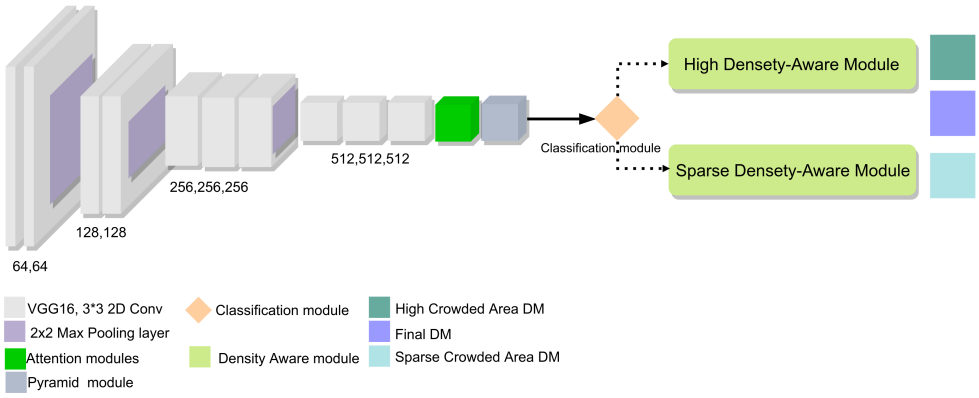


Figure 3.16: Pyramid Density-aware Attention Net architecture overview [2]. Upper branch (High Density-Aware Module) is adjusted for dense features and the lower branch (Sparse Density-Aware Module) is adjusted for sparse ones.

A very recent architecture [2] (January 2020) to the time of writing this thesis. PDANet continues to improve the accuracy of the CAN model.

This model also makes use of the VGG-16 front-end for low-level feature extraction. Pyramid feature extraction with spatial and channel attentions are attached to the front-end to produce richer features. Their work focuses on different levels of crowdedness. Classification module is proposed to adapt to different levels of crowdedness. Within the decoder module there are two branches: dense and sparse. The model is trained in a such way that the weights in the upper branch (dense) are adjusted for dense features and weights

in the lower branch (non-dense) are adjusted for sparse ones. At the end these branches are combined together to produce a final density estimation.

3.4 Proposed Architecture

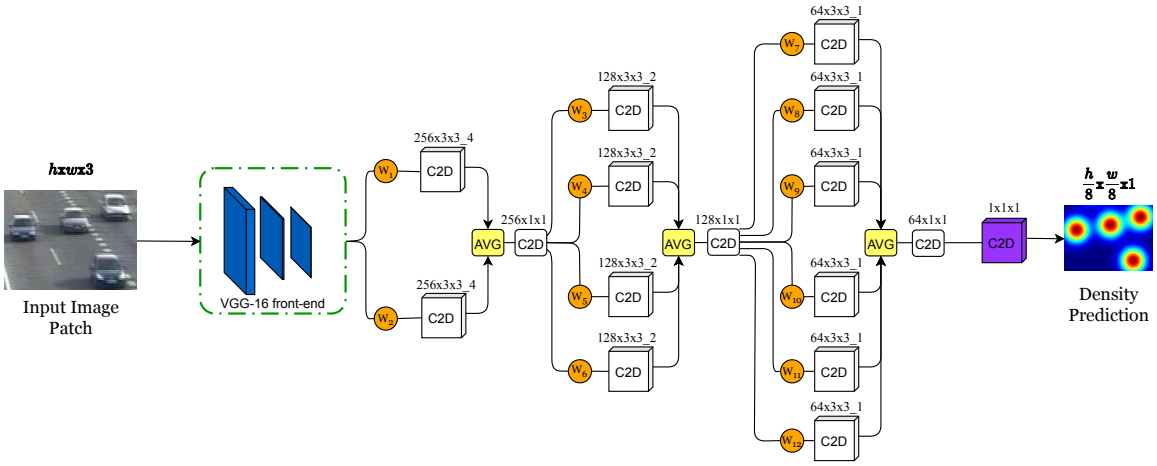


Figure 3.17: Proposed architecture. RGB patches are fed to the VGG-16 front-end network. Resulting low-level features are weighted (neurons w_i) and sent to dilated convolutions in parallel. Averaging followed by 1×1 convolving ends one such parallel layer. Dilated convolutions in parallel layers increase the receptive field of the network and help to understand the overall picture. The last parallel layer gets additional 1×1 convolving and produces density estimation. Input patch height and width dimensions are reduced by a factor of 8 due to two max-pooling layers in the VGG-16 front-end.

The proposed architecture makes use of VGG-16 front-end for low-level feature extraction. The CAN and PDANet use only first 10 layers of VGG-16 and so does the proposed solution. The architecture is depicted in the Fig. 3.17. The front-end is then connected to a tree-like structure, which is inspired by aforementioned Multi-column network (see Subsection 3.3.1).

One layer (structure) of the entire tree-like structure is composed as follows. Input to this layer gets weighted n-times (where n is the number of parallel conv layers within the corresponding structure). Each weight w_i represents one neuron, whose value is determined during training phase. These weights use Softmax activation function. Initially the value of each weight w_i is set to an increasing multiple of $\frac{1}{n}$ (n is the number of parallel convolutions in the corresponding structure). So for example the last structure has the initial values $w_7 = \frac{1}{6}$, $w_8 = \frac{2}{6}$, $w_9 = \frac{3}{6}$, $w_{10} = \frac{4}{6}$, etc.

Each weight is connected to a dilated conv layer (C2D white box in the Fig. 3.17). All dilated convolutions use the same kernel size 3×3 but they differ in dilatation rate and filter depth. The setup of these conv layers is shown in the Table 3.1 (structure count starts from the front-end).

Dilated convolutions are then averaged. Final 1×1 convolving ends one such structure. All convolutions in each structure are followed by ReLU activation function. Softmax and ReLU activation functions are also described in the Subsection 3.1.2.

Table 3.1: Dilated convolutions setup.

Structure number	Filter depth	Dilatation rate
1.	256	4
2.	128	2
3.	64	1

The final structure gets additional 1×1 convolution to produce a density estimation for a given input patch.

The weighting idea came from PDANet classifier, which is responsible for giving each of the two branches unique weights (see Subsection 3.3.4). Dilated convolutions increase the receptive field of the network and help to understand the overall picture instead of finer details [27].

Chapter 4

Proposed Architecture Implementation

A cloud solution is used during the implementation of the proposed architecture to speedup training along with evaluation of the model. This chapter describes the environment used along with implementation details. This involves details about how the model is trained and how prediction is carried out on the input image. The implementation is evaluated on datasets mentioned here.

4.1 Implementation Environment

I have chosen to implement the network in Keras framework [20]. I wanted a framework which meets certain criteria. First criterion being a user-friendly framework, meaning there is not a very steep learning curve. Secondly, I wanted a framework written in a known and used programming language, which in Keras case is widely used Python[1]. Lastly, framework's support in case of issues and bugs is also an important factor. The Keras API is documented, with examples [17] [16] [18] and for troubleshooting there is a dedicated Google group forum [11]. Keras API is built on top of TensorFlow[2] back-end [20].

Training a convolutional neural network can be a very resource hungry task. Simple networks can be trained on CPU. However, it is reasonable to take computational alternatives into consideration when operating with large architectures. The most common alternative is to use GPU instead of CPU. In such case there is an option to use NVIDIA's proprietary framework NVIDIA CUDA ®Deep Neural Network library (cuDNN) [31]. cuDNN is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers [31]. In order to use this library there are several constraints [30]. First and foremost, graphics card must be of brand NVIDIA. Next, one has to be a registered member of the NVIDIA Developer Program. Along with this your GPU must have compute capability 3.0 or higher. If these constraints are satisfied, the user is then only limited by their GPU performance.

To avoid these installation issues and get potentially better computational power than on desktop/laptop I chose Google's Colab [8] solution. The service is free to use and

[1] https://www.python.org
[2] https://www.tensorflow.org

only requires you to have a Google account. The interface is basically Jupyter notebook[3], which is allows interactive programming/computing. Colab offers to choose between three types of hardware acceleration. First being CPU, second GPU and the last type is TPU [10]. TPUs are Google's custom developed circuits specifically for purposes of training neural nets [10]. According to their TPU recommendations [10], it is good to consider this acceleration if a model training takes weeks or months. For purposes of this thesis GPU accelerator is more than sufficient. There are a few limitations and restrictions for using free Colab environment as per their FAQ [9]. The internet browser window cannot be closed during code execution, otherwise a virtual machine (VM for short) we got assigned looses its resources and changes are lost. Assigned VM gets shut down nonetheless. Each assigned VM has a maximal lifetime of 12 hours (this varies and is based on one's usage of shared resources). There might be other restrictions for assigned resources, but these two should be kept in mind. Fortunately Keras allows us to make checkpoints (savefiles)[19] in HDF5®format [43] (*.h5 file extension). Regularly saving weights compensates this VM disconnection issue. Custom progress data can be saved to a text or json file [15].

Training setup was not very complicated. I uploaded all prepared datasets to Google Drive. From Colab's Jupyter notebook I mounted the drive containing datasets and model source code files. The command syntax for mounting a drive is:

```
from google.colab import drive
drive.mount('/content/drive/')
```

The drive is mounted. To change to a different working directory one can issue:

```
%cd NEW_WDIR_PATH
```

where NEW_WDIR_PATH is a new working directory path. Colab allows both code execution like in Jupyter (source code is written directly into notebook cells, which are executable) and both encapsulation of ones code into Python files. Colab has a built-in text editor for file manipulations. I chose file-like structure approach, but it is really just a matter of preference. To execute a Python source file all that is needed is to connect to a runtime and issue the command:

```
!python src_code.py
```

For Jupyter style only a code cell execution is needed (no need to specify a file to execute).

4.2 Configuration File of the Model

During the development of the proposed model I have created a `json` config file. This decision was due to frequently changing network parameters and it is intended to avoid hard-coding such values into Python source files. JSON (JavaScript Object Notation) [15] is a lightweight data-interchange language. This language is widely used and supported (Python has a built-in library for `json`). The Listing 4.1 shows the content of the config file. The meaning and purpose of each parameter will be concisely described. Sample values are already inserted in this listing to make the explanation simpler.

[3] https://jupyter.org

```
{
  "patch_dims": {
    "input": [240, 320],
    "gt": [30, 40]
  },
  "batch_sizes": {
    "training": 2,
    "prediction": 5
  },
  "in_paths": {
    "annotated_dataset": "Dataset1",
    "weights": "model_weights_input.h5",
    "image_raw": "Dataset2/input_images/demo_image1.png",
    "image_dot": "Dataset2/dot_images/demo_image1.png",
    "video_raw": "Dataset2/raw_video.mp4"
  },
  "out_paths": {
    "test_metrics": "metrics_eval_1.txt",
    "weights": "model_weights_output.h5",
    "image_pred": "demo_image_pred.png",
    "image_gt": "demo_image_gt.png",
    "video": "video_pred.mp4"
  },
  "dense_stride": -1,
  "gauss_sigma": 15,
  "base_lr": 0.00001,
  "queue_size": 1,
  "max_epochs": 100,
  "validation_split": 0.2,
  "model_phase": "training"
}
```

Listing 4.1: Config of the model (Json file). The config uses explicit file name extensions.

On the first line `patch_dims` parameter defines patch height and patch width respectively. The first entry (`input`) is for input image and the other (`gt`) for ground truth image. Input image patch dimensions are expected to be $\frac{1}{2}$ of height and $\frac{1}{2}$ of width of the input image. Ground truth dimensions are computed from input image patch dimensions. Dividing input image patch dimensions by 8 (due to two max-pooling layers in the VGG-16 front-end) results in patch dimensions expected by the model. The `batch_sizes` parameter sets the batch size - number of image patches per batch. This parameter is adjustable for both training and prediction phase.

Next is directory/file paths setup. This setup is divided into two groups. The first group (`in_paths` - *in* group for short) is intended only for file reading, whilst the other one (`out_paths` - *out* group for short) is only for file (re)writing. `annnotated_dataset` expects a path to a dataset to train or to evaluated the model on. This path expects a specific directory/file structure. Figure 4.1 shows the expected file hierarchy and describes what it should contain (describes Must-Haves). This decision caused less changes in source code and thus made the implementation less error-prone. Output of an evaluation of the model

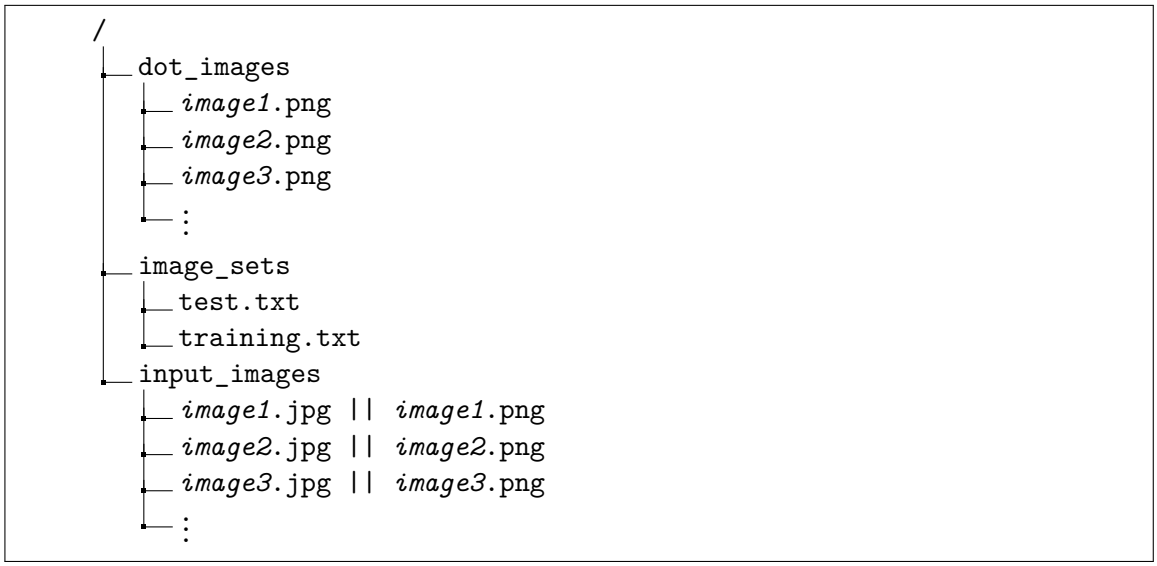


Figure 4.1: Expected `annnotated_dataset` file hierarchy. `dot_images` is a directory with *.png greyscale annotated images (dot images). Subsection 4.3.1 also explains why PNG format is expected here. `image_sets` directory contains two files which both contain file names (without *.png or *.jpg extensions) for test and training phase of the model. It is required that `input_images` directory file names are paired with `dot_images` file names.

on test set is written to the `test_metrics` file (needs to be specified only during test phase). During training phase of the model, weights are regularly saved each epoch to the file specified by parameter `weights` in the *out* group. Every but training phase of the model requires weights to be loaded, weights located inside `weights` file in the *in* group are loaded. Keras uses HDF5®files (*.h5 file extension in the path parameter) to store/load network properties. HDF5®is targeted for fast I/O processing and storage [43]. There are three parameters in both groups to perform individual image or video prediction. `image_raw` and `image_dot` are parameters for input image and its corresponding dot image. Dot image is optional - provides ground truth comparison if needed. The results of these two parameters are saved in files specified by `image_pred` and `image_gt` in the *out* group. The proposed model also supports video prediction. `video_raw` specifies an input *.mp4 video and `video` is the path of the resulting video with prediction.

Proposed architecture (Section 3.4) uses a dense patch extraction by default to do predictions. This means that the model also supports another extraction strategy - Quadrant patch extraction (see Subsection 4.3.2 for more details). The config file specifies both horizontal and vertical stride in pixels for dense extraction with parameter `dense_stride` (setting this value to `-1` changes the default strategy to the alternative extraction). Heatmap patches are created by applying Gaussian blur centered on each dot in annotated image, `gauss_sigma` specifies the σ value. Training starts with a base learning rate `base_lr`. Generation of training batches is queued up to `queue_size` batches. Model training is stopped after reaching `max_epochs` epochs. At the end of each epoch a model validation takes place. `validation_split` specifies how many images specified in the *training.txt* should be reserved for validation (a fraction part in range (from 0.0 to 1.0) which is multiplied with the total number of images specified). Lastly, `model_phase` denotes what operation the model should be carrying out when the code is being executed:

- training and validation (Subsection 4.3.1)
- test phase (Subsection 4.3.2)
- image prediction (Subsection 4.3.3)
- video prediction (Subsection 4.3.3)

4.3 Model Phases

The proposed architectures uses the VGG-16 [40] front-end (first 10 layers from the VGG-16) which includes 2 max-pooling layers. A pooling layer reduces spatial dimensions, as mentioned in the Subsection 3.1.3. Initially, both input image patch and expected ground truth image patch are cropped with the same size (a quarter of the image). Ground truth patch is scaled-down by a factor of 8 to reflect this change. To preserve the vehicle count the normalization formula for rescaling is used (see Equation 3.6).

For image manipulations OpenCV (Open Source Computer Vision Library)[4] is used. Even though it is written in C++, it has many interfaces including Python. This widely used library uses BGR colour format instead of RGB [26]. This colour format of OpenCV should be kept in mind during image manipulations or when combining other image libraries which might expect RGB format. All images processed by the network are normalized (original RGB range from 0 to 255 is normalized to be between 0.0 and 1.0).

4.3.1 Training and Validation

The proposed model uses Mean Square Error (MSE) loss and Adam optimizer to compute weights. It is very convenient to save model state during training (to use it later, to stop and resume training, to have a backup if training stops unexpectedly, etc.). Keras offers whole-model saving as well as just saving weights [19]. The first mentioned option includes saving a model architecture, state of the optimizer, training configuration and finally weights. The other option saves just model weights. Since datasets used during training are relatively small (see Section 4.4) and the training procedure on Colab was not very time-consuming (see Table 5.1) the latter option is used in this implementation.

At the beginning all image paths in the *training.txt* are shuffled and a fraction of them is reserved only for validation. Both input image and dot image pairs are loaded and normalized. Gaussian blur is applied on dot image to produce ground truth (see Figure 3.12). I have decided to use PNG format instead of JPG for dot images. The reason is that JPG format is lossy and causes unwanted information loss [7]. In the aforementioned article PNG is said to be non-lossy, although it uses more disk space in comparison to JPG. Hence, to prevent dot image from potential information loss and consequently to prevent precision loss of ground truth the PNG format is used.

Patch extraction is done by dividing an input image into four regions of equal size (non-overlapping). One of these four regions is randomly chosen and the very same region is cropped from the ground truth image. Ground truth image is scaled-down by a factor of 8.

Augmenting input data is a technique to artificially expand datasets when training. Doing so can potentially lead to a better generalization of the network and consequently to less biased predictions. In the thesis I used some of the augmentations to make the network more robust. I have employed four random augmentations: horizontal flipping, gamma

[4] https://opencv.org/about

correction, Gaussian noise and random zooming. Each patch gets a random combination of these augmentations, so there is a smaller chance for the network to encounter the same batch throughout epochs. In addition to this, patches inside a batch are shuffled at the end of each epoch to further support generalization. Figure 4.2 shows used augmentation techniques on an input image (in the figure an image is used instead of an image patch only for a better visualisation - the network augments patches during training).

(a) An image from CARPK dataset before augmentation.

(b) Augmentation techniques applied on the input image.

Figure 4.2: Augmentation techniques used during training. Augmentation techniques **from top to bottom:** 120% image zoom, image horizontal flip, image with 1.4 gamma, image with added gaussian noise (with $\mu = 0$ and $\sigma = 0.05$).

Patches are then enqueued for the net. In this context batch size denotes number of patches sent to the queue (number of both input patches and both ground truth patches). The size of the queue (number of queued batches) and batch size are adjustable parameters

in the config file. Increasing batch size or queue size can reduce or even eliminate batch generating bottleneck as well as it increases RAM usage. Finding optimal values depends on available hardware and resources.

Validation loss (uses a reserved fraction of training images) is monitored during training. Whenever this loss drops (the threshold for improvement here is $1e-6$ difference), a model checkpoint is made. Model weights are written/rewritten to a *.h5 file specified in the config file. Also, whenever this loss does not improve (does not decrease) over 11 epochs, the base learning rate decreases. New learning rate is computed as $\frac{1}{2}$ of the previous learning rate. There is lower bound for learning rate ($1e-8$) specified, which stops decreasing learning rate any further than the threshold. There is a max number of epochs during training phase. After reaching this number the training is stopped.

4.3.2 Test Phase

Each dataset mentioned in the Section 4.4 has a test set. Images in test set are generally used to report one's model accuracy. Accuracy of the proposed model is evaluated as follows.

Each test input image is loaded and split into four equally sized non-overlapping quadrants (patches). This strategy is the same as in the Context-Aware Network (as per their source code for test phase [24]). Not using dense patch extraction presents some accuracy impact (see Figure 4.3 for a visual comparison). On the other hand this approximation speeds-up test phase significantly (only four patches per one test image).

Input test image patches are sent to the network for a prediction. Model weights specified in the config file are loaded and used throughout test phase. Output patches are assembled into an estimated ground truth image. Corresponding dot image is loaded and gets applied Gaussian blur to produce the expected ground truth image.

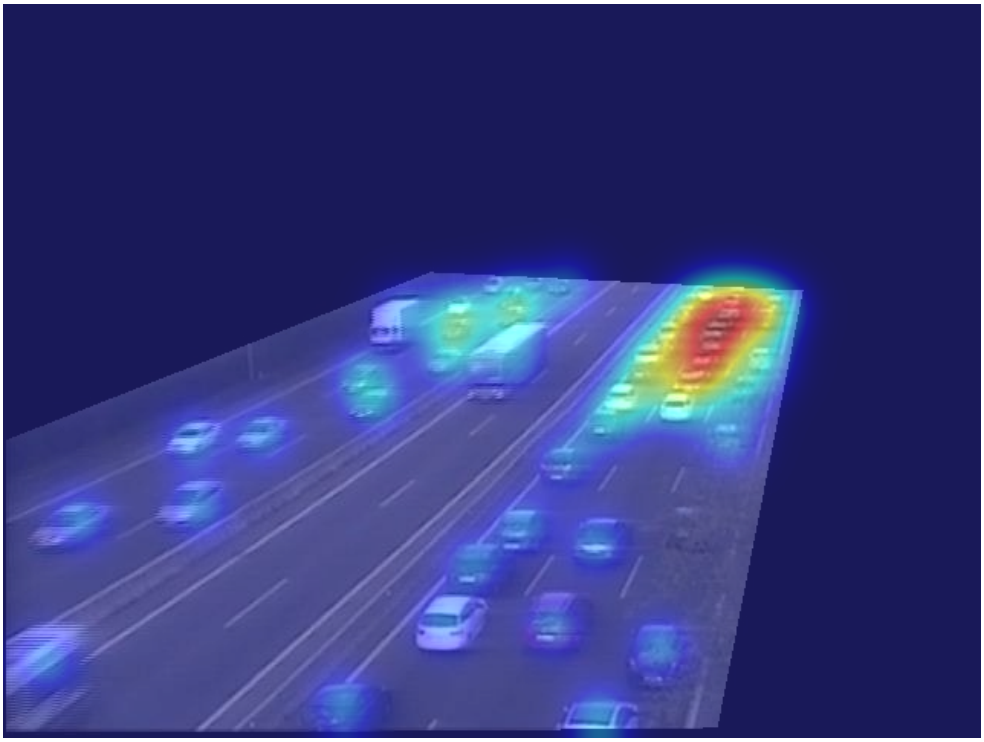
With ground truth image and predicted image the error rate (accuracy) can be evaluated. I used the same metrics as each dataset in the Section 4.4 required or reported in state-of-the-art results. Error rate is computed as a difference between expected and predicted vehicle counts. Vehicle count is obtained by integrating over both density maps (summation of density pixels). Error rate computation is described in the Chapter 5. Computed metrics are saved to a plain text file specified in the config file.

4.3.3 Prediction

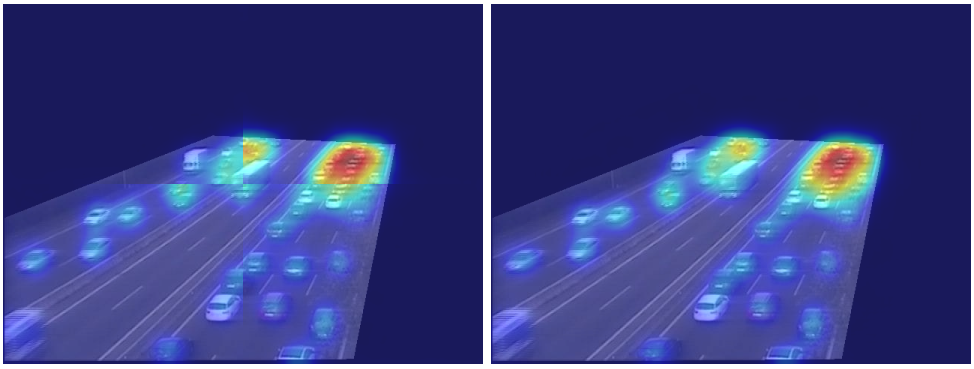
The implementation provides two methods to perform a prediction. First being prediction on one input image. In the config file the paths to input image and its respective dot image are set. The use of dot image (converted into ground truth) is merely to compute error rate and thus it is optional.

The second method is to perform prediction on a video input. Counting instances in video is basically the same as counting objects in images. Input video is split into frames where each frame is passed through the net and gets its prediction. Frames are then encoded back into a video file (frame order has to be preserved). Implemented output video format is MP4. Using a ground truth image in the first approach is optional, while here only a prediction is performed.

The rest of the procedure is the same for both methods. Input image is loaded along with specified weights file. Dense patch extraction or Quadrant patch extraction is performed on the input image. The lower the `stride` parameter (configurable parameter) the more dense extraction is and vice versa (setting `stride` to `-1` switches to Quadrant extraction).



(a) A ground truth image from TRANCOS dataset.



(b) Quadrant patch extraction (on the **left**) vs. Dense patch extraction (on the **right**).

Figure 4.3: Dense and quadrant patch extraction techniques comparison. In this case dense extraction produces 825 patches (horizontal and vertical *stride* = 10 pixels), whilst quadrant approach produces only 4. Ground truth image has 44.03 vehicles, quadrant and dense approaches have predictions of 40.43 and 41.89 vehicles respectively.

Dense patch extraction has a trade-off, however. More dense extraction means more precise prediction but also requires more time since it generates more patches. Figure 4.4 shows dense patch extraction procedure on an example.

This thesis uses *stride* = 10 pixels. Colab environment (see Section 4.1) would stop executing on high resolution images due to high RAM usage in one batch. Therefore patches are batched to fit RAM (batch size for prediction is adjustable in the config file).

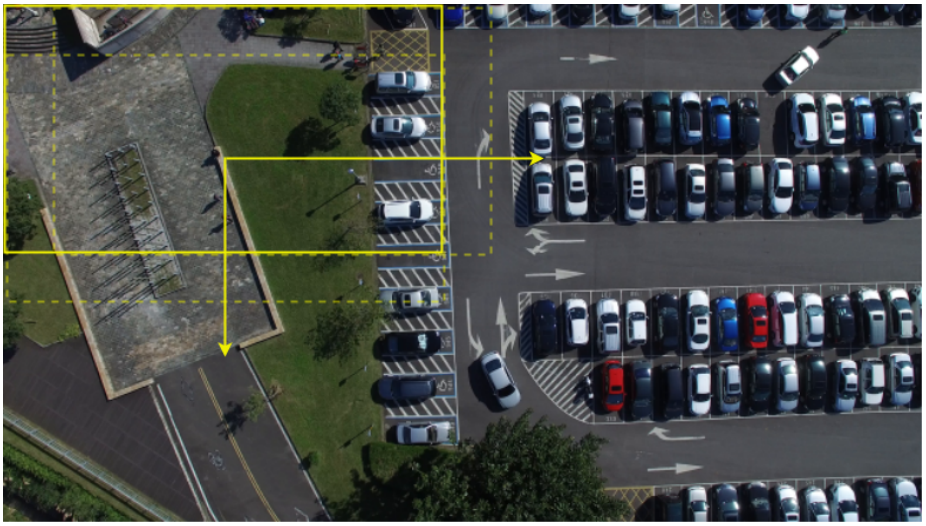


Figure 4.4: Dense patch extraction procedure for prediction phase. Patches are extracted from the top left corner towards bottom right corner. More details are mentioned at the end of the Subsection 3.2.3. The input image is from CARPK dataset.

The network returns an array of predicted patches. As mentioned in the Subsection 3.2.3 final density map is created by normalization of each position in resulting density map.

4.4 Existing Vehicle Datasets

This section contains datasets that are evaluated on the proposed model. Metrics for each dataset are mentioned in this chapter but are described in the next chapter. State-of-the-art results along with the results of the proposed model are also located in the next chapter. All datasets listed here use one of the three annotation methods.

First method is the dot annotation (also mentioned in the Subsection 3.2.3). Here a point within a dot image represents a position of a vehicle. The point has (x, y) coordinate and has a pixel value 255 (max brightness value in greyscale images). The rest of dot image is black (has value 0 brightness value).

The second method is bounding box technique. Each vehicle gets upper left and bottom right (x, y) coordinates representing the bounding box of an object (a vehicle in this context). For the purpose of my regression approach this bounding box annotation is converted to the first aforementioned method by taking the centre of the bounding box.

The last one is 'scribble' method (used in the Car Park Calibration dataset). Each vehicle is denoted by a scribbled line. Conversion to dot annotation is taking one point of the scribbled line to represent a vehicle (in this thesis a midpoint is taken).

4.4.1 TRANCOS

A Spain car dataset [12]. TRANCOS is the acronym for TRaffic ANd COngestionS. As the name suggests the dataset focuses on traffic congestions. The dataset is made by capturing traffic via selected real surveillance cameras during three week period. The dataset contains more than 1,200 images and 46,700 annotated vehicles. These images are split in 3 sets as follows:

- Training - 403 images
- Validation - 420 images
- Test - 421 images

TRANCOS uses dot annotation technique to represent each vehicle's position. Since annotations are partial (not every vehicle is annotated), they used Region of Interest (ROI). Binary masks are provided to make visible only the annotated section of image (leaving non-ROI regions black). To train a model we are given a choice between 2 methods: either to train solely on training set or to train on both training and validation sets (or as they call it - 'trainval' set). Figure 4.5 shows some sample images from TRANCOS.

Figure 4.5: Sample images from TRANCOS dataset.

TRANCOS uses its own metric to measure network's accuracy – the $GAME(L)$ metric. This metric is described in the Section 5.1. TRANCOS has MATLAB code for calculating the metric (publicly available along with the dataset on their web page[5]). I have rewritten this code from MATLAB[6] to Python to get the $GAME(L)$ evaluation of my model which is written in Keras [20].

[5] http://agamenon.tsc.uah.es/Personales/rlopez/data/trancos/TRANCOS_v3.tar.gz
[6] https://www.mathworks.com/products/matlab.html

4.4.2 PUCPR+

This dataset is a subset of the PKLot dataset [1]. As per description the captures are taken at Pontifical Catholic University of Parana (PUCPR), located in Curitiba, Brazil. Images are taken from the 10th floor of the administration building of the PUCPR. These captures are taken during various weather conditions such as sunny, overcast and rainy. Figure 4.6 shows some sample images from PUCPR+.

Figure 4.6: Sample images from PUCPR+ dataset.

Originally, the PUCPR dataset (note the missing '+' sign) is annotated only partly (100/331 parking spaces in a single image). Meng-Ru et al. [14] localized and completed missing annotations and called it PUCPR+. In total of only 125 images with nearly 17,000 cars, just 25 images are meant for model testing. Such a low number of images makes it a challenging task to train a network. PUCPR+ uses `MAE` and `RMSE` metrics. Annotations are made by bounding box technique. The dataset is available on the github[7].

4.4.3 CARPK

Meng-Ru et al. proposed their own dataset and called it CARPK [14]. CARPK provides large-scale images taken from the drone point of view. The drone took images from four various parking lots. There are nearly 1,500 images with approximately 90,000 cars in total. Figure 4.7 shows some sample images from CARPK.

Drone addresses the problem of a bias created by a fixed camera scene, where the point of view is constant. CARPK dataset uses the same metrics as PUCPR+ (`MAE` and `RMSE`). Annotations are also the same as in PUCPR+ case (bounding box technique). As well as the PUCPR+ dataset this one is also available on the github[7].

[7] https://lafi.github.io/LPN

Figure 4.7: Sample images from CARPK dataset.

4.4.4 Car Park Calibration

A dataset from the research group GRAPH@FIT[8]. The research group has its headquarters at the Faculty of Information Technology, Brno University of Technology in the Czech Republic.

The dataset provided contains over 2,800 frames taken in 2 various parking lots in Brno, Czech Republic. Each frame corresponds to one minute elapsed. Frames have resolution of either $1,920 \times 1,080$ or $1,280 \times 800$. Vehicles are annotated by aforementioned 'scribble' method.

For video demonstration purposes only a part of this dataset is used. I created train and test sets using only $1,920 \times 1,080$ resolution frames. From these frames 50 are reserved for creating a video sequence. 80% of remaining frames are reserved for training and 20% for testing. In data science it is a common practice to perform a 80/20 split for training and testing phase [41]. Resulting demo dataset has nearly 900 frames. Figure 4.8 shows some sample frames from this demo dataset.

[8] https://medusa.fit.vutbr.cz/traffic

Figure 4.8: Sample frames from the created demo Car Park Calibration dataset.

Chapter 5

Experimental Results and Evaluation

This chapter describes metrics that are used during the model evaluation on given test sets. Next, it defines how error rate (accuracy) is measured and what does each sample in metric computation represent. State-of-the-art results along with training times, last epoch saved and GPU used are provided in form of tabular data.

5.1 Used Metrics

Probably the most straightforward metric is `MAE` (Mean Absolute Error). It calculates the absolute differences between expected and predicted vehicle counts and averages them at the end. Equation 5.1 shows general formula to compute `MAE`.

$$\text{MAE} = \frac{1}{N} \cdot \sum_{i=1}^{N} |y - \hat{y}|\,, \tag{5.1}$$

where:

- N: is the total number of data samples
- y: is the expected value
- $\hat{y}$: is the predicted value

One of the frequently used metric is `RMSE` (Root Mean Square Error). Value of this metric is either 0 (a perfect fit) or a positive number. Equation 5.2 shows general formula to compute `RMSE`.

$$\text{RMSE} = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (y - \hat{y})^2}\,, \tag{5.2}$$

where:

- N: is the total number of data samples
- y: is the expected value

- $\hat{y}$: is the predicted value

The last used metric during the evaluation is $GAME(L)$ [12]. $GAME(L)$ stands for Grid Average Mean Error and is required to be used when reporting results of one's model accuracy on TRANCOS dataset. This metric takes into consideration vehicle localization error (unlike `MAE` or `RMSE`). The $GAME(L)$ metric is formulated in the Equation 5.3.

$$\text{GAME}(L) = \frac{1}{N} \cdot \sum_{n=1}^{N} \left(\sum_{l=1}^{4^L} |e_n^l - gt_n^l| \right) , \tag{5.3}$$

where:

- N: is the total number of test images
- L: is the total levels number (max 4-regions exponent)
- e_n^l: is the estimated count within region l of image n
- gt_n^l: is the ground truth count within region l of image n

The higher the L number the more strict /precise localization error estimation is. I made a toy example for demonstration purposes (see Fig. 5.1).

5.2 Error Rate Measurements

The proposed architecture is trained and tested on datasets described in the Section 4.4. I will compare official state-of-the-art results with my model. All official state-of-the-art results are pulled from the respective dataset web page. Tables with comparisons are truncated to show only Top 5 results. Values measured depict error rates between expected and predicted vehicles counts measured on the corresponding test sets (the lower the value the more precise the prediction is). Tables: 5.2, 5.3 and 5.4 show results achieved. All following measurements have the following parameters:

- Base learning rate: $1e-5$
- Epochs limit (max epochs): 100
- Validation split parameter: 0.2

Training phase was carried out on the NVIDIA Tesla P4 GPU [33] and NVIDIA Tesla P100 GPU [32]. NVIDIA P4 offers: 5.5 TeraFLOPS of performance for 32 bit floating point with single precision, 8 GB GPU RAM and 192 GB/s memory bandwidth. On the other hand NVIDIA P100 offers: 10.6 TeraFLOPS of performance for 32 bit floating point with single precision, 16 GB GPU RAM and 732 GB/s memory bandwidth. I have kept track of training times, last saved epoch and GPU used for each measured dataset. Both GPUs were more than sufficient during training and evaluation of the model (see Table 5.1). Values in this table correspond to data measured in Tables: 5.2, 5.3, 5.4 and demo Car Park Calibration results. Some of the network predictions are shown in Figures: 5.2, 5.3, 5.4 and 5.5.

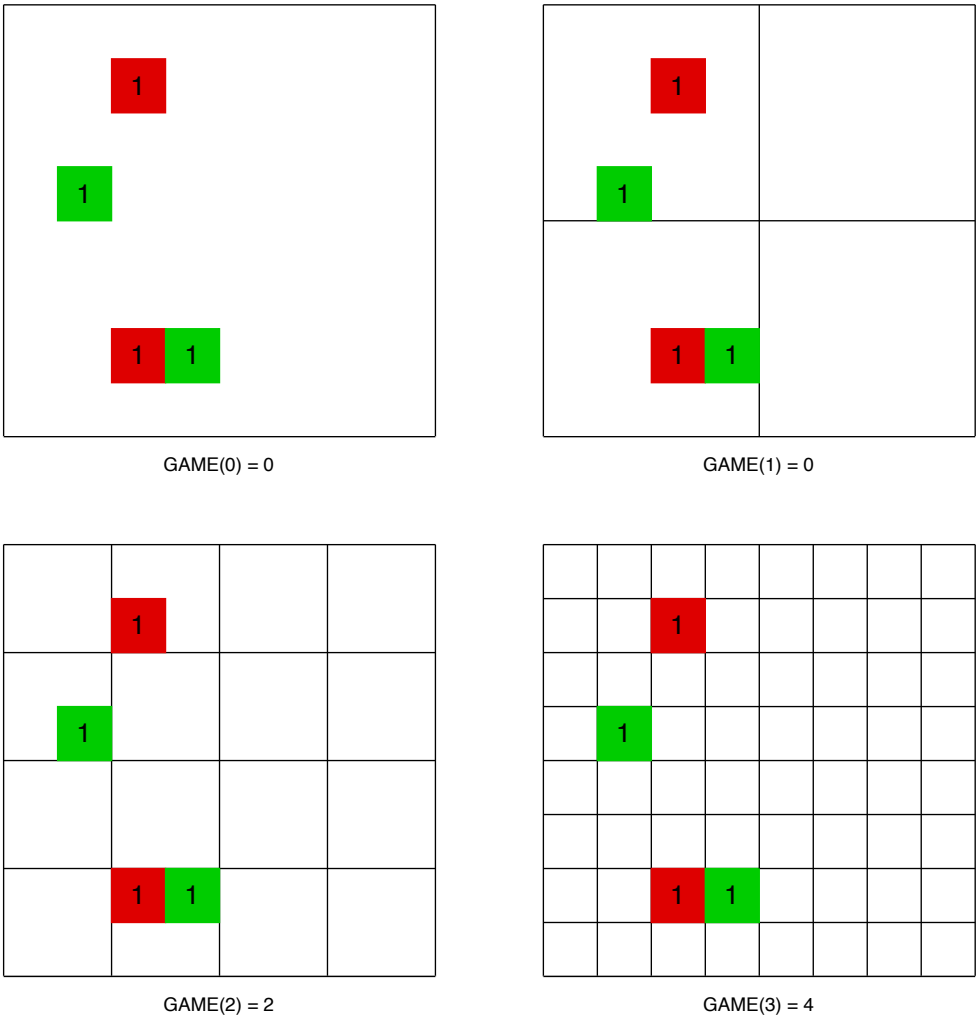


Figure 5.1: GAME(L) metric visualization. Red boxes represent expected vehicles within a region, whilst green ones represent vehicles prediction within a region. The higher the GAME(L) level is the more precise error localization is. GAME(L) with level 0 basically represents MAE metric.

Measured values for the demo Car Park Calibration dataset are:

- RMSE: 3.74
- GAME(0)/MAE: 2.17
- GAME(1): 3.02
- GAME(2): 4.46
- GAME(3): 7.03

Table 5.1: Proposed model training times for 100 elapsed epochs along with last epoch saved and GPU used.

Dataset	Time [hrs]	Epoch	GPU
TRANCOS	1.17	73	P4
PUCPR+	0.33	99	P4
CARPK	3.5	80	P4
Car Park Calibration	4.5	46	P100

Table 5.2: Evaluation of state-of-the-art methods on dataset TRANCOS. Lower number is better.

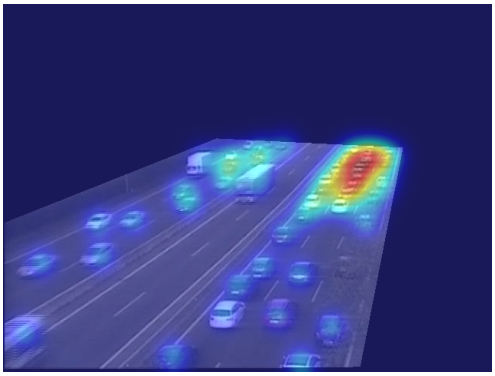
Method	GAME(0)	GAME(1)	GAME(2)	GAME(3)
IbPRIA 2015 (HOG-2) [12]	13.29	18.05	23.65	28.41
IbPRIA 2015 (Fiaschi+RGB Norm+Filters) [12]	17.68	19.97	23.54	25.84
IbPRIA 2015 (Lempitsky+SIFT) [12]	13.76	16.72	20.72	24.36
Hydra CNN (ECCV2016) [34]	10.99	13.75	16.69	19.32
proposed model	**3.40**	**4.79**	**6.29**	**8.35**

Table 5.3: Evaluation of state-of-the-art methods on dataset PUCPR+. Lower number is better.

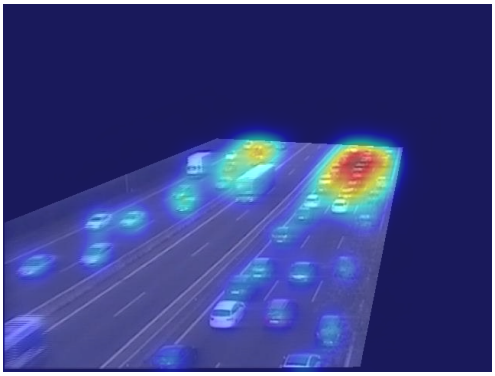
Method	MAE	RMSE
YOLO [38]	156.00	200.42
Faster R-CNN [39]	111.40	149.35
One-Look Regression [29]	21.88	36.73
Layout Proposal Network [14]	22.76	34.46
proposed model	**5.58**	**6.99**

Table 5.4: Evaluation of state-of-the-art methods on dataset CARPK. Lower number is better.

Method	MAE	RMSE
YOLO [38]	48.89	57.55
Faster R-CNN [39]	47.45	57.39
One-Look Regression [29]	59.46	66.84
Layout Proposal Network [14]	23.80	36.79
proposed model	**15.97**	**18.33**

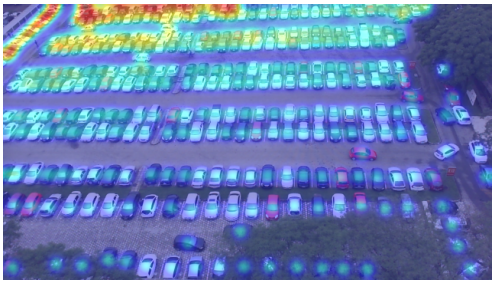


(a) Ground Truth: 44.03

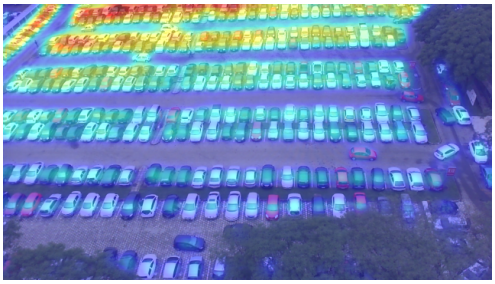


(b) Prediction: 41.89

Figure 5.2: TRANCOS prediction.

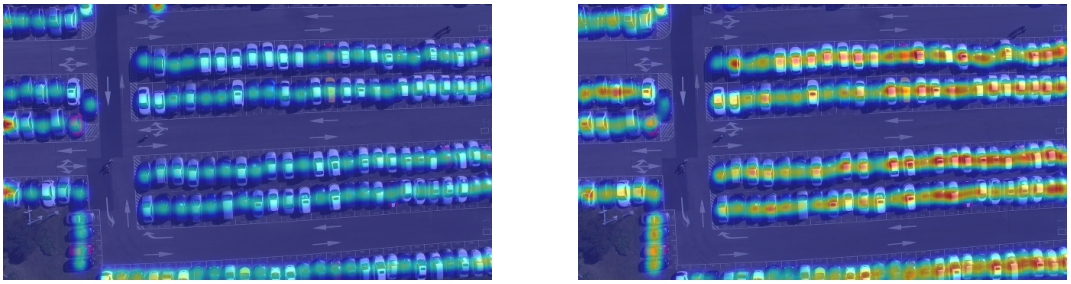


(a) Ground Truth: 328.77



(b) Prediction: 328.10

Figure 5.3: PUCPR+ prediction.



(a) Ground Truth: 152.41 (b) Prediction: 143.76

Figure 5.4: CARPK prediction.

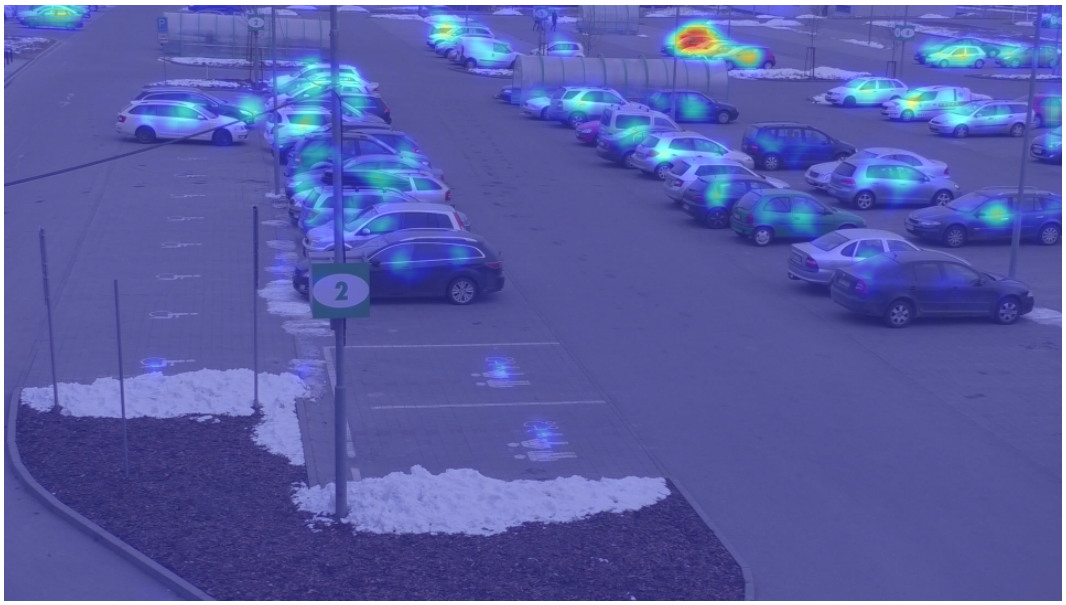


Figure 5.5: Demo Car Park Calibration prediction: 53.99 vehicles.

Chapter 6

Conclusion

This thesis addresses the issue of counting vehicles in image and video. Both traditional approaches and convolutional neural network (CNN for short) approaches to count vehicle instances are discussed with the focus on the latter. There are many principles to perform to count object instances in CNNs. One of them is Counting by Regression principle which is also the aim of this thesis. Some other principles are briefly mentioned along with the related theory background of CNNs. A new CNN architecture which is inspired by current state-of-the-art architectures is proposed and evaluated.

The proposed architecture is implemented in a popular Keras framework. The accuracy of the model was preferred over its speed during the development. With that said, dense patch extraction strategy is used for predictions instead of quadrant patch extraction. The model makes use of transfer learning technique, which significantly reduced training times (for instance, training on the TRANCOS dataset took only 1.17 hours). Downloaded pre-trained weights are a part of the VGG-16 front-end in the proposed architecture. This front-end is backed by a multi-column structure with dilated convolutions. This setup experimentally improved current state-of-the-art results on the tested datasets.

The proposed model further improves the accuracy of state-of-the-art results on all tested vehicle datasets (TRANCOS, PUCPR+, CARPK). For instance on the very small PUCPR+ dataset the Root Mean Square Error between expected and predicted vehicle counts was reduced from 34.46 to 6.99 vehicles (measured on the test set). All measured values are a result of training with only 100 epochs limit threshold. In my opinion transfer learning has its non-negligible impact in this matter.

The accuracy could be further improved if the model was trained with a higher epochs limit. Dense patch extraction yields more precise results than quadrant patch extraction. However, dense extraction is more memory and time demanding than the former. This overhead could be reduced by finding an optimal stride value or by finding some advanced strategy. The aforementioned strategy is also used in video prediction. Here video represents a frame sequence encoded into a video file. Real-time detection is not considered here and might as well be considered for a further research. All these arguments showed that there is still space for improvements and a possible further research in Counting by Regression approach.

Bibliography

[1] Almeida, P., Oliveira, L. Soares de, Jr, A., Jr, E. and Koerich, A. PKLot - A Robust Dataset for Parking Lot Classification. *Expert Systems with Applications*. february 2015, vol. 42. DOI: 10.1016/j.eswa.2015.02.009.

[2] Amirgholipour, S., He, X., Jia, W., Wang, D. and Liu, L. *PDANet: Pyramid Density-aware Attention Net for Accurate Crowd Counting*. 2020.

[3] Brownlee, J. *Machine Learning Mastery - A Gentle Introduction to Pooling Layers for Convolutional Neural Networks* [online]. July 2019 [cit. 2020-05-10]. Available at: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks.

[4] Cireşan, D., Meier, U. and Schmidhuber, J. *Multi-column Deep Neural Networks for Image Classification*. 2012.

[5] DeepAI. *Convolutional Neural Networks Definition* [online]. [cit. 2020-05-10]. Available at: https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network.

[6] Deng, J., Dong, W., Socher, R., Li, L., Kai Li et al. ImageNet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, p. 248–255.

[7] Goodnight, E. Z. *How-To Geek - What's the Difference Between JPG, PNG, and GIF?* [online]. July 2017 [cit. 2020-05-10]. Available at: https://www.howtogeek.com/howto/30941/whats-the-difference-between-jpg-png-and-gif.

[8] Google. *Colaboratory - Welcome To Colaboratory* [online]. [cit. 2020-05-10]. Available at: https://colab.research.google.com/notebooks/intro.ipynb.

[9] Google. *Colaboratory – Google - Frequently Asked Questions* [online]. [cit. 2020-05-10]. Available at: https://research.google.com/colaboratory/faq.html.

[10] Google. *Google Cloud - Cloud TPU - Cloud Tensor Processing Units (TPUs)* [online]. [cit. 2020-05-10]. Available at: https://cloud.google.com/tpu/docs/tpus.

[11] Groups, G. *Keras-users–Skupiny Google* [online]. [cit. 2020-05-10]. Available at: https://groups.google.com/forum/#!forum/keras-users.

[12] Guerrero Gómez Olmedo, R., Torre Jiménez, B., López Sastre, R., Maldonado Bascón, S. and Oñoro, D. Extremely Overlapping Vehicle Counting. In:. June 2015. DOI: 10.1007/978-3-319-19390-8_48.

[13] He, K., Zhang, X., Ren, S. and Sun, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2015, vol. 37, no. 9, p. 1904–1916.

[14] Hsieh, M.-R., Lin, Y.-L. and Hsu, W. H. Drone-based Object Counting by Spatially Regularized Regional Proposal Networks. In: IEEE. *The IEEE International Conference on Computer Vision (ICCV)*. 2017.

[15] JSON. *Introducing JSON* [online]. [cit. 2020-05-10]. Available at: https://www.json.org/json-en.html.

[16] Keras. *Keras Documentation - CIFAR-10 CNN* [online]. [cit. 2020-05-10]. Available at: https://keras.io/examples/cifar10_cnn.

[17] Keras. *Keras Documentation - Convolutional Layers* [online]. [cit. 2020-05-10]. Available at: https://keras.io/layers/convolutional.

[18] Keras. *Keras Documentation - Convolutional LSTM* [online]. [cit. 2020-05-10]. Available at: https://keras.io/examples/conv_lstm.

[19] Keras. *Keras Documentation - FAQ - What are my options for saving models?* [online]. [cit. 2020-05-10]. Available at: https://keras.io/getting_started/faq/#what-are-my-options-for-saving-models.

[20] Keras. *Keras: the Python deep learning API* [online]. [cit. 2020-05-19]. Available at: https://keras.io.

[21] Klein, L. A., Gibson, D. R. P. and Mills, M. K. *Traffic detector handbook*. 3rd ed. Federal Highway Administration, Turner-Fairbank Highway Research Center, Oct 2006.

[22] Lempitsky, V. and Zisserman, A. Learning To Count Objects in Images. In:. January 2010, p. 1324–1332.

[23] Liu, W., Salzmann, M. and Fua, P. Context-Aware Crowd Counting. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[24] Liu, W., Salzmann, M. and Fua, P. *GitHub - weizheliu/Context-Aware-Crowd-Counting - Context-Aware-Crowd-Counting/test.py at master* [online]. July 2019 [cit. 2020-05-10]. Available at: https://github.com/weizheliu/Context-Aware-Crowd-Counting/blob/master/test.py.

[25] Loy, C. C., Chen, K., Gong, S. and Xiang, T. Crowd Counting and Profiling: Methodology and Evaluation. In:. October 2013, vol. 11. DOI: 10.1007/978-1-4614-8483-7_14.

[26] Mallick, S. *Learn OpenCV (C++ / Python) - Why does OpenCV use BGR color format ?* [online]. September 2015 [cit. 2020-05-10]. Available at: https://www.learnopencv.com/why-does-opencv-use-bgr-color-format.

[27] mc.ai. *10. Introduction to Deep Learning with Computer Vision - Types of Convolutions & Atrous Convolutions* [online]. February 2020 [cit. 2020-05-10]. Available at: https://mc.ai/10-introduction-to-deep-learning-with-computer-%vision-types-of-convolutions-atrous-convolutions-2.

[28] MissingLink.ai. *Fully Connected Layers in Convolutional Neural Networks: The Complete Guide* [online]. [cit. 2020-05-10]. Available at: https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide.

[29] Mundhenk, T. N., Konjevod, G., Sakla, W. A. and Boakye, K. A Large Contextual Dataset for Classification, Detection and Counting of Cars with Deep Learning. In: Leibe, B., Matas, J., Sebe, N. and Welling, M., ed. *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, p. 785–800. ISBN 978-3-319-46487-9.

[30] Nvidia. *NVIDIA Deep Learning SDK Documentation - cuDNN Installation Guide - Prerequisites* [online]. [cit. 2020-05-10]. Available at: https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html#prerequisites.

[31] Nvidia. *NVIDIA Developer - NVIDIA cuDNN* [online]. [cit. 2020-05-10]. Available at: https://developer.nvidia.com/cudnn.

[32] NVIDIA. *Nvidia Tesla P100 Datasheet* [online]. [cit. 2020-05-20]. Available at: https://images.nvidia.com/content/tesla/pdf/nvidia-tesla-p100-datasheet.pdf.

[33] NVIDIA. *Nvidia Tesla P4 Datasheet* [online]. [cit. 2020-05-20]. Available at: https://images.nvidia.com/content/pdf/tesla/184457-Tesla-P4-Datasheet-NV-Final-Letter-Web.pdf.

[34] Oñoro, D. and López Sastre, R. Towards Perspective-Free Object Counting with Deep Learning. In:. October 2016, vol. 9911. DOI: 10.1007/978-3-319-46478-7_38.

[35] Piqsels. *Adult german shepherd, German Shepherd, shepherd, almind, denmark, dog, pets, one animal, CC0, public domain, royalty free* [online]. [cit. 2020-05-10]. Available at: https://www.piqsels.com/en/public-domain-photo-sijkl.

[36] Principles of Deep Learning. *A tutorial on global average pooling* [online]. [cit. 2020-05-10]. Available at: https://principlesofdeeplearning.com/index.php/a-tutorial-on-global-average-pooling.

[37] Rabaud, V. and Belongie, S. Counting Crowded Moving Objects. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. 2006, vol. 1, p. 705–711.

[38] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 779–788.

[39] Ren, S., He, K., Girshick, R. and Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M. and Garnett, R., ed. *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, p. 91–99. Available at: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf.

[40] Simonyan, K. and Zisserman, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014.

[41] Srinidhi, S. *How to split your dataset to train and test datasets using SciKit Learn* [online]. July 2018 [cit. 2020-05-20]. Available at: https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d.

[42] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. june 2014, vol. 15, p. 1929–1958.

[43] The HDF Group. *The HDF5®Library & File Format* [online]. [cit. 2020-05-10]. Available at: https://www.hdfgroup.org/solutions/hdf5.

[44] Walia, A. S. *Activation functions and it's types-Which is better?* [online]. May 2017 [cit. 2020-05-10]. Available at: https://www.linkedin.com/pulse/activation-functions-its-types-which-better-anish-singh-walia.