FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Crowd Counting Methods - Robustness and Applicability Limits
Student:	Bc. Martin Vadlejch
Supervisor:	Ing. Filip Naiser
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

The current state-of-the-art methods for crowd counting in images tend to have a strong bias towards images with a high number of people. This leads to the count overestimation on images from real applications where sparsity naturally occurs (e.g. during non-peak hours).
Additionally, current datasets [1, 2, 3, 4] poorly reflect other challenges like rain (umbrellas), snowing, poor illumination, and a high viewpoint variance.
The student will review crowd counting literature. Then he will gather a small dataset from different public domain webcams to capture the abovementioned conditions. Using this dataset, he will evaluate state-of-the-art methods.
He will try to design and implement improvements for found issues. He will evaluate his implementations on both standard and his own datasets.

References

[1] Y. Zhang, D. Zhou, S. Chen, S. Gao, Yi Ma - Single-Image Crowd Counting via Multi-Column Convolutional Neural Network - IEEE 2016
[2] H. Idrees, M. Tayyab, K. Athrey, D. Zhang, S. Al-Maddeed, N. Rajpoot, M. Shah - Composition Loss for Counting, Density Map Estimation and Localization in Dense Crowds - ECCV 2018
[3] H. Idrees, I. Saleemi, C. Seibert, M. Shah - Multi-Source Multi-Scale Counting in Extremely Dense Crowd Images - CVPR 2013
[4] Qi Wang, Junyu Gao, Wei Lin, Xuelong Li - NWPU-Crowd: A Large-Scale Benchmark for Crowd Counting and Localization - IEEE 2020

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 3 January 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Crowd Counting Methods – Robustness and Applicability Limits

Bc. Martin Vadlejch

Department of Applied Mathematics
Supervisor: Ing. Filip Naiser

May 3, 2023

Acknowledgements

I wish to express my deepest gratitude to my family, friends, and girlfriend, for they have always supported and encouraged me and helped me achieve my goals. I also wish to sincerely thank my supervisor Ing. Filip Naiser, for his helpful insights, expertise, and kindness.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 3, 2023 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Vadlejch, Martin. *Crowd Counting Methods – Robustness and Applicability Limits*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstrakt

Technologie sčítání davů lidí má potenciál být pro společnost cenným nástrojem v mnoha ohledech, například pro optimalizaci řízení davů a zvýšení veřejné bezpečnosti. Automatizované metody sčítání davu jsou nezbytné vzhledem k rozsahu tohoto úkolu a skutečnosti, že ruční sčítání osob v přeplněných oblastech může být neproveditelné nebo náchylné k chybám. Automatizované sčítání davů navíc umožňuje sledovat pohyb a hustotu davu v reálném čase, což může být zásadní při mimořádných událostech nebo rozsáhlých akcích.

Navzdory pokroku v metodách automatického sčítání davů však jejich robustnost zůstává problémem, protože chybovost není konzistentně úměrná skutečnému počtu osob na snímku, a často se vyskytují případy výrazného nadhodnocení i podhodnocení. Tato nekonzistence je v současnosti činí nespolehlivými pro praktické aplikace. Vzhledem k potenciálnímu využití technologie automatického sčítání davů a možným výhodám pro společnost je tato oblast výzkumu a vývoje však velmi slibná.

V této práci vyhodnocujeme přesnost a spolehlivost současných nejmodernějších metod v náročných podmínkách, které testujeme na našich specializovaných souborech dat, a navrhujeme metodiku trénování pro vytvoření robustnějších modelů. Zjistili jsme, že hlubší enkodéry silně zvyšují robustnost těchto modelů. Zjistili jsme také, že globální self-attention mechanismus zvyšuje přesnost sčítání, zejména ve zhoršených světelných podmínkách, což jsme vypozorovali na základě 72 různých moderních metod. Navrhli jsme také upravený inferenční postup, který umožňuje sčítání davů na snímcích s vysokým rozlišením s běžným hardwarem a výrazně zlepšuje přesnost s omezenou pamětí.

Klíčová slova analýza chyb, detekce objektů, hluboké učení, neuronové sítě, počítačové vidění, robustnost, sčítání davů, sčítání lidí

Abstract

Crowd counting technology has the potential to be a valuable tool for society in a variety of ways, such as optimizing crowd management and improving public safety. Automated crowd-counting methods are essential due to the sheer scale of the task and the fact that manually counting individuals in crowded areas can be unfeasible or error-prone. Moreover, automated crowd counting can enable real-time crowd movement and density monitoring, which can be crucial in emergencies or large-scale events.

However, despite advances in crowd-counting methods, their robustness remains a challenge as the error rate is not consistently proportional to the actual count of people in an image, with instances of both significant over- and underestimation occurring frequently. This inconsistency renders them unreliable for practical applications. Nevertheless, with further development and refinement, crowd-counting technology has the potential to provide essential benefits to society, making it a promising area of research and development.

In this work, we evaluate the current state-of-the-art methods' accuracy and reliability in challenging conditions, which we test on our specialized datasets, and devise a training methodology to produce more robust models. We have found that deeper encoders are critical for improving the model's robustness. We also found the global self-attention mechanism to benefit counting accuracy, particularly in low-light scenarios, which we have observed in a corpus of 72 different state-of-the-art methods. We also devise a patched inference pipeline that enables crowd counting in high-resolution images with conventional hardware and drastically improves accuracy with limited memory.

Keywords computer vision, crowd counting, deep learning, error analysis, neural networks, object detection, people counting, robustness

Contents

Introduction **1**

1 Machine Learning **3**
1.1 Introduction . . . 3
1.2 Machine Learning Fundamentals . . . 4
1.2.1 Data Splitting . . . 4
1.2.1.1 Data Split Ratio . . . 5
1.2.1.2 K-Fold Cross-Validation . . . 5
1.2.2 Data Preprocessing . . . 6
1.2.3 Loss Function, Batch, Epoch . . . 7
1.2.3.1 Classification . . . 7
1.2.3.2 Regression . . . 8
1.2.4 Model Selection and Evaluation Metrics . . . 8
1.2.4.1 Bias-Variance Tradeoff . . . 8
1.2.4.2 Mean Average Error . . . 10
1.2.4.3 Accuracy . . . 10
1.3 Deep Learning . . . 11
1.3.1 Artificial Neuron . . . 11
1.3.2 Multilayer Perceptron . . . 12

2 Computer Vision **15**
2.1 Brief History of Computer Vision . . . 15
2.2 Deep Learning Methods . . . 16
2.2.1 Convolutional Neural Networks . . . 16
2.2.1.1 Convolution . . . 17
2.2.1.2 Pooling . . . 18
2.2.1.3 Tensors . . . 19
2.2.1.4 Architectural Designs . . . 20
2.2.2 Transformer Architectures for Computer Vision . . . 21

2.2.2.1 Attention . . . 21
2.2.2.2 Multi-Head Attention . . . 22
2.2.2.3 Types of Attention Layers . . . 23
2.2.2.4 Positional Encoding . . . 23
2.2.2.5 Convolutional Transformer Architecture . . . 24

3 Crowd Counting **25**
3.0.1 Crowd Counting Datasets . . . 26
3.0.1.1 WorldExpo'10 . . . 27
3.0.1.2 UCF_CC_50 . . . 27
3.0.1.3 ShanghaiTech . . . 27
3.0.1.4 UCF_QNRF . . . 28
3.0.1.5 GCC . . . 28
3.0.1.6 NWPU-Crowd . . . 28
3.0.2 Density-Based Methods . . . 29
3.0.3 Point-Based Methods . . . 31

4 Experiments and Results **33**
4.1 Selected Models . . . 34
4.1.1 C3 Framework Models . . . 34
4.1.1.1 AlexNet . . . 34
4.1.1.2 VGG Models . . . 34
4.1.1.3 ResNet Models . . . 34
4.1.1.4 CSRNet . . . 35
4.1.1.5 SANet . . . 35
4.1.2 SASNet . . . 35
4.1.3 P2PNet . . . 35
4.1.4 Model Summary . . . 36
4.2 Negative Samples and False Positives . . . 39
4.2.1 Negative Samples . . . 40
4.2.2 High Frequency and Blob-Like Objects . . . 42
4.2.3 Impact of the Training Data on the Model Robustness . . . 44
4.2.4 Note on the Importance of Image Resolution . . . 45
4.3 Weather Conditions and Luminance . . . 47
4.4 Analysis of the Trends in Crowd Counting Research . . . 50

Discussion **53**

Conclusion **55**

Bibliography **59**

A List of Acronyms **67**

B Contents of the enclosed media **69**

List of Figures

1.1 Illustration of Train/Test Split 5
1.2 Illustration of K-Fold Cross-Validation 6
1.3 Bias-Variance Tradeoff 9
1.4 Single Layer Perceptron 11
1.5 Multilayer Perceptron 12
1.6 Dropout Regularization 13

2.1 Kernel Convolution 17
2.2 Padded Convolution 17
2.3 Pooling Layers 18
2.4 Input Tensor Illustration 19
2.5 Inception Block 20
2.6 ResNet Residual Block 20
2.7 Multi-Head Attention 22
2.8 Transformer Encoder-Decoder Architecture 24

3.1 WorldExpo'10 Dataset Examples 27
3.2 UCF_CC_50 Dataset Examples 27
3.3 ShanghaiTech Dataset Examples 27
3.4 UCF_QNRF Dataset Examples 28
3.5 GCC Synthetic Dataset Examples 28
3.6 NWPU-Crowd Dataset Examples 29
3.7 Gaussian Crowd Density Map 29
3.8 Multi-Column Convolutional Neural Network Architecture 30
3.9 Scale-Adaptive Selection Network Architecture 31
3.10 Prediction-GT Point Matching Example 31
3.11 The P2PNet Architecture 32
3.12 The CLTR Architecture 32

4.1 Comparison of SASNet_A and SASNet_B 37
4.2 Limited Detection Scale Example 38

4.3 Example Images from the empty_places Dataset 39
4.4 Example Images from the hard_images Dataset 39
4.5 Examples of Significant False Positives in the empty_places Dataset 41
4.6 Examples of Significant False Positives in the hard_images Dataset 43
4.7 Impact of Resolution on the Performance of SASNet 46
4.8 Impact of Resolution on the Performance of P2PNet (Detail) . . . 46
4.9 Examples of Weather Augmentation Methods 48
4.10 Fog-Introduced False Positives 48
4.11 Examples of Image Pairs From The DroneRGBT Dataset 49
4.12 Achieved Test Metrics in the NWPU-Crowd Benchmark per Year of Publication 50
4.13 Achieved Test Metrics in the NWPU-Crowd Benchmark per Level of Illumination 51
4.14 Achieved Test Metrics in the NWPU-Crowd Benchmark per Crowd Density Level 52
4.15 Comparison of Density-Based and Point-Based Methods in the NWPU-Crowd Benchmark 52
4.16 MOG2 Background Subtractor 53
4.17 Examples of Scattered Occlusions in Crowds 54

List of Tables

3.1 Crowd-Counting Dataset Comparison 26

4.1 Summary of Selected Models 36
4.2 Benchmark of Selected Models 37
4.3 False Positives in the empty_places Dataset 40
4.4 False Positives in the hard_images Dataset 42
4.5 False Positives in the empty_places and hard_images Datasets After Re-Training 44
4.6 Counting Accuracy of the Re-Trained Models 44
4.7 Total People Detected with Different Image Resolution 47
4.8 Model Accuracy in Challenging Weather Conditions 49

Introduction

Crowd counting is a task in computer vision that seeks to accurately predict the number of people in crowd pictures. Such technology has many potential applications, such as traffic management, identifying congestion-prone locations, re-routing traffic, and managing traffic flows in space-restricted areas, such as subways and stadiums, to ensure public safety and prevent overcrowding [1]. When combined with crowd localization techniques, these methods can be used in retail analytics to monitor foot traffic and track customer behavior to optimize store layouts and improve customer experience. Only with automated methods can accurate crowd counting be feasible in real-time, enabling the authorities to utilize this knowledge to enhance public safety and quickly respond to potential overcrowding in the event of an emergency. Deep learning approaches and neural networks currently dominate this field, as they have shown superior performance and generalization when learning from large datasets compared to traditional computer vision methods [2, 3].

However, the reliability and robustness currently hinder the real-world usability of automated methods. Many factors are challenging to overcome, such as significant viewpoint variance, varying crowd appearance, occlusions, crowd sparsity, and different illumination and weather conditions. Considering the abovementioned challenges, designing a universally applicable, reliable, and robust method is challenging. It needs to be discussed more in crowd-counting literature, which rarely mentions potential problems for real-life applications where occlusions, crowd sparsity, and poor illumination are common.

Currently, state-of-the-art methods are often benchmarked on dense crowd datasets with little to no sparsity and good illumination. However, minimizing count errors in pictures taken in close to ideal conditions rarely results in reliable and robust detectors. They either fail to detect in poor illumination conditions or hallucinate in areas with high-frequency noise. One possible cause could be the lack of sparse crowds in training data, resulting in insufficient penalization for the crowd under or overestimation.

Furthermore, current datasets do not account for crowd occlusions caused by foliage, umbrellas, or picket signs, which are often present in real-world conditions and impede the resulting accuracy and reliability of the system. Designing a system to detect occlusions and count crowds simultaneously could prove more reliable in real-world scenarios [4].

This work aims to evaluate state-of-the-art approaches to crowd counting on both commonly used datasets and our specialized datasets. We design the latter to evaluate poor weather and illumination conditions and minimize the number of false positives. We aim to improve robustness by devising ways to utilize hard negative mining and improve the training data. The contribution of this work shall provide readers with insights into designing robust crowd counters for real-life applications.

Chapter 1

Machine Learning

1.1 Introduction

Machine learning is a field in artificial intelligence focusing on developing algorithms designed to tackle challenging problems by automatically recognizing patterns, understanding data, and making predictions for new, unseen data. They do so almost universally, without the need for explicit programming and the development of specialized algorithms. They often adapt to many different domains well, making machine learning algorithms a powerful tool for solving problems unfeasible to solve otherwise [5, 6].

Several notable breakthroughs have been achieved in machine learning in recent years. For example, the AlphaFold 2 algorithm successfully tackled the problem of protein folding [7], the GPT-3 model demonstrated significant advancements in natural language processing [8], and the EfficientZero algorithm achieved super-human performance in mastering Atari games [9].

All those feats are not short of impressive, but all the achievements of machine learning models depend heavily on the availability of high-quality training data in sufficiently large quantities. Often, machine learning algorithms fail to deliver expected results or behave unpredictably if significant biases are present in the training data or if the collecting methodology is flawed [10, 11]. Widely discussed topics were human-like racist biases when using machine learning algorithms for criminal risk assessment [12], mistaking images of black people for gorillas [13], or a chatbot that picked up the sexist language and neo-nazi ideologies [14].

When collecting a training set for such an algorithm, special care must be taken to ensure it works as expected, with as few biases or imbalances as possible. Applying this thought to our crowd counting problem, if we only teach our neural network to count in images of dense crowds, it might not count accurately in images of sparse crowds or even hallucinate crowds that it always expects in images with no crowds.

1.2 Machine Learning Fundamentals

Machine learning aims to learn from experience to improve performance in specific tasks. We define a machine learning model as a single instance of a machine learning algorithm that undergoes the process of training in a particular task and, after doing so, can be used for predictions or decisions on new data inputs [15].

Three main machine learning paradigms define the nature of the inputs and outputs of the model. *Unsupervised* learning presents the model with unlabeled data, and it has to find a structure in them on its own. *Supervised* learning adds annotations or labels to the data, and the goal is to learn the mapping of the inputs to the desired outputs. *Reinforcement* learning puts the model in a dynamic environment, behaving as an agent, getting constant feedback for its actions, and trying to maximize some reward [16].

1.2.1 Data Splitting

Data splitting aims to repeatedly evaluate the model's performance on new, unforeseen data. Different sets are used in model training, selection, and performance evaluation to ensure that new data are used in every step for an unbiased evaluation [15].

Firstly, the machine learning model is *fitted*[1] on the *training* data. For supervised learning methods, the training data consists of pairs of input vectors $\mathbf{x}_i$ and their corresponding target variables y_i. During the model training process, the model predicts target values $\hat{y}_i$ for different input vectors, and the difference between the actual target value y_i and the predicted value $\hat{y}_i$ is known as the *residual*. The model training process aims to minimize the differences between predicted and actual target values. To measure these differences, an *error function*, also known as a *loss function*, is defined, quantifying how well the model performs. The choice of loss function and method of adjusting the model parameters depends on the ML algorithm used.

Secondly, the model is evaluated on the *validation* set, predicting the target values. We calculate the total *loss*, representing the model's performance on unforeseen data, but we do not update the model's parameters on validation data. However, the validation loss is still used to tune the *hyperparameters*[2] of the model, during the model selection process, and to detect *over-fitting*[3].

Lastly, the final selected model is evaluated on the *test* set, which never had any impact during the model training and selection, and provides an

[1] *fitting = The process of adjusting the model parameters to fit the training data best.*

[2] *hyperparameters = Parameters set prior to model fitting, defining the architecture of the model itself, specifying how the model learns, regularization, and more.*

[3] *over-fitting = Common problem in machine learning, when the model starts to remember training data, instead of understanding the problem, leading to high accuracy on training data, but low accuracy on validation data, and resulting in poor generalization performance.*

unbiased evaluation of the model, which should be representative of actual application performance. In the world of ML, the test data is often not shared with whoever is designing the model to ensure it never impacts the decision process for model creation. Withholding the test set is common in machine learning competitions, research, and business practice.

Validation and test sets are sometimes used ambiguously and can be used differently based on used literature. Sometimes, when we are only presented with some data to develop a model, we might use the names train and test data, but the actual test data is inaccessible to us [17].

1.2.1.1 Data Split Ratio

In machine learning, we often have to deal with a limited amount of data, as data collection and *annotation*[4] can be time or resource-expensive process. The trade-off is to which set to assign what portion of the data, as we need both many observations to train the model on and enough for a representative evaluation later [15].

Depending on the situation and data, two commonly used rules are the 80/20 or 70/15/15 rules for train-test and train-validation-test splits, as pictured on the Figure 1.1.

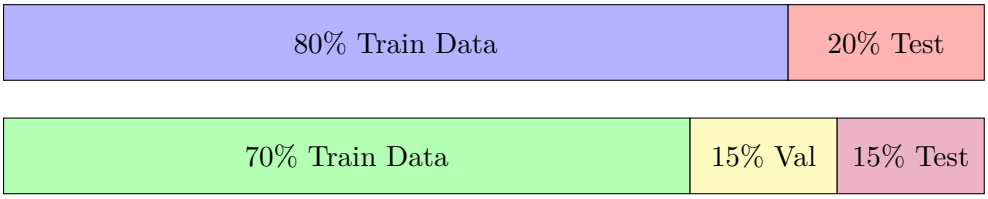


Figure 1.1: Illustration of the train/test splits.

1.2.1.2 K-Fold Cross-Validation

Another approach when pursuing maximal utilization of limited data for model selection is k-fold cross-validation. The idea behind cross-validation lies in doing multiple splits of available data to different train and validation sets and evaluating averaged model performance on k different validation sets, eliminating possible selection biases during the data splitting and limiting possible over-fitting. This approach is, however, computationally expensive, as the model has to be fitted k times on k different training sets [15, 18].

[4] *annotation = Also called a label, is the target variable y_i assigned to an observation x_i.*

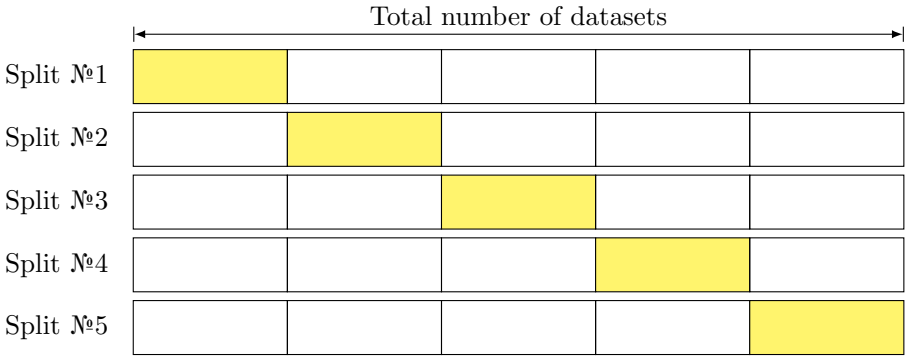


Figure 1.2: Illustration of the k-fold cross-validation.

1.2.2 Data Preprocessing

"Garbage in, garbage out" applies heavily to machine learning. Data preprocessing aims to ensure and enhance models' performance by reducing the unnecessary and manipulating the important data in training sets.

Common tasks include cleaning, imputation of missing values, range normalization, encoding of categorical features, discretization of continuous features, feature engineering, or dimensionality reduction. These tasks help address missing, or inconsistent data, irrelevant features, or high-dimensional data that can cause overfitting or slow down the training process, thus allowing machine learning models to achieve better accuracy, efficiency, and generalization performance [19].

However, we should never forget to train preprocessing methods only on training data before applying them to validation and test sets because it prevents the preprocessing algorithm from leaking information about the other sets into the training data. Suppose the preprocessing steps are also fitted on the validation or the test data before the model is trained. In that case, the model might be biased toward the specific characteristics of the validation or test data, leading to overfitting or poor generalization. Such information leakage can also lead to overly optimistic evaluation metrics. Therefore, it is crucial to keep the training, validation, and test data isolated and only fit the preprocessing steps on the training data before applying them to validation, and test sets [20].

With the rise of deep learning, data preprocessing has become even more crucial because deep learning models often require vast amounts of training data. Deep learning models are also more sensitive to the input features' scale and distribution, making preprocessing techniques like normalization and feature scaling more critical to ensure the numerical stability of deep learning models. New methods like *batch normalization* [21] were developed to alleviate the need for manual feature normalization by doing so automatically inside

the model itself. However, they have not yet obsoleted the need for manual rescaling as they cannot be used universally in all models [22].

1.2.3 Loss Function, Batch, Epoch

A loss function, sometimes also called an *error* function, measures the difference, or loss, between a predicted label $\hat{y}_i$ and a true label y_i, also called the *ground truth* [15]. The goal is to minimize the average loss during training, and update the model's parameters while doing so, a process that we call learning. Some literature refers to the average total loss on the training set as to the *cost* function. We denote the loss function as $\mathcal{L}(Y, \hat{Y})$, and the cost function is thus as follows, where N is the number of training samples.

$$\mathcal{C}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(Y_i, \hat{Y}_i)$$

With large training data, it is impossible to train the model on the entire set in a single step. Instead, we only use a smaller subset at every training step, which we call a *batch*, and we update the model parameters after every batch. Small batch sizes can result in slow training and unstable convergence [23]; too large batch sizes can be memory-expensive and lead to poor generalization [24]. For this reason, we need to treat the batch size as an essential hyper-parameter of the training process. Once we have used the entire training set, we call that a single *epoch*.

Different machine learning tasks require different loss functions, but all must meet specific criteria to work well. E.g., for algorithms using gradient descent as the method of loss minimization, a loss function must be both continuous, differentiable, and convex [15].

1.2.3.1 Classification

In classification tasks, the model learns to assign one of the k classes to a given input. We call this task a *binary* classification when $k = 2$, or *multi-class* classification for $k > 2$. Some models only predict the final class (e.g., decision trees), and some models output a vector of probability scores, to which of the k classes should the input vector $\mathbf{x}_i$ be mapped, which is a common approach in *neural networks*.

$$\hat{\mathbf{P}} = [\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_k]$$
$$\hat{P}_k = \hat{P}(Y = k | X = \mathbf{x}_i)$$
$$\hat{y}_i = \arg\max \hat{\mathbf{P}}$$

The loss most commonly used for classification is the *categorical cross-entropy*, and is defined as:

$$\mathcal{L}(Y, \hat{Y}) = -\sum_{i=0}^{k} y_i \log \hat{y}_i$$

1.2.3.2 Regression

In regression tasks, the model learns to predict a continuous numerical value based on input. Several loss functions commonly used for this task differ in their sensitivity to outliers and the values that they work with well.

$$\mathcal{L}_{MAE}(Y, \hat{Y}) = |y_i - \hat{y}_i|$$

Mean Average Error (MAE) measures the absolute difference and is not sensitive to outliers.

$$\mathcal{L}_{MSE}(Y, \hat{Y}) = (y_i - \hat{y}_i)^2$$

Mean Squared Error (MSE), on the other hand, penalizes more significant errors more heavily.

$$\mathcal{L}_{RMSE}(Y, \hat{Y}) = \sqrt{(y_i - \hat{y}_i)^2}$$

Root Mean Squared Error (RMSE) can be elegantly interpreted as the standard deviation of residuals.

$$\mathcal{L}_{RMSLE}(Y, \hat{Y}) = \sqrt{(\log(y_i + 1) - \log(\hat{y}_i + 1))^2}$$

Root Mean Squared Log Error (RMSLE) is a metric similar to RMSE but more robust to outliers. During training, the loss function is averaged across the entire batch or even across multiple regressors in multivariable regression tasks.

1.2.4 Model Selection and Evaluation Metrics

Whether a model performs as it should is never as straightforward as picking the one with the highest accuracy. Evaluating the robustness and possible shortcomings is sometimes a tedious process, in which we can compare different metrics in different scenarios to get a better insight into the model's possible shortcomings [15].

A simple example can be made of imbalanced data. If our training data contain, e.g., patients where only one in a hundred has a certain illness, then a model that always predicts the patients to be healthy has a 99% accuracy, yet is not very useful. For this reason, adequate testing methodology and the use of proper metrics are essential [25, 26].

However, the topic of different evaluation metrics is vast; thus, we will only focus on methods and terminology necessary for this thesis.

1.2.4.1 Bias-Variance Tradeoff

Bias-variance tradeoff is a fundamental concept in machine learning that describes the balance between two types of errors: bias and variance. Bias represents the difference between the predictions and ground truths, while

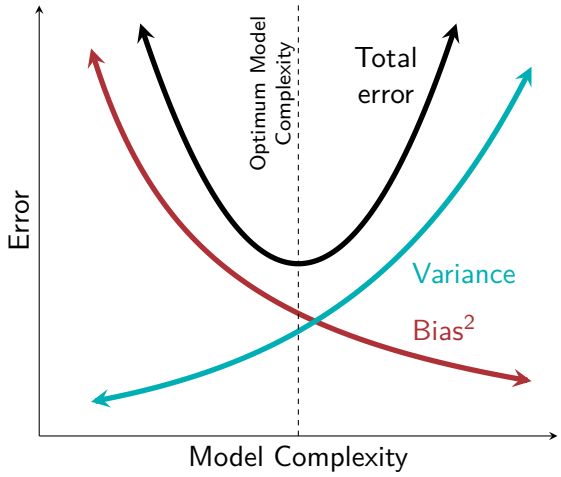


Figure 1.3: Illustration of the bias-variance tradeoff.

variance refers to the degree of variation of model predictions for different inputs.

A model with high bias under-fits the data because of its limited capacity to capture the complexity of the problem, resulting in large training loss and poor generalization performance. On the other hand, a model with high variance has too great of a capacity and over-fits the noise in the data, resulting in low training error but poor generalization performance [27].

An optimal balance between bias and variance must be found to achieve good performance, often called the "sweet spot" in model complexity. This can be achieved by hyperparameter tuning and regularization techniques such as *dropout*[5] or *early-stopping*[6].

Until recently, this was the common understanding of the bias-variance tradeoff. Recent research suggests that with increasing model complexity, second descent in total error exists and calls the peak in total error with rising model complexity the *interpolation threshold*.

This research also suggests the double descent to be the reason why neural networks with numbers of parameters way higher than data points used for training operate in the *interpolation mode*, which is the reason for their performance. Their final takeaway is that double descent does not nullify the concepts of bias-variance tradeoff and suggests that very complex models can still perform well [28, 29].

[5] *dropout = Regularization technique when random nodes and their connections in network are ignored during training.*

[6] *early-stopping = Technique to detect when over-fitting begins by monitoring the training and validation losses during training.*

1.2.4.2 Mean Average Error

The mean average error is the most straightforward metric for regression problems and is often used in crowd counting as it provides an intuitive estimate of the model's accuracy. It does not penalize outliers heavily, which can be desirable in challenging scenarios with a significant variance of viewpoints.

$$\text{Average Error} = \frac{1}{\text{N}} \sum_{i=0}^{N} |\text{Predicted Count} - \text{Real Count}|$$

It can, however, be a misleading metric, as closely matching the expected count does not mean the model was counting people in the image. As we will see later in this work, these models often find the most prevalent blob-like shape in the data, which would usually be a person's head, but it is not always the case. Minimizing MAE during training helps the model find optimal thresholds for detecting such blobs with a small error but does not result in a robust crowd counter. Thus, more metrics and tests are required for a better evaluation.

1.2.4.3 Accuracy

Accuracy is the most basic classification metric and is widely used as one of the main indicators of a model's performance, as it is easily interpretable and often telling. When applied to the crowd counting problem, the accuracy has to be interpreted more carefully, as the mean average error can easily be mistaken for accuracy.

In this work, we will take a closer look at *false positives* and *false negatives* in the predictions of crowd counting models. A false positive prediction in our context is a person hallucinated by the model, an incorrect count increase. A false negative is the opposite when the model ignores a person and does not increase the counter.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Breaking the previous formula down, we can see how a model that precisely predicts the expected number of people in a crowd can still be inaccurate if it ignores half the people in the image and hallucinates new people elsewhere. Such a model could be considered precise when judging by the average error metric, yet it does not portray the whole picture.

Due to the task of crowd counting not being of a simple classification nature but rather a regression, decoding individuals from the final count is not as straightforward. Instead, we will use small custom-designed datasets to understand the models and their limits better.

1.3 Deep Learning

The term *deep learning* refers to a class of machine learning models based around *artificial neural networks*. The first origins of artificial neurons, inspired by biological neurons and their interconnections – synapses – can be traced back to the work of McCulloch & Pitts in 1943 [30] and later to the *Rosenblatt's perceptron* in 1958 [31] that afterward became the building block for the first artificial neural networks [32] and gave birth to the entire field of deep learning.

1.3.1 Artificial Neuron

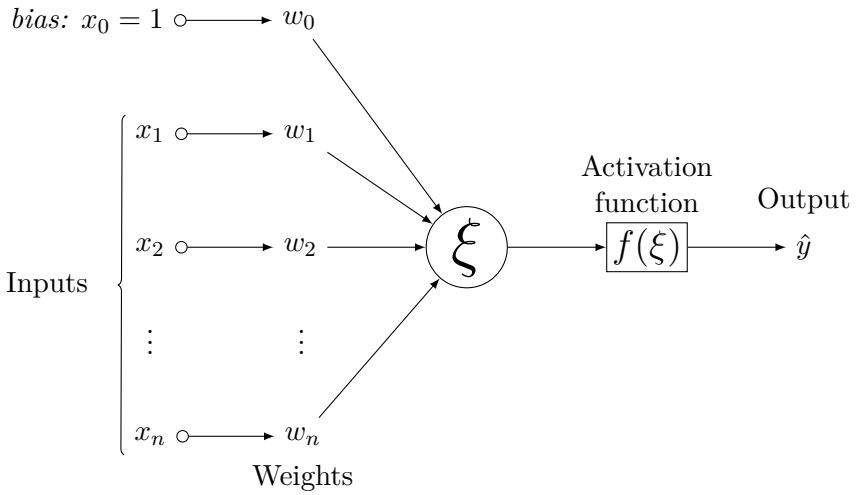


Figure 1.4: Single artificial neuron – *perceptron*.

A perceptron consists of n inputs and a single additional *bias* input, always set to output a constant value of one, which functions as an adjustable shift of the activation function. Every input has its weight w, which is a learnable parameter of the model. A weighted sum of the inputs and weights is called the *action potential* ξ, which is then fed to the *activation function* $f(\xi)$ to produce an output $\hat{y}$.

Choosing an appropriate activation function is crucial when building a neural network architecture. It must be continuous and efficiently differentiable for the gradient descent during training. The output range must also be considered to solve the task, as some commonly used activation functions have limited range [33].

Examples of some commonly used activation functions are the *sigmoid* function, *hyperbolic tangent*, or the *rectified linear units*.

1.3.2 Multilayer Perceptron

Multilayer perceptron is a class of *feedforward neural networks*, consisting of multiple layers of multiple neurons, in which the data only flows in a single direction, thus feedforward. The individual neurons are called *nodes*, and every layer can contain a different number of them, plus a bias node per layer, except for the output layer, which is the last one in the network.

Neural networks can solve complex problems, as the consequent layers can separate otherwise *linearly inseparable*[7] data by transformation. The networks also have a variable number of outputs, allowing them to solve complex tasks with multi-dimensional target variables [33].

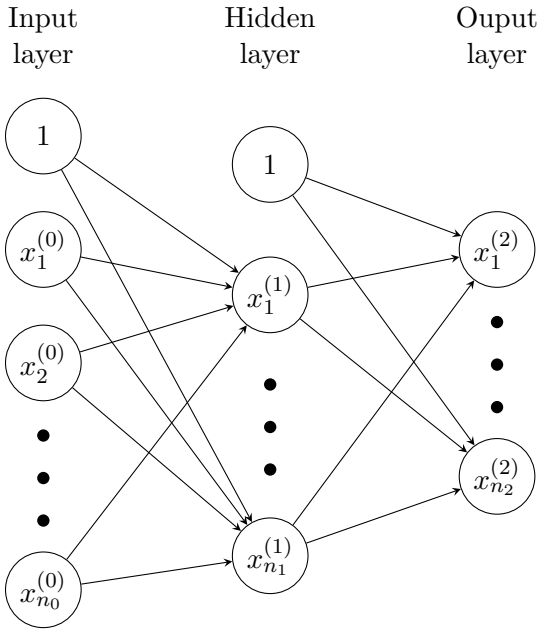


Figure 1.5: Multilayer perceptron with a single hidden layer.

In a multilayer perceptron, the l layers are *densely connected*, meaning every node is connected to all nodes from the previous layer. The following equation defines the activation of an individual node:

$$x_j^{(i)} = f(w_{j,0}^{(i)} + \sum_{k=1}^{n_{i-1}} x_k^{(i-1)} w_{j,k}^{(i)})$$

where $w_{j,k}^{(i)}$ is the k-th weight of $x_j^{(i)}$ and $x_k^{(i-1)}$ is the k-th element from the vector of outputs $\mathbf{x}^{(i-1)}$ from the previous layer $i-1$. The architecture defines the predicted variable, as the output layer can contain as little as a single

[7] *linearly inseparable = Data that cannot be separated by a single line, or a hyperplane in higher dimensions, for example the XOR function for linear classifiers such as perceptron.*

neuron for a single variable regression, a binary classification task, or as many neurons as necessary. The function of the entire network could be described as follows:

$$f(x) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$$

where n_0 and n_i are the counts of nodes in the input and output layers. We also call the number of nodes in a layer the *width* of the layer. The predicted target $\hat{\mathbf{y}}$ is the output of the last layer $\mathbf{x}^{(l)}$, defined as:

$$\mathbf{x}^{(l)} = f(\begin{bmatrix} 1 & x_1^{(l-1)} & \dots & x_{n_{l-1}}^{(l-1)} \end{bmatrix} \begin{bmatrix} w_{1,0}^{(l)} & w_{2,0}^{(l)} & \dots & w_{n_l,0}^{(l)} \\ w_{1,1}^{(l)} & w_{2,1}^{(l)} & \dots & w_{n_l,1}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,n_{l-1}}^{(l)} & w_{2,n_{l-1}}^{(l)} & \dots & w_{n_l,n_{l-1}}^{(l)} \end{bmatrix})$$

Learning in these complex models is often done with large training data, and many weights throughout the architecture must be fitted. This was made possible by the invention of the error backpropagation algorithm for multi-layer perceptron on smaller batches of data, which enabled simple *end-to-end*[8] training of these models [34].

Another important thing when training a network on such large data is regularization to prevent overfitting and improve generalization performance. An example of a commonly used technique is the dropout, when random nodes are disabled during training, forcing the network to learn robust and redundant representations for classification. Although such learning is slower overall, this approach might lead to faster convergence to a good solution. For this reason, dropout has been adapted and widely used in many different neural network architectures [22].

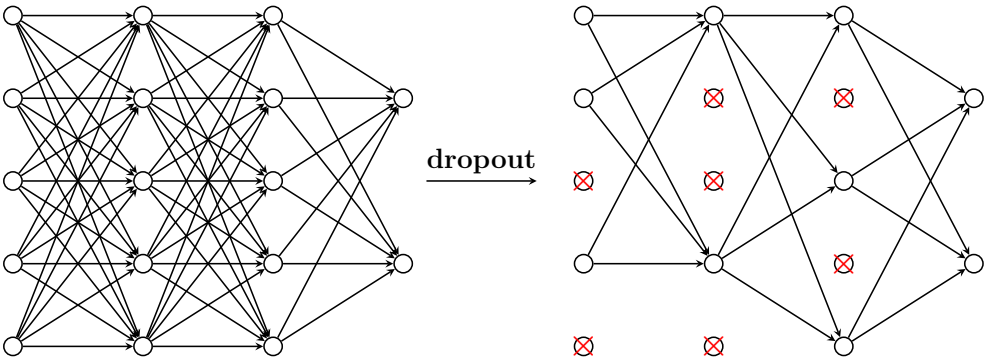


Figure 1.6: Illustration of the dropout regularization in MLP.

[8] *end-to-end training* = Compared to traditional approaches with complex feature engineering and preprocessing before training a classifier, during end-to-end training, networks learn intermediate representations of features and their transformations on their own, simplifying the entire process.

The problem with a multilayer perceptron, however, is its scalability. The number of nodes increases rapidly for larger inputs, such as images. A single 128×128 grayscale image requires 16,384 input nodes, and every following layer of the same dimension requires $16,384^2 = 268,435,456$ weights. For this reason, different neural network architectures had to be developed for computer vision tasks and the processing of images.

The densely connected layers of the multilayer perceptron are, however, still an essential building block in many neural network architectures, often close to the output of the architecture, where complex decisions can be made on pre-processed features [33].

Chapter 2

Computer Vision

Computer vision is a field of study that allows computers to extract information from the visual world around us, such as images and videos, understand them, and even base decisions on them. It is a multidisciplinary field closely associated with image and signal processing, computer graphics, robotics, neuroscience, and artificial intelligence. Its applications in our everyday lives are many, from robotics and self-driving vehicles to augmented reality and medical imaging.

2.1 Brief History of Computer Vision

The roots of computer vision research originated in the 1970s. The invention of line detection and edge extraction algorithms laid the groundwork for many modern approaches. These algorithms were used for hand-crafting of image features in combination with simple object-tracking algorithms and motion estimators, often used for military purposes of vehicle tracking and missile launches [35].

The field matured enough in the following 20 years to solve more complex tasks and understand images better. Computer vision became more available outside of military applications for tasks such as automatic number plate recognition [36], rapid detection [37], and even recognition of human faces [38]. With computer hardware and software advancements, computer vision algorithms found use in many fields, from medical imaging to industrial automation and entertainment [35].

This era had mastered the hand-crafted features with methods such as *HoG*[9] [39], *SIFT*[10] [40], and *SURF*[11] [41], but features alone are not enough for computer vision. Hard-coded algorithms and heuristics had their limits.

[9] *HoG = histogram of oriented gradients*
[10] *SIFT = scale-invariant feature transform*
[11] *SURF = speeded up robust features*

The paradigm slowly shifted towards statistical modeling, which emphasized using probabilistic models to represent and reason about the underlying structures of visual data. Together with the advancements in computation performance and larger datasets being increasingly available, hand-crafted features started to be used in combination with machine learning models such as the SVMs[12] [42] or with *k-NN*[13] [43]. These methods finally generalized well enough to be used in varied visual contexts and applied in previously difficult-to-tackle domains, as they could be trained on large datasets.

2.2 Deep Learning Methods

A major breakthrough came in the year 2012 when Alex Krizhevsky obliterated all competition in the annual *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) by implementing an efficient *convolutional neural network* (CNN) on GPUs[14], as their training on CPUs with such large data was unfeasible. This alone kickstarted a new era of deep learning in computer vision tasks, as neural network approaches have shown superior performance on very complex datasets, possibly due to their immense learning capacity and capability to learn more structure from large data [44].

Another significant improvement was that compared to traditional approaches, this model was end-to-end trainable, simplifying the entire process by learning the image features on its own efficiently. By implementing the network on a GPU rather than a CPU, the computation, which mainly consists of large matrix multiplications, could be heavily parallelized, taking advantage of many processing cores in a GPU. Suddenly, very large networks with large learning capacities could be trained in reasonable amounts of time, making graphics cards the norm in machine learning to this day [33].

2.2.1 Convolutional Neural Networks

Convolutional neural networks were inspired by animal visual cortices, mimicking the organization of brain cells. They dramatically reduce the number of needed parameters in the network by only ever comparing local neighborhoods of pixels, compared to multilayer perceptron, which compares all possible pixel pairs and does not take any advantage of the local structure and spatial locality [45].

The CNN architecture achieves this by utilizing an operation called *convolution* instead of large matrix multiplications. In simple terms, convolution is a weighted sum of some neighborhood of a pixel and can be adjusted in many different ways to achieve different goals. There are many ways how their behavior can be modified, such as different *strides* and *dilated* convolutions.

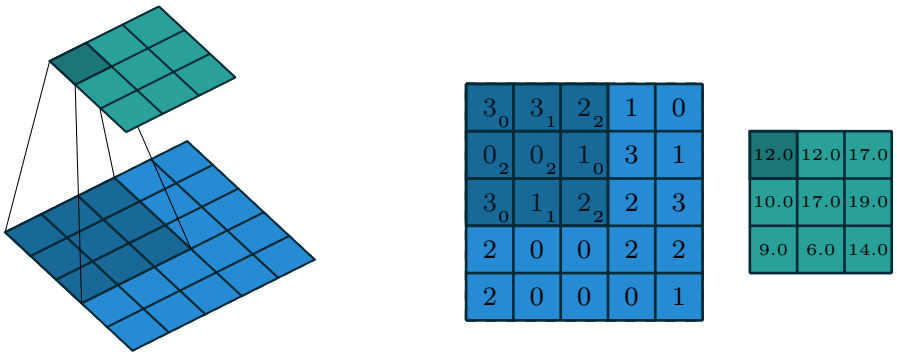
[12] *SVM = support vector machine*
[13] *k-NN = k-nearest neighbors algorithm*
[14] *GPU = graphics processing unit*

2.2.1.1 Convolution

The convolution is a simple weighted sum of a pixel's neighborhood. The neighborhood and its associated weights are called the *kernel*. This operation is used as a sliding window over the entire image. We define convolution as:

$$(\mathbb{I} * \mathbb{K})_{i,j} = \sum_{a=0}^{o-1} \sum_{b=0}^{p-1} \mathbb{I}_{i+a,j+b} \cdot \mathbb{K}_{a,b}$$

where $\mathbb{I}$ is an image with dimensions m, n, $\mathbb{K}$ is a kernel with dimensions o, p, and $(\mathbb{I} * \mathbb{K})_{i,j}$ is the element at the position i, j in the resulting matrix $(\mathbb{I} * \mathbb{K})$ with the dimensions q, r; where $q = m - 2\lfloor \frac{o}{2} \rfloor$ and $r = m - 2\lfloor \frac{p}{2} \rfloor$.



(a) Blue matrix represents the input.

(b) Single kernel applied as a sliding window.

Figure 2.1: Simple convolution with a kernel size of 3×3 and no padding [46].

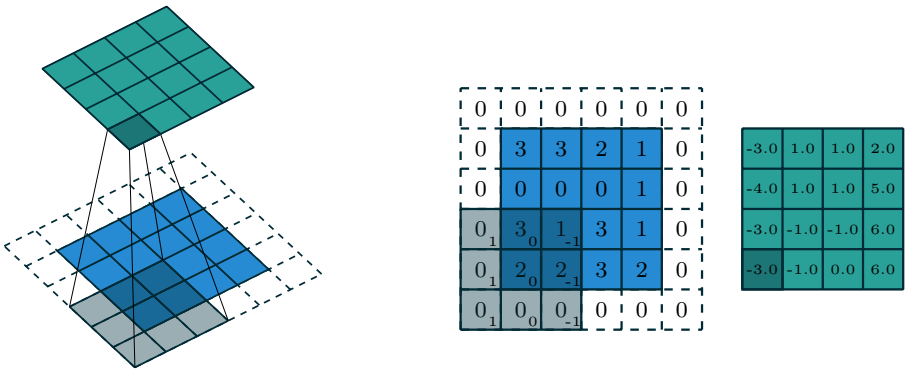


Figure 2.2: Zero-padded convolution with a kernel size of 3×3 [46].

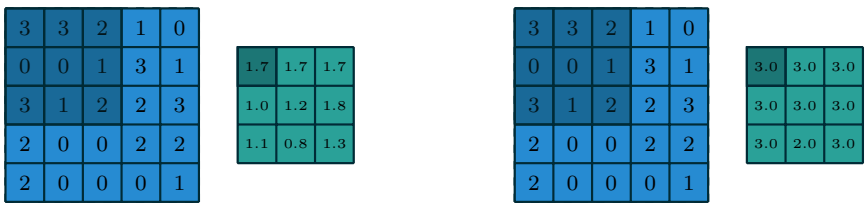
As can be seen in Figure 2.1, the convolution filter cannot produce an output of the same dimension, as the filter cannot be centered on a border

value, which results in out-of-bounds values in the weighted sum. When the same size output is desirable, we can add the out-of-bounds values in a process called *padding*. We can pad the image with any values we choose; common approaches are zero padding, as seen in Figure 2.2, or repeating the border value.

The size of a kernel is by no means limited to 3×3, different filters can work as receptive fields of different sizes, and larger kernels are often used. The only drawback is that large kernels are computationally and memory expensive. One workaround for larger kernels is to use *separable* kernels. Instead of a single $n \times n$ 2D kernel, two consequent 1D kernels are applied, $n \times 1$ and $1 \times n$, which reduces the kernel complexity, with the possible reduction of accuracy but significantly more computationally efficient, with only $2n$ kernel weights instead of n^2 [47].

2.2.1.2 Pooling

Another standard convolutional neural network building block is the *pooling* layer. Pooling layers decrease the dimensionality and, thus, computational requirements by combining filter responses over local neighborhoods while allowing for a certain degree of spatial translation invariance. This spatial invariance increases with the number of layers, allowing networks to detect features of different scales. They can also help the network reduce overfitting by forcing it to learn more robust features [48].



(a) Average pooling operation.

(b) Max pooling operation.

Figure 2.3: Different 3×3 pooling operations [46].

When shrinking the feature maps with the pooling operation, the choice between average and max pooling has to be considered, as max pooling focuses mainly on the presence of the most prominent features. In contrast, average pooling provides a smoother feature map. In practice, max pooling is used to identify prominent features, while average pooling is only used for dimensional reduction and to collapse feature maps to particular sizes.

2.2.1.3 Tensors

Convolutional neural networks might be easy to illustrate on a 2D matrix with a single convolutional filter, but that is rarely how they look in real life. In practice, inputs and outputs of networks and their layers are commonly described as *tensors*, which are multi-dimensional arrays. For example, an RGB image could be represented as a mode-3 tensor or a 3-way tensor, with the dimensions of height $\times$ width $\times$ color channels.

During the training of a CNN, we usually train the network on multiple samples at once, a single batch. The input for a network is thus a 4-way tensor $\mathcal{A}$ over a field of 32-bit float numbers $\mathbb{F}$ with the dimensions batch size $\times$ width $\times$ height $\times$ number of channels. Learning multiple kernels per convolution layer is common, which increases the number of channels in the network. A common approach is to convolve all feature maps of an image to eventually become a feature vector, with the only dimension being the number of channels, on top of which we build a classifier using fully connected layers or appropriate convolutional operation to finalize the classification.

If we replace the fully connected layers at the end for a convolution layer instead, we call the network *fully convolutional*. The advantage of fully convolutional networks is that input height and width can be of variable size, as the tiled convolution with sliding window kernels works with inputs of arbitrary height and width.

The only drawback of such an approach is the memory required to use larger, fully convolutional models with high-resolution photos. The advantage of this approach is that instead of splitting large images, they can be processed in a single go without the problems at the borders of split photos.

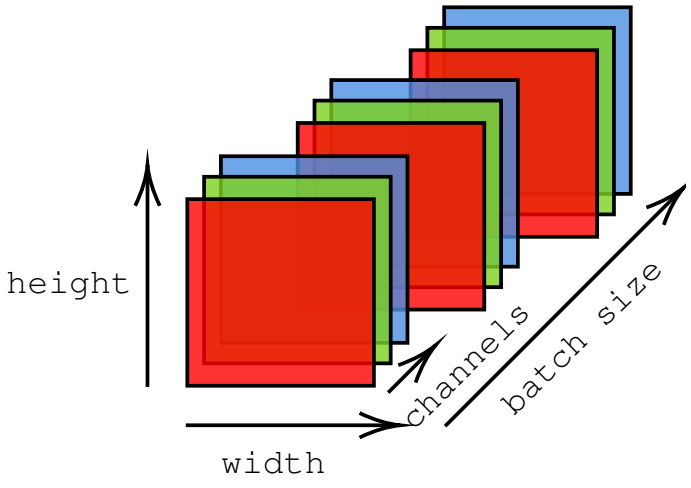


Figure 2.4: Illustration of an input tensor with 3 channels and batch size of 3.

2.2.1.4 Architectural Designs

The field of architectural design is quickly evolving, and newer architectures with marginal improvements for different tasks are published almost monthly. However, a few notable works have advanced the field, and their concepts still influence the designs of many newer architectures.

After the immense success of AlexNet in 2012, the family of VGG networks pushed the depth of convolutional networks to 16 and 19 layers, respectively, with the VGG16 and VGG19 models. They only used 3×3 kernel size for faster training and inference [49]. Going beyond 19 layers was difficult due to the vanishing gradients problem. Meanwhile, in 2014, the *Inception* family of networks was introduced, which used concatenated filters of different sizes to extract more complex features in the limited number of layers [50].

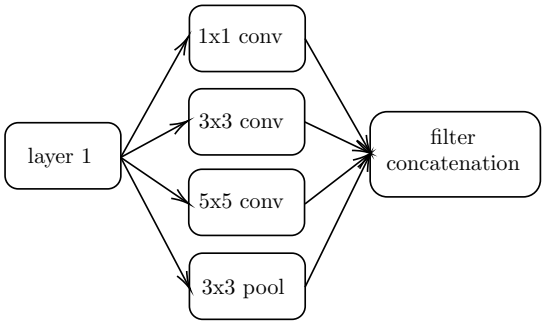


Figure 2.5: Inception block example.

A year later, in 2015, the family of ResNets was introduced by the team at Microsoft Research, which pushed the depth of CNNs to 152 layers.

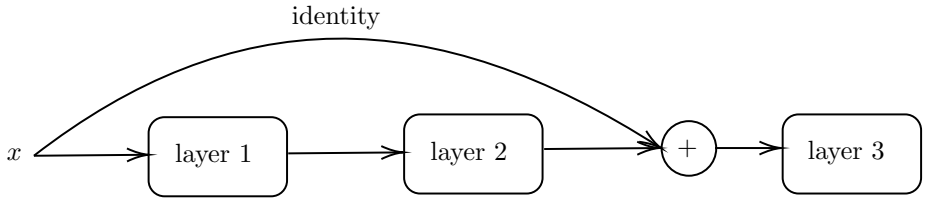


Figure 2.6: ResNet residual block example.

ResNets contain *residual* connections – identity shortcuts in the network that add the output of a layer to one later in the network, alleviating the vanishing gradient problem with increasing depth, which increases the accuracy and learning capacity of the network [51]. Since their invention, residual connections have been widespread in architectural designs of different networks, as they bring significant improvements at nearly zero computational cost.

2.2.2 Transformer Architectures for Computer Vision

The *transformer* architecture was originally designed as a *sequence-to-sequence* model (seq2seq), firstly designed for *natural language processing* (NLP), which is a field focused on problems such as machine translation. It relies solely on the *attention* mechanism, which allows the network to weigh and focus on different parts of the input. Compared to previous works in the field that utilized *recurrent* neural networks, it offers excellent potential for parallelization, as it can process the entire input at once, instead of single input at a time [52].

This approach was very successful in the field of NLP, and soon, the architecture was adopted in computer vision as well, which was difficult due to the nature of images not being a sequence. One approach uses image splitting into a set of patches treated as a sequence, and their position within a picture is positionally encoded into every image patch. This architecture was named the *Vision Transformer* (ViT) [53].

Another approach uses a convolutional network to learn a representation of the input image, flattens, and positionally encodes it before feeding it into a classical transformer encoder-decoder network for predictions [54].

All the described models, however, share a significant drawback, as they lack some of the inductive biases inherent to CNNs, and, therefore, might not generalize well when trained with insufficient amounts of data but have enormous learning capacity when the data to learn from is available [52]. This has led to an explosion of both model parameters and the size of datasets, making many transformer architectures challenging to use in domains with insufficient and hard-to-collect data. Those large models are also costly to train in terms of computation, memory, and required energy, thus only allowing researchers with plenty of resources and a powerful infrastructure to train these models. However, some smaller transformer architectures for computer vision can still be used with smaller datasets and perform well [55].

2.2.2.1 Attention

Attention in machine learning is a technique to evaluate the relevance of different input parts and pay attention to them in a weighted nature. The network learns which parts of the data are relevant and the context in which this applies. Attention is also the primary building block of the transformer architecture [52].

We denote the attention from token i to token j as $a_{i,j}$. It is important to note that attention is asymmetric, and high attention $a_{i,j}$ does not translate to high $a_{j,i}$, as those are two different values. This asymmetry is achieved by the design of attention itself, inspired by database information retrieval. One input is used as the query and is compared to a set of keys that hold the corresponding values.

An input token i is embedded in a vector x_i. Three matrices are used for the calculations: the *query weight matrix* $\mathbf{W}_Q$, the *key weight matrix* $\mathbf{W}_K$, both of which with the dimension of d_k, and the *value weight matrix* $\mathbf{W}_V$ with the dimension d_v. The query, key, and value vectors q_i, k_i, v_i are calculated as a product of the input vector and the corresponding weight matrix. The attention values are obtained as a dot product of the q_i and k_i vectors and are scaled down by the value of $\sqrt{d_k}$ and normalized using a *softmax* function. The final output is a sum of the value vectors weighted by their corresponding attention values. The calculation can be parallelized for all tokens by writing all token vectors as rows of the matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ [52].

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

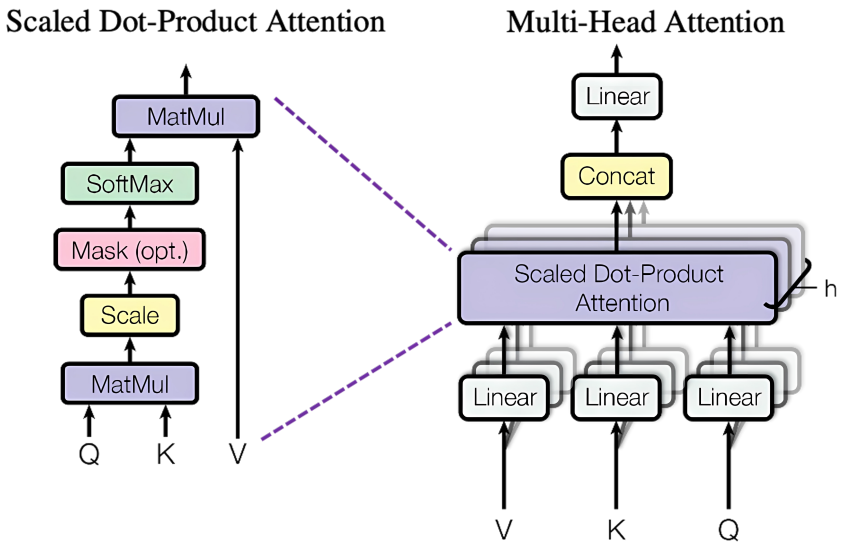


Figure 2.7: Multi-head attention [52].

2.2.2.2 Multi-Head Attention

Instead of single attention head, meaning a single combination of $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ matrices, multi-head attention linearly projects the input tokens h times and performs the computation in parallel, concatenates the outputs and re-projects them into the previous dimension. This allows the model to attend to different pairs of tokens simultaneously, allowing for different representations of tokens, for example, words with multiple meanings based on context. This is impossible with a single-head attention model.

$$\text{MultiHeadAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)\mathbf{W}^O$$

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$$

The dimensions of the projection parameter matrices are $\mathbf{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$, where d_{model} is the embedding dimension of tokens.

In the work "Attention Is All You Need," the authors used eight parallel attention layers (= heads) with reduced dimensions $d_k = d_v = \frac{d_{\text{model}}}{h} = 64$, which reduced the computational cost to that of a single-head attention with full dimension [52].

2.2.2.3 Types of Attention Layers

As described in Section 2.2.2.1, attention can be calculated between different inputs. Attention between tokens from single input is called a *self-attention*. We can find two more types of attention layers in the encoder-decoder architecture (Figure 2.8). The *encoder-decoder attention*, also called a *cross-attention*, where queries come from the decoder itself, and keys and values come from the encoder, which allows the decoder to attend to the entire input sequence. Another type is the *masked self-attention* in the decoder, in which every token is only allowed to attend to the previous tokens. This allows for an auto-regressive generation of output without attending to future outputs. This is achieved by the masking of all values corresponding to illegal attention connections [52].

2.2.2.4 Positional Encoding

To allow the transformer to learn the positions of tokens in the input, we use *positional encoding* that provides the transformer with the information where the individual words are. Positional encoding can either be function-defined or learned as a parameter matrix. The original transformer architecture used sine and cosine functions of different frequencies to create unique positional identifiers that could be shifted via a straightforward linear transform. The team hypothesized it would allow the model to learn to attend to relative positions rather than absolute ones [52, 56].

The positional encoding can also be learned as a parameter matrix during the training. The choice of function-based vs. learned PE is another hyper-parameter to be tuned, as the performance of each can differ, depending on the task, as some works show equivalent performance, and others benefit from one or the other approach. This also heavily depends on whether the transformer is used for NLP or as a Vision Transformer. However, one thing is very clear: without positional encoding, transformer models perform significantly worse [52, 53, 56, 57].

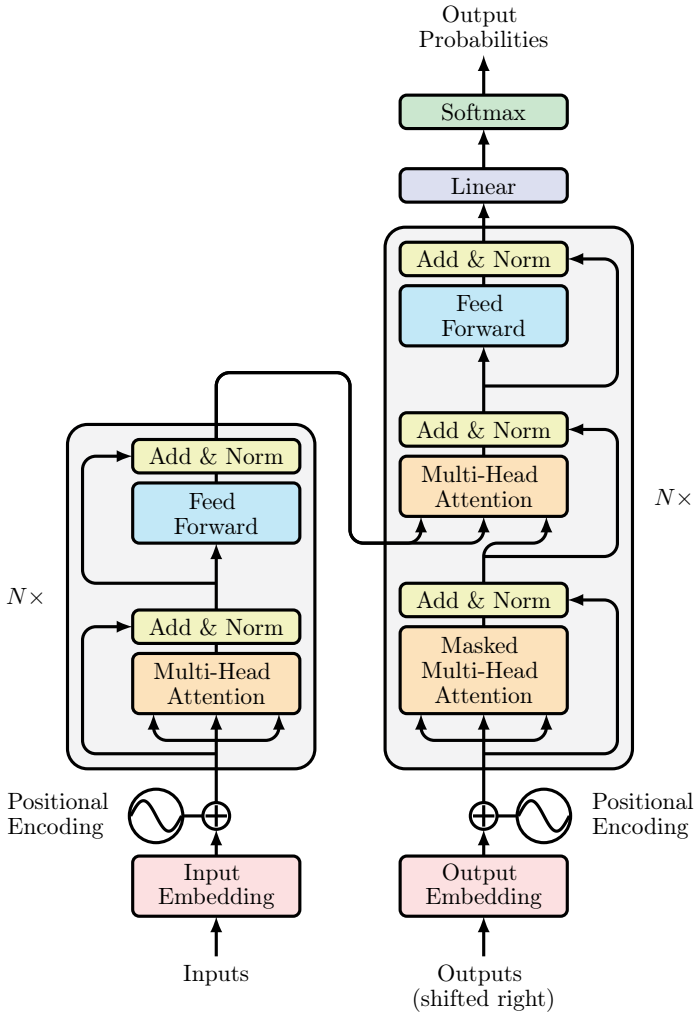


Figure 2.8: Transformer encoder-decoder architecture [52].

2.2.2.5 Convolutional Transformer Architecture

In recent years, researchers have been exploring ways of combining the strengths of CNNs and transformer models by using convolutions for local feature extraction and using the resulting feature vectors in a transformer to capture the long-range dependencies and the global context in the data. This approach has been successful in both the tasks of NLP [58], and computer vision [59], for which are both *Vision Transformers* (ViT) that use patching and *Convolutional Transformers* (CvT) that use convolutional approach commonly used today, but CvT improves significantly on computational efficiency [55, 60].

Chapter 3

Crowd Counting

Crowd counting is a research area in computer vision that aims to estimate the number of individuals present in an image or video, allowing for a wide range of applications, such as crowd tracking for traffic optimization and overcrowding prevention, urban planning, or crisis crowd management. These methods have also been successfully used in a variety of different scenarios beyond counting people, from tracking animals in wildlife surveys [61], counting cells in medical imaging [62], counting crops on a field [63] or counting vehicles on a highway [64].

Crowd counting comes with a set of challenges that are difficult to overcome. The varying crowd densities, overlapping individuals, different viewpoints, and challenging weather conditions, combined with the often poor image quality from CCTV cameras, make designing a robust and accurate crowd counter very challenging.

Earlier approaches were often detector-based, detecting individuals based on hand-crafted features and counting the individual detections. This approach had two significant limitations; the hand-crafted features generalized poorly when the crowd's appearance or characteristics changed and had trouble with occluded individuals. Such detectors are unsuitable for densely crowded scenes. Their performance suffers greatly, as it is hard to distinguish multiple individuals from multiple detections of a single person when the non-maxima suppression fails. Their error rates were also large, considering the small crowd sizes on which they even worked [65].

Since the great success of deep learning for computer vision tasks, newer approaches for dense crowd counting have been invented. The end-to-end model training paradigm allows for the automatic learning of features on large-scale datasets. These models generalize better and have outperformed all previous methods.

The main three types of crowd counting methods today are the direct regression methods, which directly predict the number of individuals, the density-based methods, which train the network to predict a density map rep-

resenting the spatial distribution of people in the image; and the point-based and localization methods, that predict center points of individuals in the image. Methods based on object detection have been almost entirely abandoned, despite R-CNN and YOLO models being very effective object detectors, for they are ill-suited for counting overlapped individuals. Density-based methods are currently the most prevalent in the field, as they are forced to learn the features of the crowd itself and are thus more robust to occluded individuals in very dense crowd scenarios. Point-based methods, on the other hand, provide us with more precise information about crowd localization and could be better suited for individual or crowd flow tracking and crowd behavior prediction [66].

3.0.1 Crowd Counting Datasets

As discussed earlier in Chapter 1, machine learning models can only be as good as the data they are trained on. In the field of crowd counting, there are a few essential datasets that have become the norm for the benchmarking of crowd-counting methods. Each has certain limitations, as the data collection methods and the data characteristics are very different in every dataset. For proper evaluation, all methods are usually benchmarked on multiple datasets for a better overall evaluation of the method.

In our work, we have picked four datasets to evaluate our models on, but we mention other notable datasets widely used. The main characteristics are summarized in Table 3.1. The datasets are sorted by their year of publication, and a few essential metrics are listed. The critical fact is that most datasets do not contain negative samples, the importance of which will be shown later in this work. Image resolutions and total annotation counts are essential measures of how much information can be extracted from the dataset using augmentation, such as region cropping. Larger image resolutions, however, can be problematic during the evaluation of some models due to the large memory demand for count inference. All datasets contain point annotations for centers of heads, and some include bounding boxes.

Dataset	Total Images	Avg. Image Resolution	Count Statistics			
			Total	Min	Avg	Max
WorldExpo'10	3,980	720×576	199,923	1	50	253
UCF_CC_50	50	2888×2101	63,974	94	1,279	4,543
ShanghaiTech A	482	868×589	241,677	33	501	3,139
ShanghaiTech B	716	1024×768	88,488	9	123	578
UCF_QNRF	1,535	2902×2013	1,251,642	49	815	12,865
GCC (synthetic)	15,212	1920×1080	7,625,843	0	501	3,995
NWPU-Crowd	5,109	3209×2191	2,133,375	0	418	20,033

Table 3.1: Comparison of selected crowd counting datasets [1–3, 67–69].

3.0.1.1 WorldExpo'10

This dataset was collected during the 2010 WorldExpo in Shanghai and features 108 different surveillance camera bird views. The main drawback of this dataset is limited resolution and small numbers of people per image, with no larger scenes with denser crowds [67].

Figure 3.1: Examples from the WorldExpo'10 dataset [67].

3.0.1.2 UCF_CC_50

This dataset was collected from FLICKR images, shows mainly extremely dense crowds, and features very high image resolution. However, it contains only 50 grayscale images, thus offering small viewpoint and crowd appearance variance, compared to other datasets [68].

Figure 3.2: Examples from the UCF_CC_50 dataset [68].

3.0.1.3 ShanghaiTech

The ShanghaiTech dataset is the most often used crowd-counting dataset, measured by open-access papers mentioning it [70]. It features two parts with a total of 1198 images. Part A is collected from the internet, while images from part B were collected from the streets in Shanghai, and all images have been taken from different viewpoints [3].

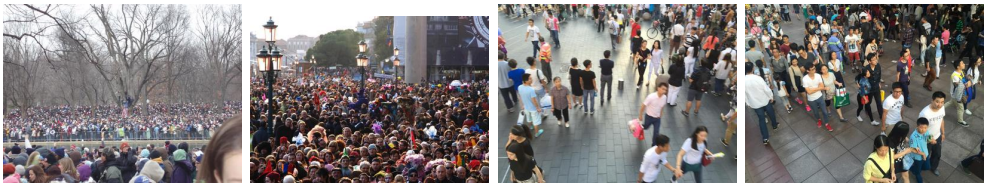


Figure 3.3: Examples from the ShanghaiTech dataset, first two pictures from part A, second two from part B [3].

3.0.1.4 UCF_QNRF

This large dataset was collected from the internet in high resolution, featuring very dense crowds. Compared to previous datasets, UCF_QNRF features a very large diversity, as these images have been sourced specifically from all parts of the world to ensure a wide variety of crowd appearances. Crowd sparsity combined with vegetation and buildings is present in the data, which makes the dataset better-suited for the training of crowd counters in real-world scenarios [2].

Figure 3.4: Examples from the UCF_QNRF dataset [2].

3.0.1.5 GCC

GCC is a synthetic dataset and framework designed to generate large crowd images with pixel-perfect annotations on demand. This allowed the authors to collect many pictures of diverse crowd scenarios without time-consuming and labor-intensive human annotation. The authors also propose a GAN[15]-based method to translate the synthetic images into realistic scenes. The authors show that larger datasets make for better crowd counters and pre-train their networks before benchmarking on different datasets to beat previous state-of-the-art methods [69].

Figure 3.5: Examples from the synthetic GCC dataset [69].

3.0.1.6 NWPU-Crowd

The NWPU-Crowd is a diverse dataset designed to benchmark[16] modern crowd counting and localization and features 5109 high-resolution images and the most extensive density range. It is also the only dataset mentioning negative samples in the data for robustness evaluation. They also provide an impartial benchmark for researchers to validate their results. The results also

[15] GAN = Generative Adversarial Network

[16] This benchmark is accessible at `https://crowdbenchmark.com/`.

include robustness information by measuring the MAE and MSE loss on images categorized by luminance levels and scene details [1].

Figure 3.6: Examples from the NWPU-Crowd dataset [1].

3.0.2 Density-Based Methods

Crowd counting is an open-set problem, as there is no real upper bound to how many people there can be in the picture. For direct count regression methods, a problem of proper generalization beyond crowd sizes present in the training data arises. In such models, we also have no idea on what basis the model decides the final count, nor do we have any information about crowd localization. Many different methods have been proposed to tackle the open-set nature of the problem, such as the spatial divide-and-conquer method. However, they suffer from inaccuracy at the division lines, of which many exist, especially in very dense crowds [71].

Figure 3.7: Example of the crowd density map.

Instead of global density estimation by direct count regression models, another paper proposed a 2D regression of the image annotations captured on a density map representing the crowd based on numbers of individuals in different areas, as seen in Figure 3.7. This density map is generated by placing a 2D Gaussian kernel at each annotation and summing the resulting maps into a total density map [67]. The selection of Gaussian variance σ depends on several factors and impacts the crowd counter's final behavior. Instead of performing a grid search for another hyperparameter of a constant value of σ, adaptive methods exist that determine the value based on local

density estimate. Local density is usually estimated by the distance to k-nearest neighbors, and results in a better density map generation.

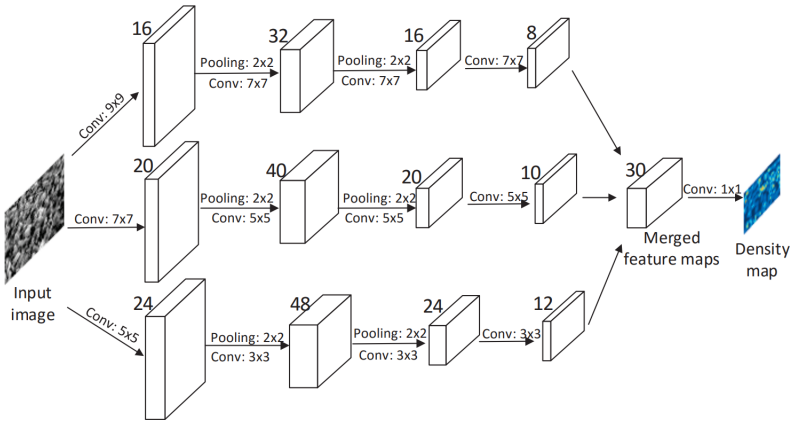


Figure 3.8: Multi-column convolutional neural network (MCNN) [67].

However, the scale of features with increasing crowd density can still be problematic, which different works often treat similarly by extracting and merging feature maps of different scales. The *Multi-Column Convolutional Neural Network* (MCNN) uses three different pathways inside the network, each with a different size of convolution filters, to create feature maps of different scales and then merges them to produce the final density map, as seen in Figure 3.8 [67]. Newer works often adopt the *feature pyramid network* (FPN) in their architecture, such as the *Scale-Adaptive Selection Network* (SASNet), which uses a VGG16 backbone to extract features of different scales on the bottom-up pathway, and uses the top-down pathway as a decoder to produce multiple density maps, which are fused to obtain the final prediction, as seen in Figure 3.9. The network also learns which feature layers are most precise at the given scale and merges them as a weighted average, which helps mitigate the gap between discrete feature scale levels [72].

Although CNN models have achieved state-of-the-art accuracy in many different tasks, certain design aspects of their architecture might be holding their accuracy back. Concretely, fixed-size and fixed-weight convolutional kernels limit the receptive fields and are applied regardless of crowd geometric and visual variations. Some works tried to enhance the receptive fields with either *dilated* convolutions [73], *deformable* convolutions [74], or most recently, the transformer architecture. Although it comes at a great computational expense, it improves accuracy in specific scenarios [75]. We were, however, not able to reproduce these results, as most works either did not publish their source code [75, 76], published incomplete or broken code [77], or did not publish pre-trained weights and have too great computational requirements for us to train [78].

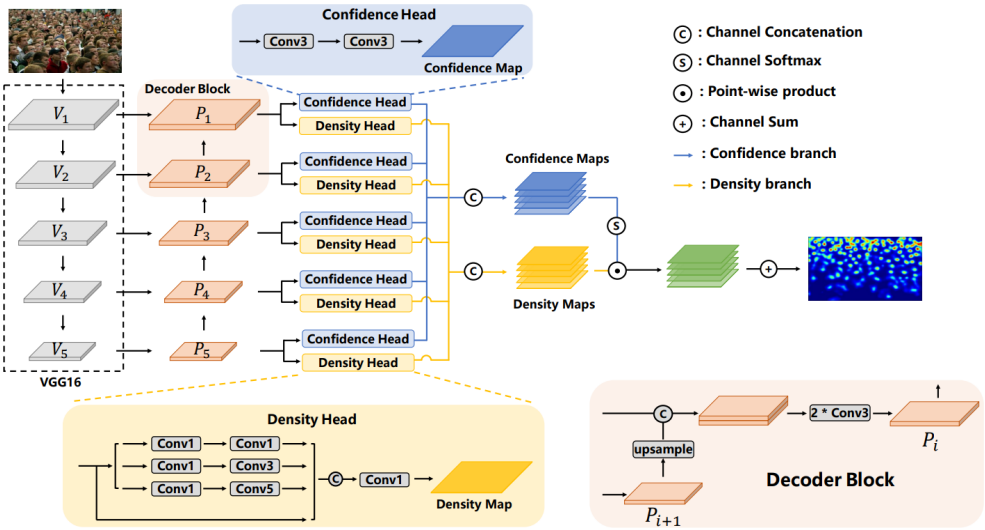
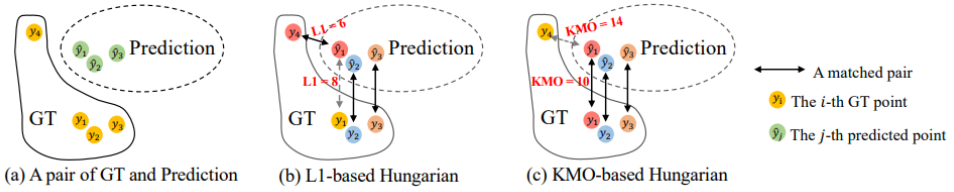


Figure 3.9: Scale-adaptive selection network (SASNet) [72].

3.0.3 Point-Based Methods

In contrast to density-based methods, point-based methods predict individuals directly in the crowd by their coordinates, which results in easier-to-interpret data that can be better used for crowd flow or individual tracking. Density-based methods can also be modified by applying *find-maxima* functions in post-processing. However, the localization of individuals is inaccurate in extremely dense crowds, and individuals cannot be easily distinguished from the density map [79].

However, a few problems with this approach make designing these models more technical. One of these problems is the design of the loss function itself, as the predicted points and ground truth values must be matched first before calculating any distances. The Hungarian algorithm is often used; however, as noted in [80], it is not always ideal, as seen in Figure 3.10.



(a) A pair of GT and Prediction (b) L1-based Hungarian (c) KMO-based Hungarian

(a) A pair of GT and predictions. (b) The $L1$-based Hungarian generate unsatisfactory matching results. (c) The proposed KMO-based Hungarian models the context as the matching cost, generating more reasonable matching results.

Figure 3.10: Examples of different prediction-GT point matching algorithms, (c) uses k-NN matching objective as auxiliary matching cost [80].

The second issue is the network's design to tackle the problem's open-set nature. The P2PNet starts with $H \times W \times 4$ point proposals (or 8 for the UCF_QNRF dataset with denser crowds) positioned in a grid throughout the pyramid levels, on which it builds two prediction branches – classification and regression – to produce both individual points and their corresponding confidence scores, as seen in Figure 3.11 [66].

In another transformer-based network called CLTR, authors replaced point proposals with trainable decoder instance queries. In this architecture, the authors used a CNN backbone and encoder-decoder architecture to process image features, on top of which they have built regression and classification heads, similar to the P2PNet architecture. The authors did, however, not publish all source files, so we could not test this method, and the impact of instance queries instead of point proposals on the maximum predicted density, combined with the global self-attention mechanism for better accuracy [80].

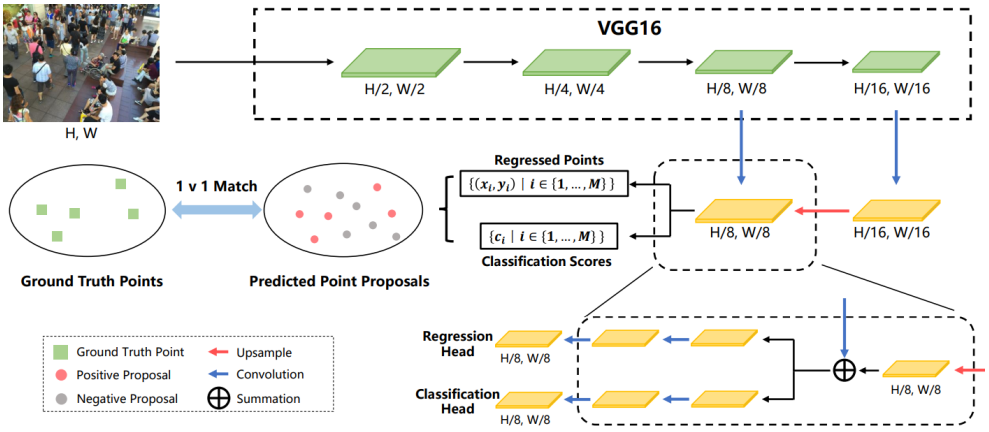


Figure 3.11: The P2PNet architecture [66].

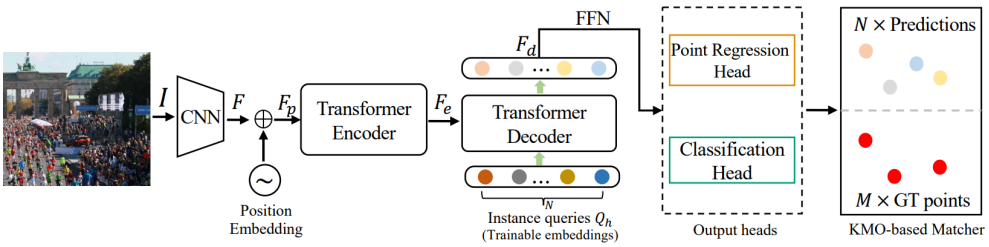


Figure 3.12: The CLTR architecture [80].

Chapter 4

Experiments and Results

In this chapter, we aim to assess and contrast the approaches employed in crowd counting over recent years regarding their robustness and accuracy. Our analysis identifies when these methods fail to detect or mistakenly hallucinate people or crowds in different places and under what circumstances it happens. We investigate how these methods perform with different illumination, whether they fail in dimly-lit areas or direct sunlight, measure their accuracy in sparsely-populated and densely-packed areas, and investigate whether these techniques hallucinate heads or crowds in blob-like shapes in the picture.

We test a few selected neural networks with published and reasonably readable code and either published pre-trained weights or within our computational capacity to train. We could reproduce ten different approaches, nine of which are density-based methods, and a single one is point-based. Unfortunately, we could not reproduce any methods that use the transformer architecture. Most works either did not publish their code and weights or behaved unpredictably with the published code and weights, which was the case for the LoViTCrowd and CLTR methods [77, 80].

We tested on a remote machine with a Xeon E5-2699 v3 CPU, Tesla P100 16GB GPU, and 32GB RAM, running Ubuntu 22.04. The main struggle, however, was often not computational power but the number of different methods tested. Each might require different preprocessing for each dataset, and the published source codes are poorly documented. This task was made much easier thanks to the *C3 Framework*, which implements some baseline models with the required preprocessing. However, it is not longer maintained since 2019, so newer works had to be recreated separately [81].

During the latter stages of working on this thesis, additional resources in the form of the public leaderboard on the NWPU-Crowd dataset came to light, which evaluates over 80 different approaches in-depth [1]. We use this data to perform further statistical analysis on many different state-of-the-art methods, properly evaluate the current field of crowd counting, and provide insights into the success of some methods over others.

4.1 Selected Models

In our work, a total of ten models were selected for benchmarking. Eight are implementations from the C3 Framework, and two more were recent publications selected as two original implementations from authors of these methods, one density-based and one point-based. Although the models implemented in the C3 Framework are not necessarily state of the art anymore and are only pre-trained on the synthetic GCC dataset, they are used to illustrate different architectural designs to provide an intuitive insight into their performance.

4.1.1 C3 Framework Models

4.1.1.1 AlexNet

A baseline model designed from the AlexNet architecture [44], with a few changes to be more suitable for this work. The padding operations are modified so that the output from the feature encoder part is 1/16 of the original image size. The decoder comprises two convolutional and one up-sampling layer, producing the estimated density map [81].

4.1.1.2 VGG Models

Two VGG-based models are implemented in the C3 Framework and differ only in the decoder. The first ten convolutional layers from the VGG architecture are used as an encoder. On top of that, a decoder is built similarly to the AlexNet model by using two convolutional layers. The other VGG-based model employs three additional *transposed convolution*[17] layers, which could be thought of as a trainable upsampling, to produce a more precise density map. The second model will be denoted as `VGG_decoder` [81].

4.1.1.3 ResNet Models

A total of three ResNet-based models are implemented in the C3 Framework. The `Res50` and `Res101` models are slightly modified ResNets with the corresponding layer counts, with modified strides to preserve scale at the final density map. On top of both models, a simple two-layer decoder is built.

The third model, denoted as `ResSFCN-101` or `SFCN+` is an implementation of the *Spatial Fully Convolutional Network* introduced in [69], which adds dilated convolutions and a *spatial encoder* to the top of the backbone, from which it regresses the final density map. The spatial encoder presents a convolution sequence in four directions (down, up, left-right, and right-left).

[17] Note: True inverse convolution operation is rarely used in CNNs. Instead, transposed convolution operations are used [82].

4.1.1.4 CSRNet

Implementation of a dilated convolutional network specifically designed for crowd counting [73]. In the original work, the authors argue that standard convolutional filters combined with pooling layers result in limited receptive fields and too significant reduction of spatial resolution, which needs to be upsampled with *deconvolutional* layers in the decoder. They dilate convolutions to increase their receptive fields to 5×5 and 7×7 while keeping the 3×3 weights per kernel. The base model is VGG-16, with only three pooling layers instead of five. Their resulting density map is 1/8 of the input size, which they bilinearly interpolate with a factor of 8.

4.1.1.5 SANet

The *Scale Aggregation Network* (SANet) is heavily inspired by the Inception network [50]. To accurately count at different scales, they implement and concatenate kernels of size $1, 3, 5, 7$ in each of the four encoder layers and transposed convolutions in the decoder for upscaling [83].

4.1.2 SASNet

As described in Section 3.0.2, the SASNet architecture implements a complete feature pyramid and learns the internal correspondence between different feature scales during end-to-end training. The authors also propose a loss function to supervise the network at all pyramid levels [72]. The source code published is, however, incomplete. The loss and training scripts were withheld from the repository. Researchers attempting to validate this method that managed to obtain the code from the authors describe the model as very difficult to train with convergence issues [84]. We are only provided with pre-trained weights from ShanghaiTech part A and ShanghaiTech part B datasets. The model's architecture can be seen in Figure 3.9.

4.1.3 P2PNet

Despite being from the same authors as the SASNet, the P2PNet is published with all source codes and pre-trained weights. The architecture of this model can be seen in Figure 3.11. Also described in Section 3.0.3, this model is the only point-based method we could validate and train. The P2PNet is also, at the time of writing this work, the highest-scoring model with complete published source code in the NWPU-Crowd benchmark leaderboard. All methods with better scores are, at this moment, closed-source. For this reason, this model will be the primary model to validate our experiments on.

Model	Total Params	Total Mult-Adds	Required Memory F/B Pass (MB)
AlexNet	2,502,721	986.21 M	5.87
VGG	7,701,057	18.32 G	139.47
VGG_decoder	8,397,377	20.47 G	154.66
Res50	8,674,625	9.75 G	342.89
Res101	27,666,753	29.15 G	770.71
SFCN+	38,596,801	40.41 G	787.95
CSRNet	16,263,489	27.09 G	154.67
SANet	1,388,721	5.95 G	231.21
SASNet	38,898,698	232.38 G	3367.68
P2PNet	21,579,344	26.13 G	313.66

Table 4.1: Summary of selected models, computed for an input tensor of shape (1,3,256,256).

4.1.4 Model Summary

Table 4.1 lists a few essential details about the models tested. The published research papers rarely mention how computationally efficient the models are and whether their large number of parameters translates into a well-used learning capacity. The computational difficulty significantly impacts real-world scenarios in which such models could be used. Many of the listed models would be impossible to run with reasonable inference times on any low-power embedded hardware in CCTV cameras, where such technology could be of great use. This table also shows why researchers had difficulties training the SASNet architecture, as their implementation of the feature pyramid is computationally inefficient and excessive. For this reason, we resized all images in UCF_CC_50 and UCF_QNRF datasets so that the longer side has no more than 1024 pixels, with the aspect ratio preserved. Resized images were used in all benchmarks in Table 4.2 for a fair comparison of different approaches.

As can be seen in Table 4.2, the pre-trained models from C3 Framework perform significantly worse than newer approaches in the form of SASNet and P2PNet, which can be attributed to multiple causes. First, those models are only trained on synthetic data with rich colors and less-detailed features than real-world pictures. We observed inferior performance on grayscale images across all of these models, which is most noticeable on the grayscale-only UCF_CC_50, compared to P2PNet and SASNet. All networks from the C3 Framework performed significantly better when presented with an RGB version of the same picture. In some cases, the detection failed entirely in the grayscale variants of pictures.

An important behavior to note is the difference in SASNet's performance with the weight set trained on ShanghaiTech part A or part B. As shown in Figure 3.3, part A contains dense crowds with often only overlapped heads

	SHA	SHB	UCF_CC_50	UCF_QNRF
AlexNet	201.18	26.56	1097.92	288.14
VGG	245.19	48.77	1223.74	392.10
VGG_decoder	262.32	41.61	1257.80	426.89
Res50	342.13	52.11	1231.14	581.22
Res101	181.13	27.18	1042.74	317.91
SFCN+	239.31	29.09	1259.84	318.59
CSRNet	256.76	45.77	1271.04	386.62
SANet	294.95	68.47	1037.10	470.41
SASNet*	68.30/150.01	21.95/**6.38**	357.23/757.09	212.97/277.70
P2PNet	**62.81**	22.20	**302.54**	**202.55**

Table 4.2: Baseline benchmark of selected models MAE.
**SASNet was tested with two sets of pre-trained weights, from ShanghaiTech part A and part B, and are listed in this order.*

visible, while part B contains few pedestrians with most of their bodies visible within the frame. The model fine-tuned to this smaller crowd density performed significantly better in this scenario, while the part A model performed better in any dense-crowd scenario with overlapped individuals.

This scenario can be seen in Figure 4.1, where both SASNet variants are tested, denoted as SASNet_A and SASNet_B. SASNet_A accurately predicts at all scales, while SASNet_B completely omits the middle part of the crowd, where it fails to detect the highly-overlapped heads.

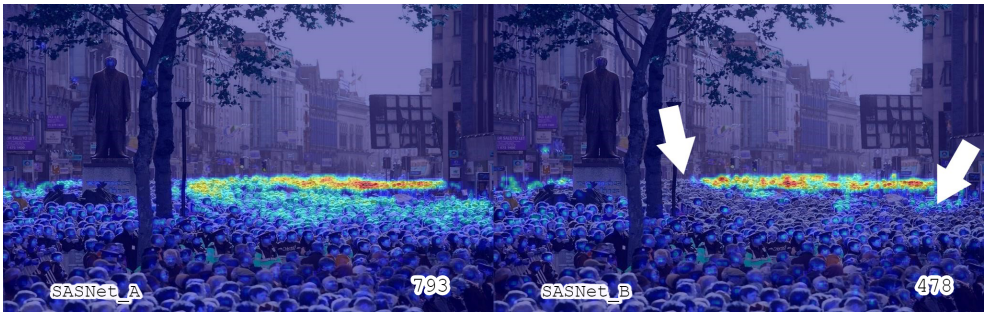


Figure 4.1: Comparison of SASNet_A and SASNet_B models. GT: 1154 people.

Another important fact is that while the GCC dataset used to pre-train the C3 Framework models contains a wide range of crowd densities, these models were not designed with sufficiently robust multi-scale detection in mind and only learned to detect crowds in limited density ranges. This behavior can be seen in an extreme example from the UCF_QNRF dataset, with a wide range of densities in the picture, as seen in Figure 4.2, where the models fail to detect at different scales.

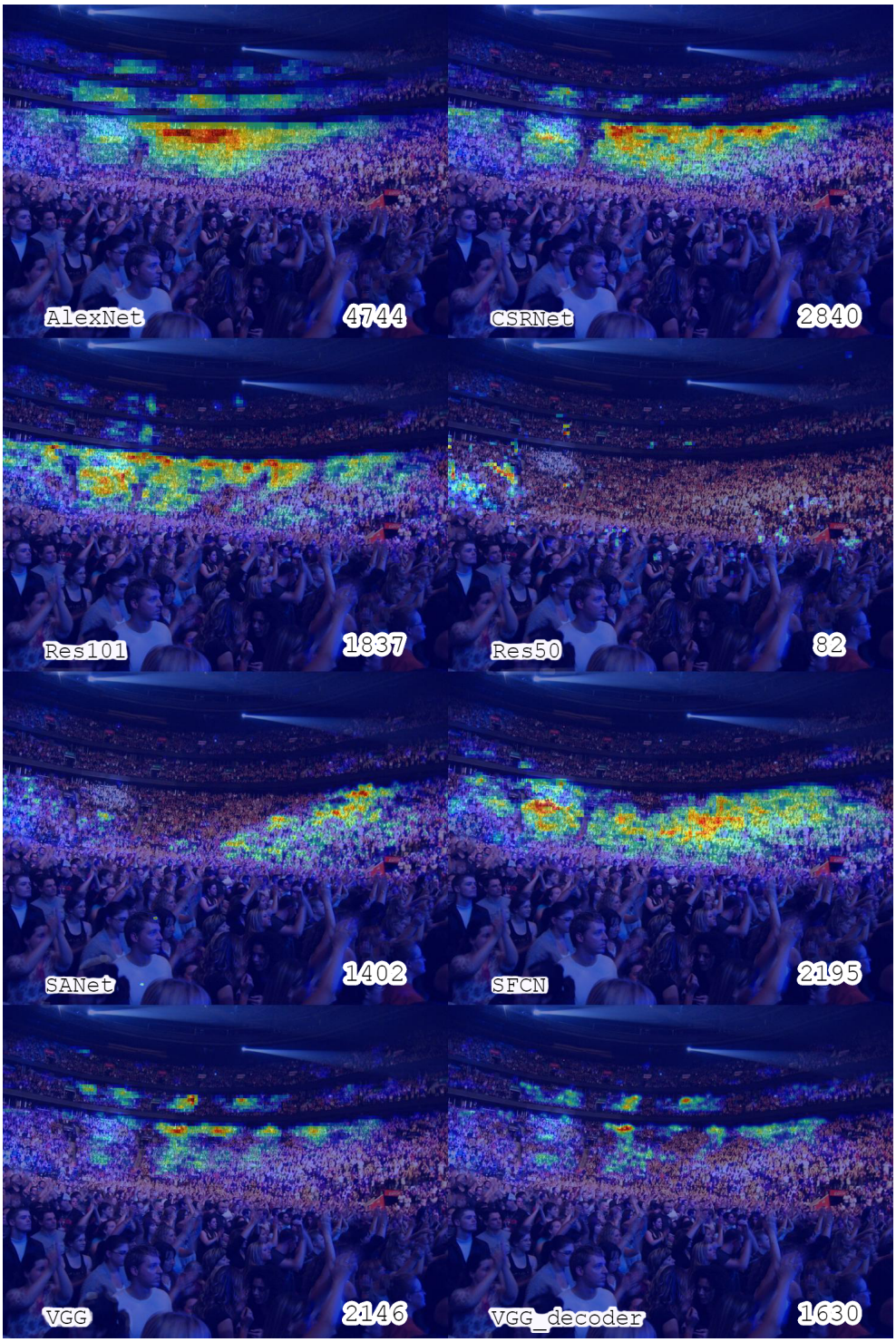


Figure 4.2: Examples of failed detection in an image from UCF_QNRF with extreme difference in scale of individuals. GT: 3566 people.

4.2 Negative Samples and False Positives

In the previous section, we have shown that these networks can fail to detect crowds of specific properties if the crowd structure is too far from what the model expects. In this section, we inspect the opposite scenario, of when the environment itself contains such a structure that the model incorrectly perceives it as a crowd. For this experiment, two different datasets were collected.

The first dataset consists only of crowd-counting negative samples – 232 images of empty outdoor and indoor spaces captured from various viewpoints during different times of the day, weather, and illumination. At first, we wanted to collect such a dataset from public web cameras. However, we had to abandon this approach, as only a few online cameras featured high enough resolution, and data with reasonable diversity were unobtainable from these. Another important consideration was the licensing of the pictures. Eventually, we decided to collect data from royalty-free image providers, such as *Unsplash.com* and *Pixabay.com*, as well as using personally taken pictures. The individual pictures collected online are documented with links and author names. In a few pictures, some individuals had to be removed using the infill feature in graphical software. All pictures were resized so that the width nor length exceeded 1024 pixels. We denote this dataset as `empty_places`.

Figure 4.3: Example images from the `empty_places` dataset.

The second dataset is a set of 200 images hand-picked from the ImageNet Challenge (ILSVRC 2010), with the very intention of tricking these networks into false positive predictions, further testing the robustness of the networks.

Those images intentionally contain high-frequency noise and lots of blob-like structures and objects, sea waves, plants, chainmail, animals, peanuts, etc. The images were carefully inspected to ensure no people were pictured in this dataset so that only false positives could be detected. We denote this dataset as `hard_images`.

Figure 4.4: Example images from the `hard_images` dataset.

4.2.1 Negative Samples

Negative samples in the `empty_places` dataset are pictures taken in areas where a crowd counter typically expects to find a crowd, yet there are zero people. In this experiment, we specifically target this scenario to test whether these crowd counters can handle the extreme case of crowd sparsity. In the same way, as crowd counters generalize poorly beyond crowd densities present in the training data, our initial hypothesis was that the same would apply to models trained on datasets with insufficient or no negative samples at all, and they would hallucinate crowds that they expect to find in every image.

Model	MAE	MSE	>0 FP Imgs	Total FP
`AlexNet`	59.82	34709.96	126/150	8973
`VGG`	5.55	271.63	85/150	833
`VGG_decoder`	5.62	172.88	146/150	843
`Res50`	2.50	41.77	72/150	376
`Res101`	**1.11**	**7.74**	**43/150**	**167**
`SFCN+`	2.43	59.84	61/150	364
`CSRNet`	6.61	201.53	135/150	992
`SANet`	25.21	2571.91	149/150	3781
`SASNet_A`	19.86	3618.07	118/150	2979
`P2PNet`	31.20	14940.89	92/150	4680

Table 4.3: False positives hallucinated in the `empty_places` dataset.

In our experiment, we notice how the smaller models `AlexNet` and `SANet` perform poorly, possibly due to their limited capacity to distinguish features of crowds from other structures. Apart from just the size of the models, where the `VGG` and `VGG_decoder` are close to `Res50`, the much deeper ResNet architecture has achieved a much smaller number of false positives. The model summary can be found in Table 4.1.

We believe this is because the deeper architecture allows the ResNet to learn more complex features than the much shallower VGG-based models, as both were trained on the same data until the same convergence criteria were met. Learning more complex features could help the model distinguish crowds from different structures in challenging scenarios.

This trend continues with the very deep `Res101` and `SFCN+` models, which have achieved the lowest false positives on negative samples. They have even outperformed much newer and more complex `SASNet` and `P2PNet`, possibly because they were trained on the GCC dataset. GCC used for training the C3 Framework models contains some negative samples, which hints at the possibility of increased robustness when the models are supervised on better data encompassing these scenarios and penalized for any false predictions, which forces the learning of more robust features. Selected examples of significant failures per model are shown in Figure 4.5.

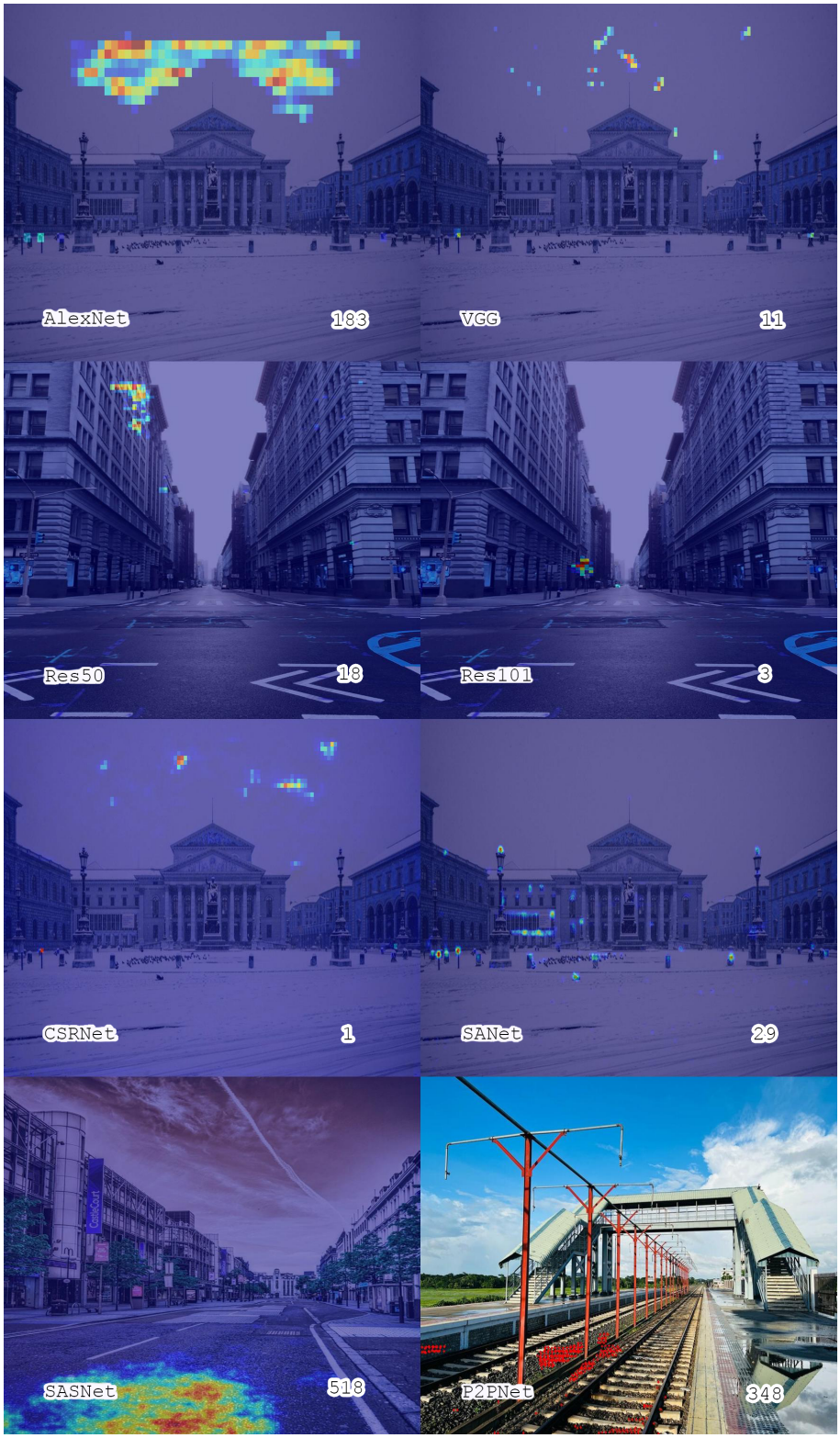


Figure 4.5: Examples of significant false positive predictions in the `empty_places` dataset.

4.2.2 High Frequency and Blob-Like Objects

With the `hard_images` dataset, we have found the models to behave similarly to the `empty_places` dataset, where the smaller `AlexNet` and `SANet` predicted high numbers of false positives. In contrast, the deeper ResNet-based models found significantly fewer false positives and have proven their superior robustness in these scenarios. They never hallucinated crowds in random noise areas, such as cloudy skies, but only in blob-like structures, where a remote resemblance to the human heads can be found. Other models hallucinated crowds of various densities in foliage, plants, stones, and grass, whereas the ResNet-based models had no issues with false positives in these pictures.

The `P2PNet` and `SASNet` models performed poorly, especially when compared to simpler VGG-based models, where we would expect at least similar performance, which again hints at the training data used for these models, as the GCC dataset is around 30× larger, as shown in Table 3.1, and captures a wider variety of scenarios, as well as some negative samples [69].

Model	MAE	MSE	>0 FP Imgs	Total FP
`AlexNet`	56.59	22238.83	166/200	11318
`VGG`	21.51	5722.86	137/200	4302
`VGG_decoder`	16.96	2533.03	166/200	3391
`Res50`	4.06	**90.97**	124/200	811
`Res101`	**3.66**	207.96	**70/200**	**731**
`SFCN+`	6.38	846.83	77/200	1276
`CSRNet`	19.59	4027.83	180/200	3918
`SANet`	66.35	17430.73	200/200	13270
`SASNet_A`	44.15	8827.94	193/200	8829
`P2PNet`	80.77	54373.84	196/200	16153

Table 4.4: False positives hallucinated in the `hard_images` dataset.

As for the magnitude of the measured errors, the only point-based approach scored the worst of all models tested. In some scenarios, the `P2PNet` predicted extremely dense crowds, with much higher total counts, than what other models hallucinated in these scenarios, as seen in the outlier-sensitive MSE metric. As it is the only point-based model we could reproduce, it is impossible to evaluate whether the model's approach is the cause. Because this model was designed to count in very dense crowds, the number of hallucinated individuals can quickly get out of hand in challenging scenarios.

The range of errors measured for the density-based methods is large. The models from the C3 Framework feature similar decoders, so we attribute this difference to the encoders of these models responsible for the feature extraction, on top of which the decoders regress the density maps with widely varying results. Using a better encoder should be the first consideration for model improvement unless a high-fidelity density map is required as an output.

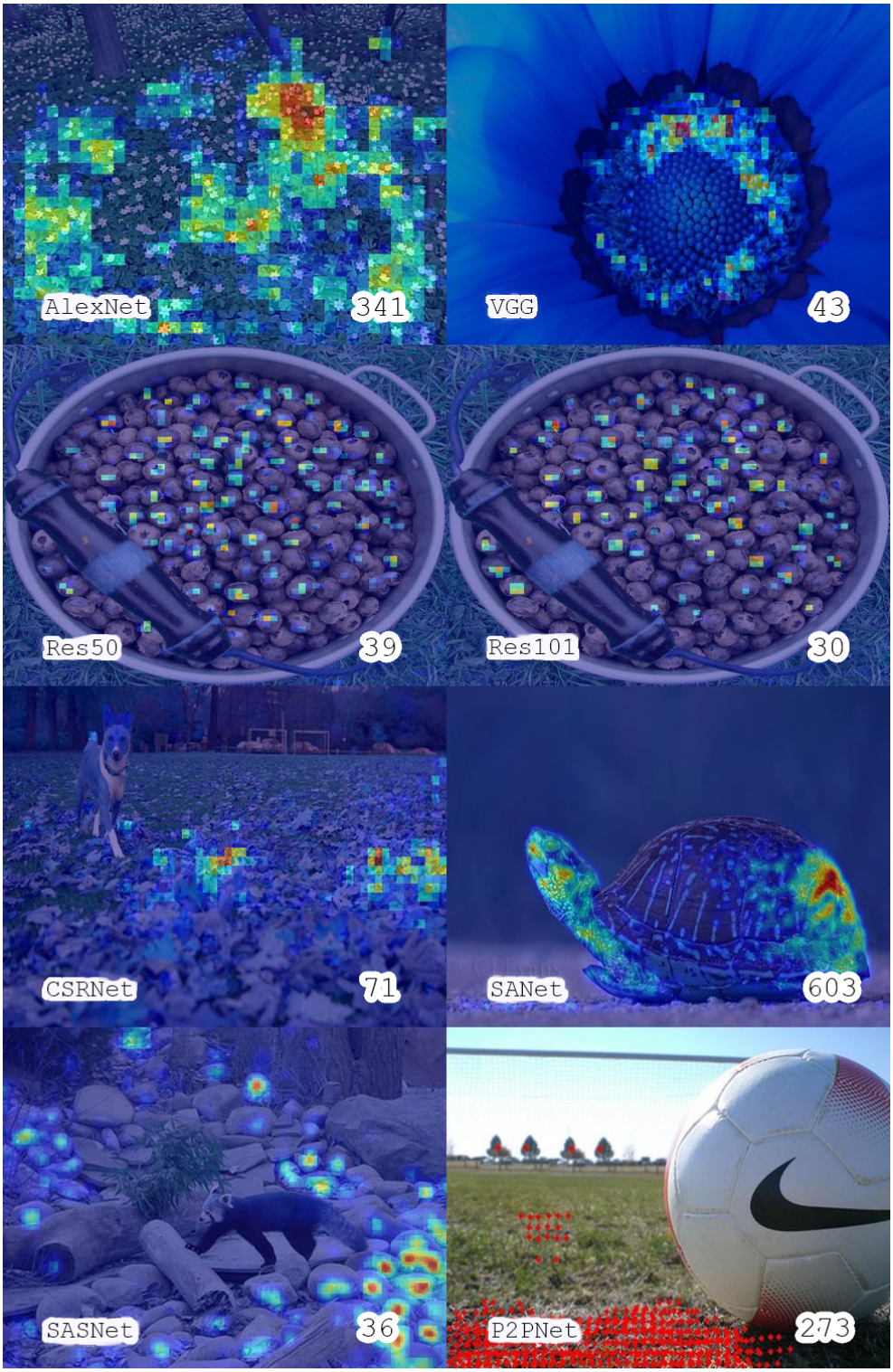


Figure 4.6: Examples of significant false positive predictions in the `hard_images` dataset.

4.2.3 Impact of the Training Data on the Model Robustness

To investigate and evaluate the impact of the training data on the final model accuracy and robustness, we have re-trained the `P2PNet` on the ShanghaiTech part A, adding 82 negative samples from the `empty_places` training set to penalize the network for false positives. The batch size was set to 16 samples, the learning rate of the prediction heads to 10^{-4} and VGG backbone to 10^{-5}. The weight decay rate was set to 10^{-4}. We trained for 500 epochs, and the validation MAE continued to improve throughout the training. We used random rescaling in $[0.7, 1.3]$ range, random horizontal flip, random crop, and random patching during the training.

	`empty_places`			
Model	**MAE**	**MSE**	**>0 FP Imgs**	**Total FP**
`SASNet_A`	19.86	3618.07	118/150	2979
`P2PNet` (SHA)	31.20	14940.89	92/150	4680
`P2PNet` (SHA+empty)	11.22	1592.50	54/150	1683
`P2PNet` (curated)	**5.80**	**989.84**	**40/150**	**870**

	`hard_images`			
Model	**MAE**	**MSE**	**>0 FP Imgs**	**Total FP**
`SASNet_A`	44.15	**8827.94**	193/200	8829
`P2PNet` (SHA)	80.77	54373.84	196/200	16153
`P2PNet` (SHA+empty)	72.64	29156.14	159/200	14528
`P2PNet` (curated)	**38.39**	28449.61	**125/200**	**7677**

Table 4.5: False positives hallucinated in the `empty_places` and `hard_images` datasets after model re-training.

The re-trained model, denoted as `P2PNet(SHA+empty)` detects fewer false positives in both the `empty_places` test set and in the `hard_images` dataset, as shown in Table 4.5. However, that does not translate into a better model overall. The model behaves as if the learned detection threshold was just set higher. It hesitates to detect when in doubt and fails to accurately detect occluded individuals in dense crowds, underestimating the total count. This can be illustrated by the abysmal performance when re-tested on the ShanghaiTech part A test set, where the model scored far worse than before, as shown in Table 4.6, often predicting much smaller counts.

Model	**ShanghaiTech A (MAE)**
`P2PNet` (SHA)	**62.81**
`P2PNet` (SHA+empty)	323.76
`P2PNet` (curated)	66.94

Table 4.6: Counting accuracy of the re-trained models.

Having so many negative samples did not improve the model performance as we expected, for which we composed a new training set with mere 30 negative samples and combined training sets of both parts of the ShanghaiTech dataset, plus the 50 samples of extremely dense crowds from UCF_CC_50. We kept the training parameters the same but decreased the batch size to 8. We have denoted this model as `P2PNet(curated)`. We trained for 500 epochs, and the best validation MAE was achieved around epoch 475.

The re-trained `P2PNet(curated)` is much more robust to negative samples, as seen in Table 4.5, while preserving the accuracy in regular scenarios, as seen in Table 4.6. This shows that for any crowd counter to be used in real-life applications, special care must be taken to curate a training set with a wide variety of different crowd densities, including negative samples, which are essential for a robust model. Furthermore, architectures with deeper encoders have shown superior robustness in our testing and should be considered for crowd counters operating in difficult conditions.

4.2.4 Note on the Importance of Image Resolution

Following the data-hungry approach, we tried to train the model on the NWPU-Crowd dataset, unfortunately, without much success. The source codes published had to be modified to enable training with negative samples, as the point matcher and the augmentation pipeline were broken in such cases. We have fixed those issues for training on negative samples and smaller crowds. However, the densities in the NWPU go far beyond what the model would encounter even in the UCF_CC_50 dataset, with up to 5× larger crowds. At this point, the `P2PNet` architecture had to be modified, increasing the number of point proposals, and the hyperparameters had to be re-tuned, as the authors did not publish theirs, used for the NWPU-Crowd benchmark.

Given our limited computational resources, we could not reach a stable convergence of the model in the few tests we conducted, as a single re-training of the model required 14 hours on the Tesla P100. Another reason for the unstable convergence could be that we had to resize the pictures so the larger side was no longer than 1024 pixels due to our limited GPU memory. The resizing had removed most details in the extremely dense crowds (up to 20 thousand people in a single picture), and the model could not distinguish the individuals and learned poor features.

According to the NWPU-Crowd benchmark website, the authors of the `P2PNet` used Tesla V100 cards, a much newer accelerator with more memory and compute cores, that could enable learning with higher resolution pictures. Unfortunately, training on such large data is beyond the reach of many researchers. However, larger crowds will require such computation to be accurate. We illustrate this in the following example.

We take the densest image from the NWPU-Crowd set as an example. With the resolution of 4032×3024 pixels and $20,033$ annotated individuals,

in an ideal scenario with an even distribution of individuals, the model would have 407 pixels per individual to identify. In the resized image to fit our memory, with the resolution of mere 1024×768 pixels, we have, at best, 39 pixels per individual, which is insufficient for accurate detection. For this reason, accurate crowd counting in large crowds will require better data.

We have implemented a patched inference pipeline for high-resolution images to test this hypothesis. Instead of resizing to a smaller resolution, the image is split into multiple smaller patches that fit into our memory. The resulting prediction is stitched, and counts are summed. The only problem arises at the patch boundaries, where counting is inaccurate. This can be partially alleviated by counting in overlapped patches and averaging the overlaps.

We have tested this approach with both the `P2PNet(mix)` and `SASNet_A` and used a patch size of 512×512 pixels. Table 4.7 shows the results of this experiment. Both models were able to detect in regions where they previously could not, as shown in Figures 4.7 and 4.8. We can see that the prediction is still far from perfect, but this is an extreme scenario, with a density far beyond what the models were trained on.

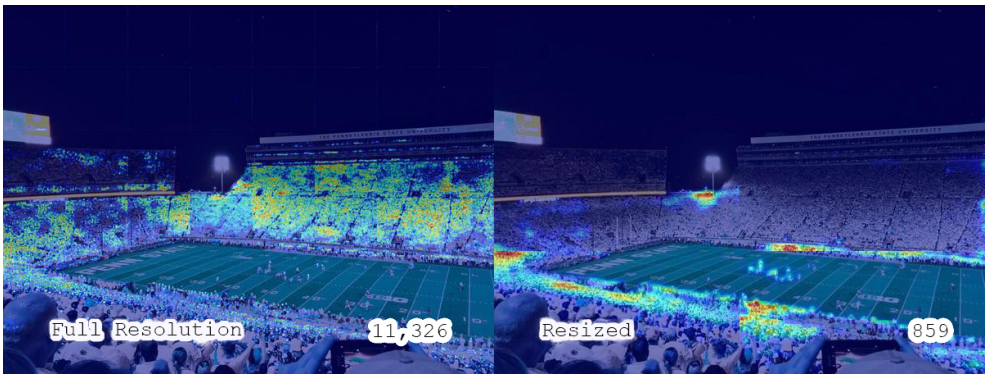


Figure 4.7: Illustration of the impact of image resolution on `SASNet`.

Figure 4.8: (Detail) Illustration of the impact of image resolution on `P2PNet`.

However, the difference is evident; the high resolution is necessary for accurate counting. A similar patched approach could be used to train the models to utilize all the original data without such significant memory demands. We could, however, not test this, as the original data would require close to a week to train with our resources. The key takeaway is that high-resolution images can still be processed on weaker devices at the cost of increased inference time. In cases where latency is not critical, this enables high-accuracy counting in challenging scenarios.

	4032×3024 px	1024×768 px
`SASNet_A`	11,326	859
`P2PNet` (mix)	10,303	1,462

Table 4.7: Total people detected with different image resolutions, out of the total 20,033 annotations.

4.3 Weather Conditions and Luminance

A robust crowd counter must count accurately under challenging weather conditions. Rain, fog, bright sun, or shadows can affect the crowd counter's accuracy adversely. To evaluate the performance during these conditions, however, we would need a very specialized dataset, and the annotation of such images in reasonable quantity would be very labor-intensive. For this reason, we have decided on augmentation of already annotated images, which we alter in a way that preserves labels. The rain, shadow, and brightness shift augmentation code was based on an open-source library [85]. During the augmentation, random polygon shadow, heavy rain with variable slant, fog implemented as Gaussian-smoothed white noise, and illumination change implemented as HSV colorspace value shift were applied to the data.

In this experiment, we selected ShanghaiTech part A as the base dataset and applied different augmentations to the entire test set beforehand. We measure the impact of these conditions on the two most accurate models, the `SASNet` and `P2PNet`, to compare a density-based and a point-based method. Both models had weights fitted to the ShanghaiTech part A training set without weather augmentation.

Table 4.8 illustrates the loss of accuracy during different conditions. Altering the brightness had a minor impact on counting accuracy, possibly because the pictures remained sharp and focused. We have observed different behavior with pictures from online cameras. The image quality suffers significantly from the small sensor size, as the cameras use high ISO values to compensate for the exposure. In other words, crowd counters can perform well in different illumination levels as long as the camera does not introduce too much noise.

Figure 4.9: Examples of weather augmentation methods.

Figure 4.10: Fog-introduced false positives. GT: 172 people.

Fog and rain resulted in significantly worse accuracy, as they obscure most details in the scene. The density-based method has a slight advantage here, possibly because these predict a crowd density instead of individuals, which can prove more robust with very little detail left in the scene. Fog and rain resulted in the inability to count and introduced many false positives into the predictions, as seen in Figure 4.10.

To alleviate these issues, we experimented with re-training of the `P2PNet` on ShanghaiTech part A dataset, with 1/6 probability of every augmentation type, including no augmentation, batch size set to 16 samples, learning rate of prediction heads 10^{-4} and VGG backbone 10^{-5}. The weight decay rate was set to 10^{-4}. The lowest validation MAE was achieved after 100 epochs. We trained for 600 epochs, which continued to improve the training loss, but only due to gradual overfitting. Checkpoint weights with the lowest achieved validation MAE were used for benchmarking.

This approach did lessen the error during the rainy weather but at the cost of performing worse overall. Upon inspection of the test results and predictions of the model, the performance was abysmal, as the predicted heads were simply random in any higher-frequency parts of the image. Overall, this approach was

not successful. Again, the possible cause is the lack of detail in the pictures caused by the augmentation methods, as the rain and fog remove too much detail in dense parts of the image to base any reasonable predictions upon. The model was forced to learn features for detection in such areas, which resulted in bad learned features and inferior accuracy in most scenarios.

	Original	Bright	Dark	Fog	Rain	Shadow
SASNet_A	68.30	71.75	83.45	**99.45**	281.44	71.86
P2PNet	**62.81**	**66.17**	**74.57**	101.72	302.86	**65.29**
P2PNet_aug	125.11	127.01	118.82	144.95	**128.26**	128.15

Table 4.8: Model accuracy in challenging weather conditions (MAE).

There is no arguing that physics poses certain limits on how much detail can even be captured with a camera, which sets an upper limit on counting accuracy in certain weather conditions. When classic RGB cameras might not be enough, adding thermal cameras could benefit the crowd counting, as seen in a relatively recent trend of RGB-T crowd counting with both CNN [86] and transformer architectures [87, 88].

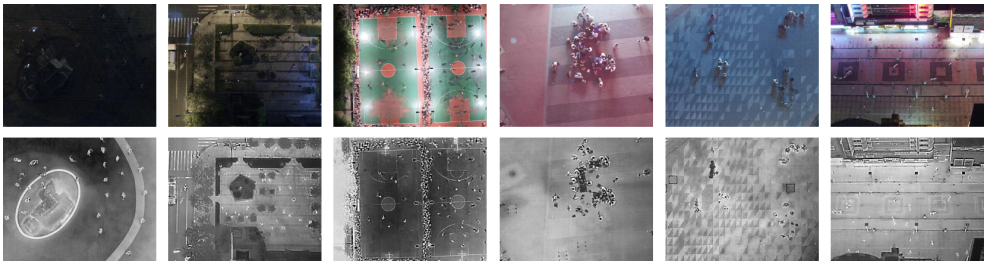


Figure 4.11: Examples of image pairs from the DroneRGBT dataset [89].

Thermal cameras do not require any illumination and could enable crowd counters to work at nighttime without additional light. This does, however, not solve rainy conditions, as the water absorbs infrared radiation from objects. Heavy fog also decreases the effective receptive range of a thermal camera. However, light fog can still be transparent to infrared radiation for long enough distances to be used in this scenario. The only drawback of this approach is the much higher price for such a system.

To summarize, the input image quality is as important as the model itself. During heavy rain or fog, the image quality decreases significantly, and the lack of detail in the resulting picture results in inaccurate counting. As for the illumination, the models performed well in both bright and dark scenarios if the image was still sharp and in focus. If a worse quality image was supplied, with significant digital noise or very low contrast, the model was at a disadvantage from the beginning and did not perform well.

4.4 Analysis of the Trends in Crowd Counting Research

A convolutional network or a transformer? Most computer vision researchers in recent years probably faced this question. The transformer architecture has shown great ability to model long-range dependencies in the visual data, thanks to global self-attention. It is, however, still a very recent approach, and the models are resource-heavy, making optimization difficult with the limited computational resources of many researchers. CNNs, on the other hand, are a long-established method and are well-understood and optimized.

When the CNN architecture is designed as fully convolutional, it can take an image of arbitrary size as an input. In contrast, transformers require preprocessing through patching and positional encoding or a convolutional encoder. Despite being challenging to implement and use, these transformer models have recently proven superior in many computer vision tasks.

This dilemma gave birth to a wide variety of hybrid architectures, which combine convolutions with attention in many different ways, either using convolutional backbone in an encoder-decoder transformer model [80], in an encoder-only transformer model [77] or mix the attention blocks into an existing convolutional architecture [78]. These architectures make a comparison between the two approaches more difficult.

Thanks to the leaderboard of the NWPU-Crowd benchmark, we can compare many different works and their performance neatly. For this, we have developed a simple web scraper that has downloaded the achieved scores in different categories of benchmarks. This data, however, was insufficient on its own, as it does not contain information about the model's architecture and the approach type, whether the model is point-based or density-based. From the 87 submitted results, we have successfully identified 72 methods and their corresponding papers to obtain this data. We have also sorted the methods by the year they were published.

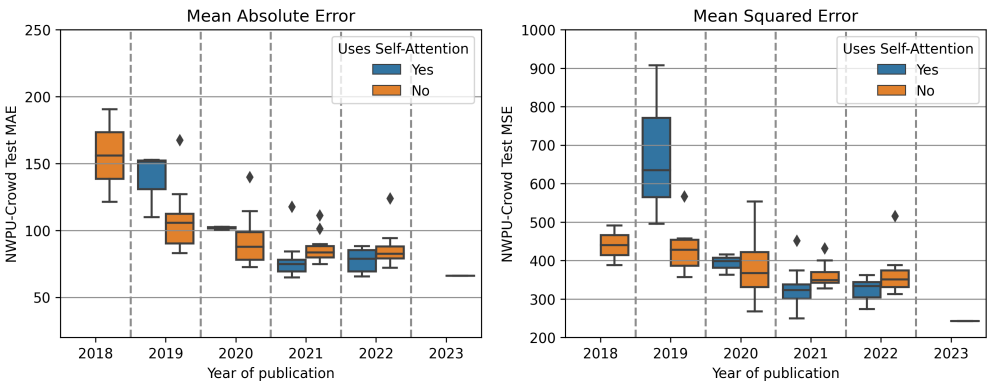


Figure 4.12: Achieved test metrics in the NWPU-Crowd benchmark per year of publication.

As for the classification of the approach type, we have considered two scenarios: convolutional networks, and networks built around self-attention, that include the abovementioned hybrid architectures. We do so to measure the impact of self-attention's holistic, long-range dependency modeling capabilities and whether they benefit crowd counting.

Figure 4.12 shows the achieved test metrics. It is clear that over the years, the methods have improved overall. The first published methods using self-attention blocks scored poorly in the MSE metric but have recently matured and outperformed CNN approaches. In the downloaded data, only a single method published during 2023 was benchmarked, a convolutional-only point-based network designed initially for tracking wildlife mammals that can also be used for crowd counting [90].

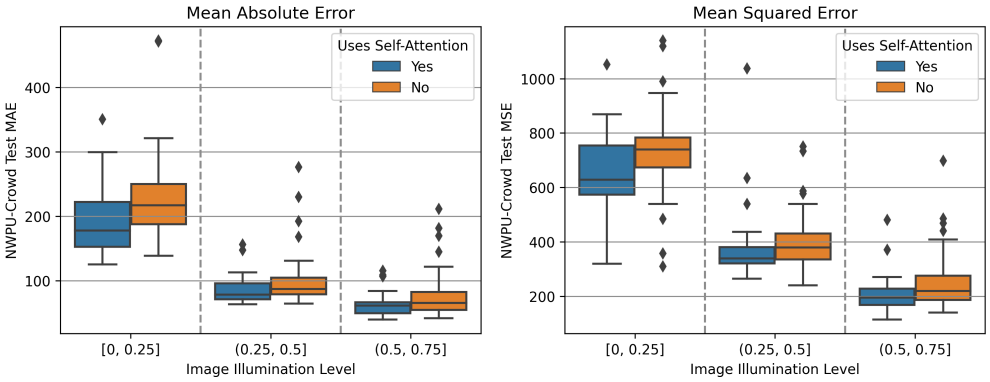


Figure 4.13: Achieved test metrics in the NWPU-Crowd benchmark per level of illumination.

Figure 4.13 shows that self-attention improves performance in different luminance levels. The most significant difference is during low-light scenarios, where the global context enables better density estimation in hard-to-discern regions with insufficient local details. Overall, incorporating self-attention into the network architecture is beneficial.

Upon inspection of whether self-attention improves accuracy in specific crowd densities, we have found only minor differences, as shown in Figure 4.14. For example, we have expected better robustness to negative samples, but self-attention yielded no significant improvements. This table also illustrates what accuracy can be achieved in different crowd density scenarios. Note the logarithmic y-scale in the plot.

We have also tested whether methods that can localize individuals instead of estimating crowd density have higher counting accuracy. To clarify, some methods marked as crowd localization methods are modified density-based methods to allow for precise individual localization, such as the FIDTM [79], which we consider crowd localization method, as they meet the criterion that individuals are distinguishable in the final prediction.

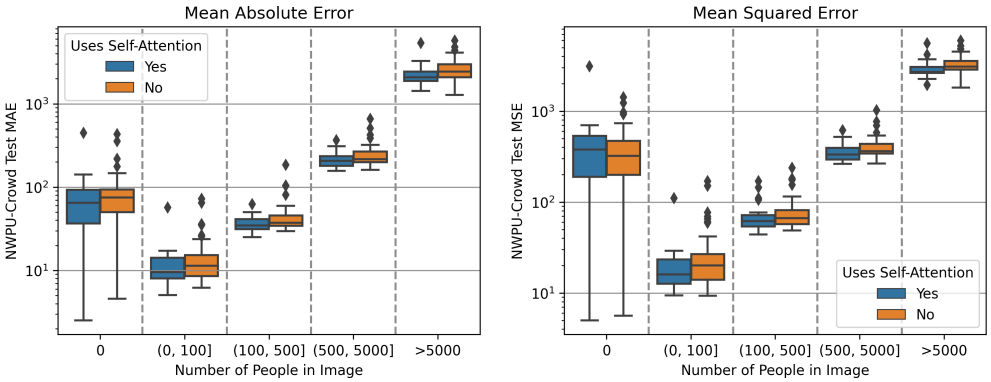


Figure 4.14: Achieved test metrics in the NWPU-Crowd benchmark per level of crowd density.

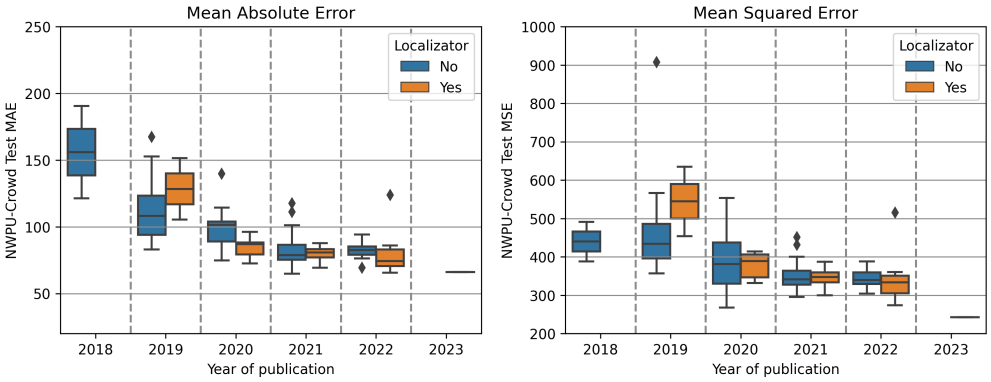


Figure 4.15: Comparison of density-based and point-based methods in the NWPU-Crowd benchmark.

Although more challenging to design, Figure 4.15 shows that these methods have recently begun to outperform pure density-based methods, possibly due to their better ability to distinguish individuals in challenging scenarios, which is most likely the result of forced learning of better individual-detecting features rather than crowd-density features, enabling more accurate counting. The resulting prediction format can also benefit applications like crowd flow tracking, which is challenging with density-based methods.

To summarize, the analysis of the NWPU-Crowd benchmark results shows that self-attention benefits the accuracy of a crowd counter, which is most noticeable in low-light scenarios with insufficient levels of detail present in local regions, where global context enables better accuracy. We have, however, not observed any significant increase in robustness to negative samples nor accuracy in extremely dense crowds, possibly due to a lack of image detail in the pictures, as mentioned earlier in this work. Furthermore, we have also observed that crowd localization methods achieve lower errors than purely density-based approaches that do not consider the localization of individuals.

Discussion

In this work, we have evaluated the current trends in crowd counting and how the recent approaches perform in different scenarios. It is clear, that in the datasets we currently use, with annotated heads of individuals, newer models achieve higher accuracy, but does that make them suitable for use in real-world scenarios?

In real-world applications, the faced problem can be eased with application-specific knowledge, which can filter the data to increase accuracy. One such approach can be seen in [67], where *regions of interest* (ROI) are defined in the data – areas where crowds can be detected. This approach can be used effectively with any camera with a fixed view and location, which could be the case for CCTVs. Such ROI can eliminate hallucinated people in areas that would not make sense, such as the sky, as seen in the `empty_places` examples in Figure 4.3, making for a more robust crowd counter.

Figure 4.16: Example of the MOG2 background subtraction algorithm from the OpenCV library [91].

We have experimented with background-subtraction methods for automated ROI generation, such as the k-NN and MOG2 algorithms implemented in the OpenCV library [91], an example of which is shown in Figure 4.16.

These algorithms work on a frame sequence, detecting movement via pixel changes. For this reason, they could not be used on pictures from our other datasets.

During our testing, however, these methods introduced much noise, and even minor changes in the sunlight could throw the results off for multiple frames. Our experiments were conducted on CCTV footage with a single frame per second, which was not enough to stabilize the background model, possibly due to compression artifacts in the video stream, which the algorithm considered a movement. These algorithms also run on CPU only, increasing the inference time of the crowd counter with a significant delay on every input image.

Predefined ROI work very well for real-world applications, and we were unsuccessful in reasonably automating this process. To some extent, the self-attention mechanism could achieve the same, attending to the relevant regions of the image, which we could not test due to poorly documented published models and lack of computational power. There lies some potential for future work, determining whether newer semantic segmentation techniques could benefit crowd counting and improve the current models by automatically generating ROI masks to eliminate false positives in the background.

Another problem of all current crowd-counting methods we have identified is handling scattered occlusions. In rainy weather, people wear umbrellas; during rallies, people march with picket signs. All these occlusions scattered throughout the crowd make accurate counting difficult, and only counting unoccluded individuals is inaccurate.

Figure 4.17: Examples of scattered occlusions in crowds [92].

We found only a single work that identifies and tackles this issue by using custom data, as no public datasets with annotated occlusions exist to our knowledge at the time of writing this thesis [4]. Their architecture uses three density prediction heads for crowd, umbrellas, and picket signs, which they merge into a final density prediction and count. Sadly, we could not obtain the data or source codes from the authors, so the robustness of such an approach could not be evaluated in this thesis. Despite this, there is potential for improvement in real-world use cases, as our tested methods have all performed poorly in these specific scenarios.

Conclusion

In this work, we evaluated the current state of the art in crowd counting regarding accuracy, reliability, and robustness. We have done so by tracking key accuracy metrics in various challenging scenarios that can occur during real-world applications. We have also set out to conduct experiments that would lead to further insights on eliminating false positives via improved training data, and those resulted in more significant findings.

We have reproduced ten different works to evaluate the impacts of architectural design on the final performance and found that networks with multi-scale detection designs, such as a feature pyramid, were more accurate than simpler networks. We have also observed that networks trained on data in which a specific crowd density range is absent fail to detect it and do not generalize beyond ranges found in the training data.

To evaluate whether and how much different models hallucinate false positives, we have collected two different challenging datasets, one containing empty outdoor and indoor areas and the second containing objects and scenes with blob-like structures remotely similar to crowds. Initially, our goal was to collect this data from online web cameras. However, a stock photography database had to be used as an alternative to achieve the desired picture quality and sufficient diversity.

In both datasets, deeper ResNet-based architectures performed superior to any shallower architectures. This suggests that architectures with deeper encoders learn more robust features than shallower architectures with similar computational complexity and should be a first consideration when selecting a backbone for a crowd counter model.

Upon addition of 82 negative samples to the training data with 300 regular samples, we have observed decreased numbers of false positive predictions, but at the cost of worse accuracy in other scenarios. With a more balanced dataset with additional regular samples from different datasets with a wide range of crowd densities and fewer negative samples, the robustness increased significantly without sacrificing nearly any accuracy.

Considering the aforementioned and the increase in accuracy after fine-tuning a model to predict a specific density range, we believe fine-tuning any crowd-counting model to the conditions in which it will be deployed will benefit the accuracy. Furthermore, when training a universal crowd counter, the range of different densities in the training data should be balanced for the optimal performance of the model.

We have also evaluated the performance of different model architectures in challenging weather conditions and have found that fog and rain are the most difficult to tackle accurately. Changes in brightness or sharp shadows altering the contrast had little impact on the accuracy. We have retrained the `P2PNet` model with an augmented dataset to enhance the model performance during challenging weather conditions. This model faced poor convergence, yet it achieved lesser average error during rain, but the predictions were seemingly random, and the accuracy was abysmal in any common scenario.

The impact of rain cannot easily be mitigated, as it removes too much detail from the picture, and there is not enough detail left for accurate counting. When designing a crowd counter, the image quality from a camera used for counting has to be adequate for the desired accuracy and encountered densities. We illustrate this with an extreme example of 20 thousand people, where poor resolution results in detection failure. We have implemented a patched inference pipeline to circumvent the memory limitations of our hardware and estimate crowds in a highly detailed photograph. With this approach, the resulting accuracy improved drastically, proving that high resolution will be necessary to count larger crowds accurately.

The most significant limitation of this thesis is the lack of transformer-based models in our evaluations. Unfortunately, we could not obtain source codes or trained weights for published transformer-based crowd-counting models, and training those requires more memory than our hardware had available. Although computationally expensive, these models are a missing piece from our architectural benchmarks on robustness and accuracy. Instead, we have focused on evaluating the published results in the NWPU-Crowd benchmark, for which we implemented a web scraper to obtain the accuracy data from various methods. We have successfully identified 72 of them, with their corresponding published research, analyzed trends in crowd counting research, and evaluated methods built around self-attention, such as the transformers, recently gaining much attention in computer vision.

From the collected data, it is clear that architectures built around self-attention have achieved higher accuracy and, most noticeably, a better low-light performance compared to purely convolutional models. We believe this is due to the self-attention long-range modeling capabilities being essential for decisions where insufficient details are present in local hard-to-discern regions. We have also found that networks that localize individuals have recently outperformed density-based approaches in crowd counting.

In future work, more focus should be paid to the importance and benefits of self-attention mechanisms in crowd-counting architectures and the possible impact of their combination with deeper convolutional encoder backbones on model robustness.

Another promising improvement lies in tackling the high resolution needed for accurate counting in extremely dense crowds. Most current methods we inspected have only downscaled any high-resolution data for training, losing much information in the process. A patch-inference and training pipeline could utilize all the original data and predict accurately with arbitrary image size.

Another option would be the research of both memory-efficient and small-footprint architectures that could fit the high-resolution input into a limited memory. Candidates for such networks could be the compact convolutional transformers [55] or attention-approximating architectures that reduce the quadratic space complexity to linear, such as the Performer architecture [93].

Lastly, apart from the accuracy of the crowd counters, handling scattered occlusions is another problem to be tackled if a reliable system is to be used in real-world scenarios. By counting occlusions and the context of where they appear, it is possible to estimate the number of fully occluded individuals, contributing to a more accurate total count.

With these improvements, we envision the development of more accurate, robust, and reliable crowd-counting models that can handle a variety of challenging scenarios, including challenging weather conditions and both sparse crowds and large crowds. These models could find numerous applications in crowd management, security, and event planning. Efficient models that handle high-resolution input data with limited computational resources could open new possibilities for real-world applications.

Bibliography

[1] Wang, Q.; Gao, J.; et al. NWPU-Crowd: A Large-Scale Benchmark for Crowd Counting and Localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 43, 2020: pp. 2141–2149.

[2] Idrees, H.; Tayyab, M.; et al. Composition Loss for Counting, Density Map Estimation and Localization in Dense Crowds. In *European Conference on Computer Vision*, 2018.

[3] Zhang, Y.; Zhou, D.; et al. Single-Image Crowd Counting via Multi-Column Convolutional Neural Network. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016: pp. 589–597.

[4] Almalki, K. J.; Choi, B.-Y.; et al. Characterizing Scattered Occlusions for Effective Dense-Mode Crowd Counting. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021: pp. 3833–3842.

[5] Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers. *IBM J. Res. Dev.*, volume 44, 1967: pp. 206–227.

[6] Hierons, R. M. Machine learning. *Software Testing, Verification & Reliability*, volume 9, 1999: pp. 191–193.

[7] Jumper, J. M.; Evans, R.; et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, volume 596, 2021: pp. 583 – 589.

[8] Brown, T. B.; Mann, B.; et al. Language Models are Few-Shot Learners. *ArXiv*, volume abs/2005.14165, 2020.

[9] Ye, W.; Liu, S.; et al. Mastering Atari Games with Limited Data. In *Neural Information Processing Systems*, 2021.

[10] Caliskan, A.; Bryson, J. J.; et al. Semantics derived automatically from language corpora contain human-like biases. *Science*, volume 356, 2016: pp. 183 – 186.

[11] Mehrabi, N.; Morstatter, F.; et al. A Survey on Bias and Fairness in Machine Learning. *ACM Computing Surveys (CSUR)*, volume 54, 2019: pp. 1 – 35.

[12] Flores, A. W.; Bechtel, K.; et al. False Positives, False Negatives, and False Analyses: A Rejoinder to "Machine Bias: There's Software Used across the Country to Predict Future Criminals. and It's Biased against Blacks". *Federal Probation*, volume 80, 2016: p. 38.

[13] Garcia, M. Racist in the Machine: The Disturbing Implications of Algorithmic Bias. *World Policy Journal*, volume 33, 2016: pp. 111 – 117.

[14] Metz, R. Why Microsoft Accidentally Unleashed a Neo-Nazi Sexbot. 2016, [visited 2023-02-21]. Available from: `https://www.technologyreview.com/2016/03/24/161424/why-microsoft-accidentally-unleashed-a-neo-nazi-sexbot/`

[15] Mohri, M.; Rostamizadeh, A.; et al. Foundations of Machine Learning. In *Adaptive computation and machine learning*, 2012.

[16] Du, K.-L.; Swamy, M.; et al. Fundamentals of machine learning. *Neural Networks and Statistical Learning*, 2014: pp. 15–65.

[17] Brownlee, J. What is the Difference Between Test and Validation Datasets? 2017, [visited 2023-02-23]. Available from: `https://machinelearningmastery.com/difference-test-validation-datasets/`

[18] Kohavi, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *International Joint Conference on Artificial Intelligence*, 1995.

[19] Brownlee, J. *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*. Machine Learning Mastery, 2020.

[20] Géron, A. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2017.

[21] Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv*, volume abs/1502.03167, 2015.

[22] Chen, G.; Chen, P.; et al. Rethinking the Usage of Batch Normalization and Dropout in the Training of Deep Neural Networks. *ArXiv*, volume abs/1905.05928, 2019.

[23] Smith, S. L.; Le, Q. V. *A Bayesian Perspective on Generalization and Stochastic Gradient Descent*. 2018.

[24] Mishkin, D.; Sergievskiy, N.; et al. Systematic evaluation of convolution neural network advances on the Imagenet. *Computer Vision and Image Understanding*, volume 161, Aug 2017: p. 11–19, ISSN 1077-3142.

[25] Chodounský, D. *Detection of COVID-19 in X-ray images using Neural Networks*. Bachelor's Thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

[26] Vadlejch, M. *Real-time Facial Expression Recognition in the Wild*. Bachelor's Thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

[27] Fortmann-Roe, S. Understanding the Bias-Variance Tradeoff. 2012, [visited 2023-02-27]. Available from: `http://courses.washington.edu/me333afe/Bias_Variance_Tradeoff.pdf`

[28] Nakkiran, P.; Kaplun, G.; et al. Deep double descent: where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, volume 2021, 2019.

[29] Jared Wilber, B. W. Double Descent. 2021, [visited 2023-02-28]. Available from: `https://mlu-explain.github.io/double-descent/`

[30] McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, volume 5, no. 4, 1943: pp. 115–133.

[31] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, volume 65 6, 1958: pp. 386–408.

[32] Ivakhnenko, A. G.; Lapa, V. G. Cybernetics and Forecasting. *Nature*, volume 219, 1968: pp. 202–203.

[33] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[34] Werbos, P. *Beyond Regression : "New Tools for Prediction and Analysis in the Behavioral Sciences*. 1974.

[35] Szeliski, R. Computer Vision - Algorithms and Applications. In *Texts in Computer Science*, 2010.

[36] Sage, K.; Young, S. J. Security applications of computer vision. *IEEE Aerospace and Electronic Systems Magazine*, volume 14, 1999: pp. 19–29.

[37] Viola, P. A.; Jones, M. J. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, 2001: pp. I–I.

[38] Navarrete, P.; del Solar, J. R. Comparative Study between Different Eigenspace-Based Approaches for Face Recognition. In *Advances in Soft Computing*, 2002.

[39] Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, 2005: pp. 886–893 vol. 1.

[40] Lowe, D. G. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, 1999: pp. 1150–1157 vol.2.

[41] Bay, H.; Tuytelaars, T.; et al. SURF: Speeded Up Robust Features. In *European Conference on Computer Vision*, 2006.

[42] Cortes, C.; Vapnik, V. N. Support-Vector Networks. *Machine Learning*, volume 20, 1995: pp. 273–297.

[43] Cover, T. M.; Hart, P. E. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory*, volume 13, 1967: pp. 21–27.

[44] Krizhevsky, A.; Sutskever, I.; et al. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, volume 60, 2012: pp. 84 – 90.

[45] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, volume 36, 1980: pp. 193–202.

[46] Dumoulin, V.; Visin, F. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016, `1603.07285`.

[47] Howard, A. G.; Zhu, M.; et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv*, volume abs/1704.04861, 2017.

[48] Jarrett, K.; Kavukcuoglu, K.; et al. What is the best multi-stage architecture for object recognition? *2009 IEEE 12th International Conference on Computer Vision*, 2009: pp. 2146–2153.

[49] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, volume abs/1409.1556, 2014.

[50] Szegedy, C.; Liu, W.; et al. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[51] He, K.; Zhang, X.; et al. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015: pp. 770–778.

[52] Vaswani, A.; Shazeer, N. M.; et al. Attention is All you Need. *ArXiv*, volume abs/1706.03762, 2017.

[53] Dosovitskiy, A.; Beyer, L.; et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *ArXiv*, volume abs/2010.11929, 2020.

[54] Carion, N.; Massa, F.; et al. End-to-End Object Detection with Transformers. *ArXiv*, volume abs/2005.12872, 2020.

[55] Hassani, A.; Walton, S.; et al. Escaping the Big Data Paradigm with Compact Transformers. *ArXiv*, volume abs/2104.05704, 2021.

[56] Panda, A. K. Concepts about Positional Encoding You Might Not Know About. 2021, [visited 2023-03-14]. Available from: `https://towardsdatascience.com/concepts-about-positional-encoding-you-might-not-know-about-1f247f4e4e23`

[57] Chen, L.; Varoquaux, G.; et al. UNDERSTANDING THE ROLE OF POSITIONAL ENCODINGS IN SENTENCE REPRESENTATIONS. 2023. Available from: `https://openreview.net/forum?id=8taH4yjN62m`

[58] Gehring, J.; Auli, M.; et al. Convolutional Sequence to Sequence Learning. 2017, `1705.03122`. Available from: `http://arxiv.org/abs/1705.03122`

[59] Xiao, T.; Singh, M.; et al. Early Convolutions Help Transformers See Better. In *Neural Information Processing Systems*, 2021.

[60] Wu, H.; Xiao, B.; et al. CvT: Introducing Convolutions to Vision Transformers. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021: pp. 22–31.

[61] Chaudhary, R.; Zehra, N.; et al. Estimating density of leopard (Panthera pardus fusca) using spatially explicit capture recapture framework in Gir Protected Area, Gujarat, India. *Biologia*, volume 78, 2022: pp. 487 – 495.

[62] Benaggoune, K.; Masry, Z. A.; et al. A deep learning pipeline for breast cancer ki-67 proliferation index scoring. 2022, 2203.07452.

[63] Liu, S.; Baret, F.; et al. A method to estimate plant density and plant spacing heterogeneity: application to wheat crops. *Plant Methods*, volume 13, 2017.

[64] Neupane, B.; Horanont, T.; et al. Real-Time Vehicle Classification and Tracking Using a Transfer Learning-Improved Deep Learning Network. *Sensors (Basel, Switzerland)*, volume 22, 2022.

[65] Šarbortová, H. Počítání individuálních osob v částečně zaplněné scéně. *ČVUT FEL, Diploma Thesis*, 2015.

[66] Song, Q.; Wang, C.; et al. Rethinking Counting and Localization in Crowds: A Purely Point-Based Framework. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021: pp. 3345–3354.

[67] Zhang, C.; Li, H.; et al. Cross-scene crowd counting via deep convolutional neural networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015: pp. 833–841.

[68] Idrees, H.; Saleemi, I.; et al. Multi-source Multi-scale Counting in Extremely Dense Crowd Images. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013: pp. 2547–2554.

[69] Wang, Q.; Gao, J.; et al. Learning from Synthetic Data for Crowd Counting in the Wild. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8198–8207.

[70] ShanghaiTech Dataset. https://paperswithcode.com/dataset/shanghaitech, accessed: 2023-03-26.

[71] Xiong, H.; Lu, H.; et al. From Open Set to Closed Set: Counting Objects by Spatial Divide-and-Conquer. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019: pp. 8361–8370.

[72] Song, Q.; Wang, C.; et al. To Choose or to Fuse? Scale Selection for Crowd Counting. In *AAAI Conference on Artificial Intelligence*, 2021.

[73] Li, Y.; Zhang, X.; et al. CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018: pp. 1091–1100.

[74] Dai, J.; Qi, H.; et al. Deformable Convolutional Networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017: pp. 764–773.

[75] Wei, X.; Kang, Y.; et al. Scene-Adaptive Attention Network for Crowd Counting. *ArXiv*, volume abs/2112.15509, 2021.

[76] Yang, S.; Guo, W.; et al. CrowdFormer: An Overlap Patching Vision Transformer for Top-Down Crowd Counting. In *International Joint Conference on Artificial Intelligence*, 2022.

[77] Tran, N. H.; Huy, T. D.; et al. Improving Local Features with Relevant Spatial Information by Vision Transformer for Crowd Counting. In *British Machine Vision Conference*, 2022.

[78] Qian, Y.; Zhang, L.; et al. Segmentation Assisted U-shaped Multi-scale Transformer for Crowd Counting. In *British Machine Vision Conference*, 2022.

[79] Liang, D.; Xu, W.; et al. Focal inverse distance transform maps for crowd localization. *IEEE Transactions on Multimedia*, 2022.

[80] Liang, D.; Xu, W.; et al. An end-to-end transformer model for crowd localization. *European Conference on Computer Vision*, 2022.

[81] Gao, J.; Lin, W.; et al. C^3 Framework: An Open-source PyTorch Code for Crowd Counting. *arXiv preprint arXiv:1907.02724*, 2019. Available from: `https://github.com/gjy3035/C-3-Framework`

[82] Paszke, A.; Gross, S.; et al. CONVTRANSPOSE2D, PyTorch Library Documentation. `https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html`, accessed: 2023-04-02.

[83] Cao, X.; Wang, Z.; et al. Scale Aggregation Network for Accurate and Efficient Crowd Counting. In *European Conference on Computer Vision*, 2018.

[84] Song, Q.; Wang, C.; et al. TencentYoutuResearch/CrowdCounting-SASNet/issues. `https://github.com/TencentYoutuResearch/CrowdCounting-SASNet/issues`, 2021, accessed: 2023-04-02.

[85] Saxena, U. Automold. `https://github.com/UjjwalSaxena/Automold--Road-Augmentation-Library`, 2018, accessed: 2023-04-07.

[86] Gu, S.; Lian, Z. A Unified Multi-Task Learning Framework of Real-Time Drone Supervision for Crowd Counting. *ArXiv*, volume abs/2202.03843, 2022.

[87] Chen, P.; Gao, J.; et al. MAFNet: A Multi-Attention Fusion Network for RGB-T Crowd Counting. *ArXiv*, volume abs/2208.06761, 2022.

[88] Liu, Z.; liuzywen. RGB-T Multi-Modal Crowd Counting Based on Transformer. In *British Machine Vision Conference*, 2023.

[89] Peng, T.; Li, Q.; et al. RGB-T Crowd Counting from Drone: A Benchmark and MMCCN Network. In *Asian Conference on Computer Vision*, 2020.

[90] Delplanque, A.; Foucher, S.; et al. From crowd to herd counting: How to precisely detect and count African mammals using aerial imagery and deep learning? *ISPRS Journal of Photogrammetry and Remote Sensing*, 2023.

[91] Bradski, G. OpenCV Background Subtractors. `https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html`, 2000, accessed: 2023-04-23.

[92] Alex Block; James Eades; et al. Various Pictures of Occluded Crowds. `https://unsplash.com/photos/ZrrBdq0VNio`, `https://unsplash.com/photos/TOmJC_oz2s4`, `https://unsplash.com/photos/0FTASntRc2M`, accessed: 2023-04-23.

[93] Choromanski, K.; Likhosherstov, V.; et al. Rethinking Attention with Performers. *ArXiv*, volume abs/2009.14794, 2020.

Appendix A

List of Acronyms

AI Artificial Intelligence

ANN Artificial Neural Network

CCTV Closed-Circuit Television

CNN Convolutional Neural Network

CvT Convolutional Transformer

CLTR Crowd Localization Transformer

FCN Fully Convolutional Network

FP False Positive

FPN Feature Pyramid Network

GAN Generative Adversarial Network

GCC GTA5 Crowd Counting

GT Ground Truth

GPU Graphics Processing Unit

HoG Histogram of Oriented Gradients

ILSVRC ImageNet Large Scale Visual Recognition Challenge

k-NN K-Nearest Neighbors Algorithm

MAE Mean Average Error

MCNN Multi-Column Convolutional Neural Network

ML Machine Learning

MSE Mean Squared Error

MOG Mixture of Gaussians

NLP Natural Language Processing

PE Positional Encoding

P2PNet Point-to-Point Network

RMSE Root Mean Squared Error

RMSLE Root Mean Squared Log Error

ROI Region of Interest

SANet Scale Aggregation Network

SASNet Scale-Adaptive Selection Network

SFCN Spatial Fully Convolutional Network

SIFT Scale-Invariant Feature Transform

SEQ2SEQ Sequence-to-Sequence Model

SVM Support Vector Machine

SHB ShanghaiTech Part B

SHA ShanghaiTech Part A

UCF_CC University of Central Florida Crowd Counting

ViT Vision Transformer

YOLO You Only Look Once Network

Appendix B

Contents of the enclosed media

All files were submitted, with an exception of the public datasets used in this thesis, due to their size. Links to download this data for evaluation can be found in the readme file.

- `src` ..... all source files and model weights
  - `crowdbench_scraper` ..... files for crowdbenchmark.com evaluation
  - `datasets` ..... scripts for data preprocessing and data
    - `EMPTY_PLACES` ..... our `empty_places` dataset
    - `HARD_IMAGES` ..... our `hard_images` dataset
    - `scripts` ..... data preprocessing scripts
      - `augmentation` ..... data augmentation scripts
  - `networks` ..... source codes of implemented networks
    - `C3_fw` ..... source codes of the C3 framework models
    - `P2PNet` ..... source codes of the P2PNet model
    - `SASNet` ..... source codes of the SASNet model
- `DP_Vadlejch_Martin_2023.pdf` ..... this thesis in the PDF format
- `DP_Vadlejch_Martin_2023.zip` ..... LaTeX source files for this thesis
- `README.md` ..... further description of all the submitted files