BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

COUNTING CRATES IN IMAGES

POČÍTÁNÍ PŘEPRAVEK V OBRAZECH

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **PETR MIČULEK**
AUTOR PRÁCE

SUPERVISOR **Prof. Ing. ADAM HEROUT, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2020/2021

Bachelor's Thesis Specification

Student: **Mičulek Petr**
Programme: Information Technology
Title: **Counting Crates in Images**
Category: Image Processing

Assignment:

1. Familiarize yourself with the topic of computer vision with a strong focus on modern convolutional neural network architectures.
2. Create a dataset of simulated data for prototyping algorithms of image-based counting of crates (or similar objects).
3. Carry out experiments with the dataset. Iteratively improve the solution.
4. Design an image-based object counting solution for crates.
5. Implement the object counting solution, evaluate/test the solution on appropriate data.
6. Evaluate the results and suggest further improvements to the project. Create a poster and a short video for presenting the project.

Recommended literature:

- Bharath Ramsundar, Reza Bosagh Zadeh: TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning, O'Reily Media, 2018
- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011

Requirements for the first semester:

- items 1 through 3

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

Supervisor: **Herout Adam, prof. Ing., Ph.D.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2020
Submission deadline: May 12, 2021
Approval date: March 22, 2021

Abstract

This thesis deals with the topic of using deep learning to count crates in images. I have designed a crate-counting solution for blocks of matchboxes, using a fully convolutional classification-based network with a high resolution output. The original project proposition counted on using a dataset of photos of crates from a beer brewery warehouse. I did not get access to the dataset in the end. On the recommendation of my supervisor, I based the crate-counting solution on a custom dataset of matchbox photos. The CNN is trained using image patches, leading to a fast solution working even on smaller datasets. Matchbox keypoints are detected by the CNN in the input images and they are processed by a keypoint estimation and crate-counting algorithm to produce the final crate count. On validation data, the solution has a 12.5 % failure rate and a MAE of 11.14. Thorough experimentation was performed to evaluate the solution and the results verify that this approach can be used for object counting.

Abstrakt

V této práci se zabývám tématem počítání beden v obrazových datech pomocí technik hlubokého učení. V práci jsem navrhl řešení pro počítání beden, které představuji na fotkách krabiček sirek. Ačkoli původní řešení počítalo s využitím datové sady beden ze skladu pivovaru, sada nakonec nebyla dodána a na doporučení vedoucího práce byly pro řešení vybrány bloky krabiček sirek. Implementované řešení využívá plně konvoluční neuronovou síť založenou na klasifikaci, umožňující výstup ve vysokém rozlišení. Tato síť je trénována na výřezech fotek z datové sady, díky čemuž je řešení rychlé a síť je vhodná i pro použití na menších datových sadách. Síť detekuje ve fotkách klíčové body krabiček sirek, které jsou následně zpracovány algoritmem pro odhad klíčových bodů z predikce sítě a výpočet finálního počtu beden. Na validačním datasetu dosahuje řešení následujících výsledků – ve $12,5$ % případů predikce selže a ve zbylých případech má průměrnou absolutní odchylku (MAE) $11,14$. Pomocí rozsáhlých experimentů bylo řešení vyhodnoceno a výsledky potvrzují, že tento přístup může být použit pro počítání objektů.

Keywords

Image Processing, Convolutional Neural Networks, Keypoint Detection, Object Counting

Klíčová slova

zpracování obrazu, konvoluční neuronové sítě, detekce klíčových bodů, počítání objektů

Reference

MIČULEK, Petr. *Counting Crates in Images*. Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Prof. Ing. Adam Herout, Ph.D.

Counting Crates in Images

Declaration

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

.....
Petr Mičulek
May 19, 2021

Acknowledgements

I would like to thank my supervisor Prof. Ing. Adam Herout, Ph.D. for his help, inspiration, and valuable feedback.

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Contents

Chapter 1

Introduction

Many surveillance cameras are nowadays used in various places but the amount of data is so large that it cannot be ever processed manually. Computer vision attempts to make sense of the visual world so that it can be analyzed automatically. The large amounts of available data have contributed to the explosion of deep learning – automatically finding solutions to complicated problems through data and learning, without having to define the problem solution explicitly. This thesis deals with the topic of estimating the number of crates in images using deep learning.

The motivation for this thesis was making use of data from surveillance cameras in a beer brewery warehouse, to keep track of the number of beer crates in the warehouse. However, the data, a key ingredient of any deep learning solution, never became available. After much waiting, the solution has been solved using similar data. On my supervisor's recommendation, a matchbox counting solution was designed and implemented. The solution can serve as a proof of concept for a future beer crate-counting solution.

This work uses convolutional neural networks for the detection of matchbox keypoints. The keypoints are then geometrically interpreted to count the matchboxes in the image. The solution is designed to be simple, fast, and lightweight. An approach using fully convolutional classification-based network has been chosen. This approach has been investigated rigorously and the proposed solution is capable of making meaningful estimates on the used data.

This thesis is divided into 6 chapters. Chapter 2 introduces the reader to the topic of Convolutional Neural Networks, then presents some of the tasks that Convolutional Neural Networks are used for and some main ideas of the task solutions. In Chapter 3, the core of this work is presented – the crate-counting solution design and the details of its implementation. In the following Chapter 4, the implemented solution is explored and thoroughly evaluated.

Chapter 2

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are the prime approach to many computer vision tasks nowadays. They have been one of the main drivers of the deep learning explosion. In the diverse spectrum of deep learning methods, there is one common principle – any task can be solved better by learning its solution from data, rather than by directly encoding prior knowledge. The broad field of deep learning is well defined by the following quote from Yann LeCun [20]:

> *Deep Learning is building a system by assembling parameterized modules into a (possibly dynamic) computation graph and training it to perform a task by optimizing the parameters using a gradient-based method.*

There are many terms used in the context of deep learning, for a simple overview of their scope, they can be categorized like this:

$$\text{CNNs} \subset \text{Neural Networks} \subset \text{Deep Learning} \subset \text{Machine Learning}.$$

This chapter introduces the reader to the topic of Convolutional Neural Networks. The description of neural networks fundamentals in this chapter is restricted to supervised learning only. Other learning paradigms are out of the scope of this work. The chapter starts with the topic of training neural networks as the most fundamental part. Then follows an introduction to the Convolutional Neural Network building blocks. At last, a few key CNN architecture elements are presented and a practical aspect of training neural networks – training on imbalanced data – is discussed.

2.1 Training Neural Networks

The training of a neural network is an optimization process that can be informally summarized as "learning by doing". During training, the network weights (parameters) are repeatedly changed until they reach values that lead to a desirable outcome. The correctness of a model prediction is evaluated by a loss function (objective function), whose value the model is trying to minimize.

Purely for illustration, a naïve but working way of learning could be to repeatedly [13]:

- Run a prediction for some input.
- Evaluate the prediction output.
- Change one of the network weights slightly.
- Run and evaluate the same prediction again.
- Keep the weight change if it improves the prediction.

This approach would be extremely slow, so neural networks use a different training method – the gradient descent. It is based on the differentiability of the loss function w.r.t. model parameters.

When a prediction is run and evaluated by a loss function, a partial derivative of the loss function w.r.t. every model weight is calculated – this is called the gradient. A gradient is a vector that can be interpreted as the direction of the steepest increase of the loss function. To minimize the loss, a step is taken in the opposite direction.

The learning step can be described as follows:

$$w_{t+1} = w_t - \eta \cdot \nabla J(w_t), \tag{2.1}$$

where w_t are the model weights at the step t, η is the learning rate, and $\nabla J(t)$ is the gradient of the loss function.

The step size is called the learning rate. To take a training step, the error must be propagated back through the network to adjust the weights accordingly – this is done using the backpropagation algorithm. Backpropagation uses the chain rule of calculus to compute the derivatives, going backward through the network computational graph.

In summary, feature activations are propagated forward during the prediction (forward pass) to produce an output, and then the error is propagated backward to determine the weight changes. The whole training process consists of iteratively improving the predictions through small changes to the weights until the desired performance is reached.

2.2 Building Blocks

Neural networks follow a modular structure that is built up from layers of operations connected into a directed graph. The following section describes the basic layers found in most network architectures.

2.2.1 Fully Connected Layer

The fully connected layer is the basic computing layer in neural networks. It consists of many neurons, which are fully pairwise connected to all neurons in the adjacent layers.

A single neuron only performs a weighted sum of the inputs (and adds a bias factor). The following Eq. (2.2) shows the operation of a single layer of neurons:

$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{x}^T \mathbf{w} + b, \tag{2.2}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is the vector of neuron's weights and b is the neuron's bias. The weights and bias values are learned using gradient descent.

Networks using fully connected layers are the successors of linear classifiers. Linear classifiers only use a single layer of neurons, though. Even though a single layer neural network can approximate any function with arbitrary precision [9, p. 192], the depth of neural networks seems to play a big role in their generalization power.

A fully connected layer is only suitable for inputs of a fixed size, which is impractical for handling images. As a result, it is not the usual core computation layer in the image processing domain.

2.2.2 Convolution Layer

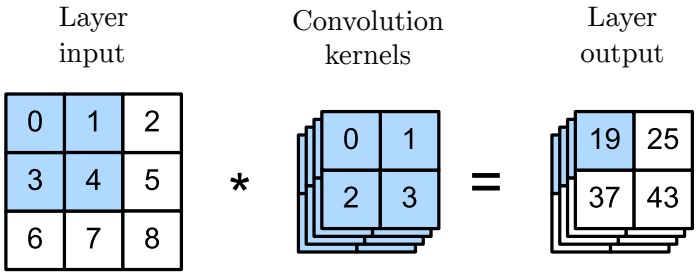


Figure 2.1: Convolution layer with 4 channels[1]**.** Highlighted cells denote values used at a single convolution kernel position.

The convolution operation is a weighted sum operation that is applied to a region of input data, acting similarly to an image filter (e.g. oriented edge detection). It is applied in a sliding window fashion, fitting the convolution kernel to every spatial position (as illustrated in Figure 2.1 above).

A single position of the convolution operation can be expressed as follows:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n), \tag{2.3}$$

where I is a two-dimensional input image and K is a $m \times n$ convolution kernel. In a single convolution layer, this operation is applied for every position (i,j), creating a $i \times j$ feature map.

In contrast to manual image filters, the values inside the convolution kernels are not hand-designed but instead are learned by the network through gradient descent. The output of a convolution layer is an activation map of the detected features. These start as low-level features (e.g. oriented edges, color gradients) and they get more complex and abstract (e.g. numbers, objects, faces) with every succeeding layer of the network. The feature detection aspect can be also thought of as aggregating evidence of local image features and outputting that as a single value. In practice, a convolution layer consists of many stacked kernels[2], as shown in Figure 2.1 below. Each kernel usually spans all the input image channels, so a given convolution layer can mix information from the whole previous layer but the output channels are computed separately to produce various features.

As opposed to the fully connected layer, the convolution layer can be applied to an input of arbitrary size. The number of parameters in a convolution layer does not change with the input size. A fixed input size is usually used during training for faster performance, though.

[1] Image adapted from: Dive into Deep Learning [34]

[2] Also referred to as the kernel width or the number of (output) channels

In usual image processing scenarios, a convolution layer contains far fewer parameters than a fully connected layer that would perform the same amount of computation. A convolution layer with the kernel size equal to the input image size would be equivalent to a fully connected layer in terms of the number of parameters. Since image features are usually not global in their nature – or they can be decomposed to several local features – the kernel sizes are commonly as small as 3×3, yet they capture enough information when stacked into many layers.

This parameter sharing also introduces translational equivariance[3] since the convolution is applied just the same at every position and so it reacts to features in any part of the input image[4] equally.

Another use for a convolution is the 1×1 convolution layer. It serves as a simple channel mixer without processing any spatial information. Such a convolution layer can also be an equivalent of a fully connected layer, where the number of convolution channels is the replacement of the number of neurons in the fully connected layer.

Since a convolution is an affine transformation, just stacking many convolutions in a row would not increase the representational strength of the model – it would still be a linear classifier[5]. [9, p. 168]

2.2.3 Activation Layers

Activation layers (nonlinearities) solve the aforementioned problem by warping the feature space to allow the neural network to learn nonlinear decision boundaries. Activation layers usually follow computation layers and they play an essential role in the neural network learning capabilities. Activation layers are usually applied element-wise.

Sigmoid

The sigmoid function is defined by the following Eq. (2.4). It has been used even before the deep learning explosion as an activation function in logistic regression. It maps an unbounded input value range to $[0, 1]$. This property has been used for learning to output probabilities in machine learning models. Since the derivative of a sigmoid is small for high positive and negative input values (saturation region), the gradients are not back-propagated well through it. For this reason, sigmoid does not get used as the main activation layer.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

ReLU

Rectified Linear Unit is defined by the following Eq. (2.5). ReLU is a common activation layer that is universally used in most modern neural networks. Even though ReLU is a piece-wise linear function, it is very efficient in increasing the model representational power.

$$\sigma(x) = \max(x, 0) \tag{2.5}$$

[3] A translation of inputs results in an equivalent translation of output features.

[4] Although convolutions can be used for any data – not just images – these cases are out of the scope of this work.

[5] An equivalent of a single layer neural network

As opposed to some other activation functions (e.g. sigmoid), ReLU does not have a saturation region.

Softmax

The softmax function is usually only used as the last layer in classification networks. It is defined by the following Eq. (2.6).

$$\sigma(x_c) = \frac{e^{x_c}}{\sum_i e^{x_i}} \tag{2.6}$$

In classification tasks, neural networks are trained to output the probabilities of each of the non-overlapping target classes. Softmax aids in this by transforming values from an unbounded input range to $w_c \in [0, 1]$ such that $\sum_i w_i = 1$. This means that the softmax output values satisfy the expected properties of probabilities.

Softmax output values follow the same idea as the use of the sigmoid activation function in logistic regression. However, neural networks are not implicitly calibrated to output accurate probabilities. For this reason, the output of a softmax layer should not be considered a reliable measure of accurate probabilities. It can still serve as a confidence score of the network, though. One practical consequence of this discrepancy can be an unreasonably confident network prediction for out-of-distribution inputs [6, p. 14].

2.2.4 Pooling Layers

Pooling layers downsample the input image, in order to reduce the volume of information passed further down the network. The downsampling can also help features with strong activations to stand out. The most commonly used pooling types are **maximum pooling** which outputs the maximum of input values and **average pooling** which outputs the input mean, as illustrated by Figure 2.2 below. Pooling layers do not have any trainable parameters.

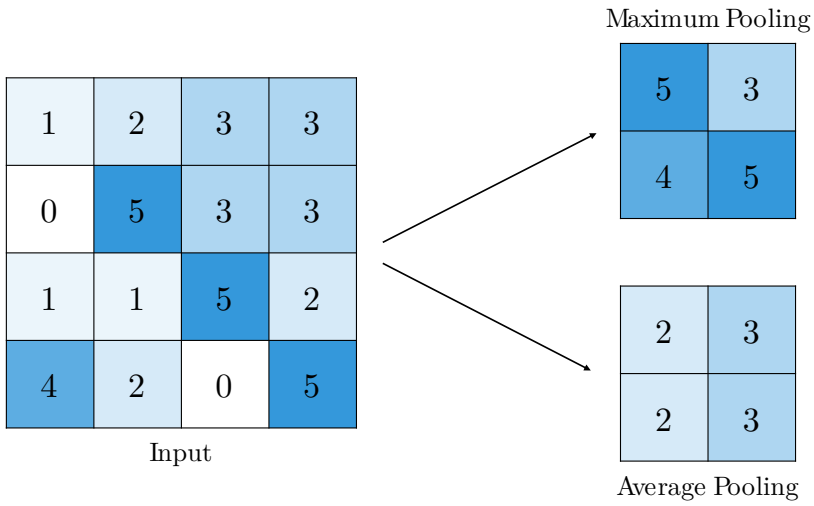


Figure 2.2: Illustration of pooling layers. Both are applied with a stride of 2.

Similarly to the convolution layer, the pooling layer can be thought of as sliding over the input image with a given kernel and stride. Since it is desirable to change the information volume as slowly as possible throughout the network layers, pooling is usually only used

for halving the input spatial dimensions. In order to compensate for the loss of spatial information, many networks double the number of channels before pooling.

A pooling layer can be also used in a global mode, reducing an input of arbitrary size to a single value. This has been used at the end of some classification networks, e.g. Inception [31] (using global average pooling) as a lightweight tail instead of fully connected layers. By replacing the fully connected layers with global average pooling, the model also becomes fully convolutional, which enables it to process variable-sized images.

In some object counting network architectures **sum pooling** is used. The network can be trained using variable-sized input patches (crops) which get collapsed to the scalar count prediction. This input size independence achieves higher generalizability of the model even when using small training datasets.

2.2.5 Batch Normalization

Most machine learning algorithms before deep learning used some type of data normalization technique (feature scaling) [10, p. 72, 333]. Shifting the data to zero mean and scaling them to unit variance has shown to be effective since many tasks work with Gaussian-distributed data.

Batch Normalization does mostly the same. For every batch, the mean and variance of every layer input are calculated and the tensor is normalized (sample-wise). Since this makes the model behavior dependent on the input data, which is not desirable at test-time, the running mean and variance of training batches are saved and used during test-time instead.

Apart from the normalization, the Batch Normalization layer also adds learnable scale and shift parameters that allow the model to learn the optimal distribution of data. The Batch Normalization formula can be seen in the Eq. (2.7) below, where $\mathbf{x}$ is the input tensor, $\hat{\mathbf{x}}$ is the output tensor, and $E[\mathbf{x}]$ and $Var[\mathbf{x}]$ are the mean and the variance of $\mathbf{x}$, respectively.

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - E[\mathbf{x}]}{\sqrt{Var[\mathbf{x}]}} \tag{2.7}$$

2.3 Influential Architecture Elements

This section introduces some of the influential CNN architecture elements.

2.3.1 Fully Convolutional Networks

In contrast to earlier CNN architectures like AlexNet [18] and VGG [30], which employed fully connected layers at the end of the network to mix the spatial information, a fully convolutional network only uses convolution layers for computation. The benefit of this approach is that the network can run inference on images of arbitrary resolution.

Since a 1×1 convolution can be equivalent to a fully connected layer, the fully connected layers at the end of VGG-like classification CNNs can be replaced by them while keeping the functionality identical [10, p. 473]. This has been used in earlier image segmentation architectures [23]. A single network prediction run on a larger image performs the same as multiple predictions run separately on parts of the image, but the single prediction is much more efficient.

Fully Convolutional Networks are usually used for image-to-image scenarios, where the output size is equal or similar to the input image size. In UNet-like architectures, which are commonly used for the task of object detection or image segmentation, the output is a heatmap of class activations which are usually trained to approximate Gaussian kernels.

2.3.2 Residual Networks

Residual connections first appeared as a solution to the vanishing gradient problem and the inability to train deep convolutional networks because of it [12]. Such connection adds the output of a convolution branch to the identity branch output, as illustrated by Figure 2.3 below.

During the forward pass, its benefit is the ability to retain original information deeper throughout the network. During the backpropagation, the gradient is carried through the identity branch. Residual blocks have also been used as a way to achieve multi-scale context aggregation [25].

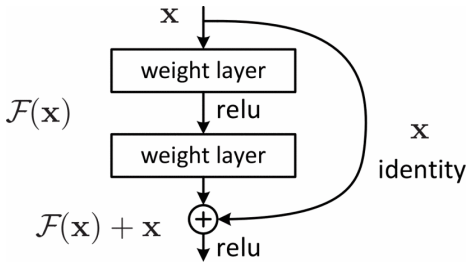


Figure 2.3: A residual connection illustration. $\mathbf{x}$ is the layer input, $\mathcal{F}(\mathbf{x})$ is the output of a computation (convolution, fully connected) layer[6].

2.3.3 Dilated Convolution

Dilated convolution (also called atrous convolution) is a variation of the classical convolution layer which uses a "sparse" kernel with empty space (zeroes) in between the kernel values, as shown in Figure 2.4 below. Dilation rate is the distance between neighboring non-zero kernel values.

This allows for a faster receptive field growth. As opposed to the pooling layers, the downscale caused by these layers is learned. This should allow the network to select the relevant information to retain.

Dilated convolutions have been used in crowd-counting networks like CSRNet [21] and in semantic segmentation networks (by Yu and Koltun [33]).

The second listed work proposes an exponential growth scheme for consecutive convolution layers. Setting the dilation rate to successive powers of two leads to the maximum receptive field growth that does not cause a loss of resolution. All of the convolutions use 3×3 kernels. There are still 3 pooling layers with *stride* $= 2$ but most downsampling is done through the dilated convolutions.

[6] Image source: Deep Residual Learning for Image Recognition [12]

[7] Image source: https://docs.nvidia.com/deeplearning/performance/dl-performance-convolutional/index.html

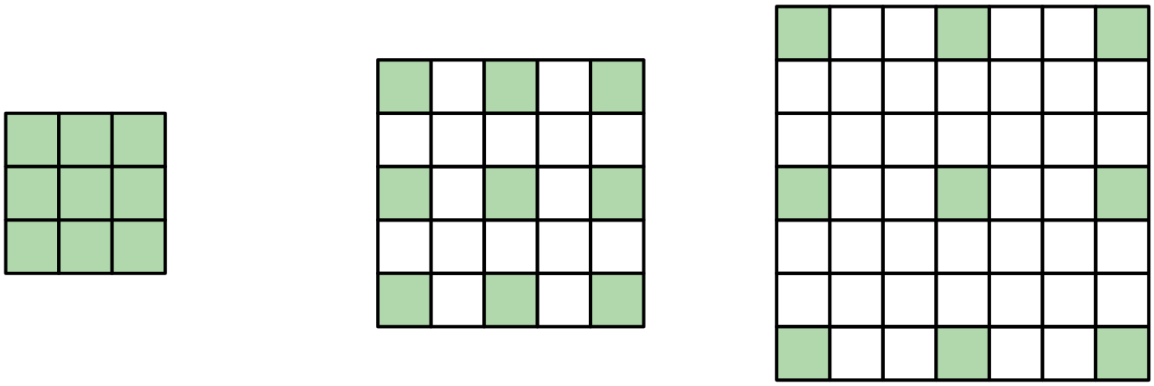


Figure 2.4: 3×3 convolution kernels with a dilation rate of 1, 2, 3 (left to right). Trainable kernel values are in green, the rest are zeroes.[7]

2.4 Addressing Class Imbalance in Data

Many CNNs perform tasks where the data is inherently imbalanced, e.g. classification must recognize rare classes, or object detection must process images that are mostly background. This may lead to poor model performance, as the prediction loss is dominated by the common classes and the model gets little incentive to learn the rare classes well.

This problem can be approached in multiple ways. The first approach is to weigh the classes based on their number of samples. When the prediction loss is calculated, the loss is multiplied by the class weight of the ground-truth class.

- **Uniform** – All class weights are the same, equal to 1.
- **Inverse Frequency** – Class weights are inversely proportional to the number of samples. The class weight Eq. (2.8) is shown below, where c is the class for which the weight is computed, s_i is the number of samples of class i and n is the total number of classes.

$$w_c = \frac{\sum_{i=1}^{n} s_i}{n \cdot s_c} \tag{2.8}$$

  Inverse Frequency weighs examples according to the likelihood ratio, minimizing the total loss given the ratios of class samples.

- **Effective Number of Samples** – The class weighing approach was introduced in [3] follows the intuition that the importance of samples from a given class decreases with every additional sample.

$$w_c = \frac{1 - \beta_c^s}{1 - \beta} \tag{2.9}$$

  β is referred to as a hyperparameter in the original paper but only a single for β is used: $\beta_c = \frac{s_c - 1}{s_c}$, where s_c is again the number of samples of class c. Overall, β is expected to lie in $[0, 1]$, with $\beta = 0$ making the weights uniform and $\beta = 1$ being equivalent to the inverse frequency weights. The values in-between allow for manual balancing of the resulting weight distribution.

Interestingly, there are also findings [2] claiming that deep neural networks tend to learn to ignore the class weights.

Another way of combating the class imbalance is **oversampling** of the minority classes. If the dataset is resampled so that all classes are equally distributed, the loss computation is equivalent to using Inverse Frequency class weights. In this case, the advantage of oversampling over the class weights is that the per-batch gradients are smoother. This is because a rare sample is seen many times, compared to seeing a sample rarely but with a high importance.

Extending on the concept of oversampling, there are also ways of generating new samples. One of them is the Synthetic Minority Over-sampling Technique (SMOTE) [1]. Samples that lie close in the feature space are interpolated to enrich the dataset.

A similar concept related to the class distribution balancing is **Online Hard Example Mining** (OHEM) [29]. The idea is that some samples are easy to classify, the model learns to classify them correctly within the first few epochs and after that, they do not add any value to the training. In object detection tasks, there are many possible object locations, most of which are easy to evaluate. Computing the loss for all of them and applying the gradient computed from only the hardest ones makes the training more effective.

2.5 Applications of Convolutional Neural Networks

This chapter presents common tasks for which Convolutional Neural Networks are used.

In the field of computer vision, the following recognition tasks received great attention and became big drivers in the development of CNNs. Classification is the task of assigning a class label to a whole image. Usually, only one label is assigned and it describes the most prominent object in the image. Object localization usually refers to finding an object's bounding box. Object detection both localizes the objects and determines their categories. It can be noted that the tasks have been listed roughly in the order of their dominance in the field.

The tasks of image (semantic) segmentation and instance segmentation lie close to the object detection domain. Semantic segmentation produces contiguous areas of object classes in the image. Instance segmentation does the same and also distinguishes between individual object instances.

2.5.1 Object Detection Approaches

Since object detection is a universally applied task, there exist many approaches. Most major object detection architectures strive for generality and the typical output is a set of bounding boxes wrapped around instances of class objects.

Although not very popular nowadays, there have been many solutions to this task that used a **classification-based approach**. Classification of a sliding window is performed, creating activation maps for every class. This approach is simple in its nature, but it makes the assumption that the object can be detected solely by its local features – disregarding the global context. The advantage of this approach is that it can work well with small datasets.

There have been successful uses of classification-based detection in, e.g. road detection [24] and corn kernel counting [16]. This approach has also been used in combination with weakly supervised learning to obtain object positions without the necessity of accurate labels [26].

On a side note, it can be argued that the per-pixel classification output is neither object detection nor image segmentation.

Since the last convolution layer in a classifier network contains both spatial and categorical information, its activations can also be used as a cue for localization of objects. These class activation maps either directly represent or can be mapped to the predicted classes [35].

Modern CNN architectures perform the detection directly, unlike the classification-based approach. There are two categories of object detectors – two-stage detectors and one-stage detectors.

Two-stage detectors have evolved from the sliding window approach [14]. Classification of every possible object position has been replaced with a Region Proposal Network that suggests likely object locations. The second stage is a classifier that either detects some object category or dismisses the region as background. The defining architecture of this category was the R-CNN [8]. There have been many follow-up architectures like the Fast R-CNN [7], the Faster R-CNN ,[28] and the Mask R-CNN [11]. These architectures further improved the speed and performance gradually turning the multiple subsystems into an end-to-end solution.

Two-stage detectors usually contain a classifier network as the feature detector/region classifier part of the model.

One stage detectors perform the detection end-to-end, within a single network. The most famous object detector overall is the YOLO architecture. YOLO poses the prediction of bounding boxes as a regression task. The input image is divided into a square grid for the object classification. Another grid is used for bounding box suggestions. The bounding box predictions contain box width, height, and scaling factors in the two spatial dimensions. Base bounding box sizes are used (anchor boxes) to ease the learning of the correct bounding box absolute sizes.

There are also point-based object detectors, such as CornerNet [19], CenterNet [36, 4][8]. They fit into the one-stage detector category and their output usually contains a heatmap of Gaussian kernels. Apart from general object detection, these architectures are also used for human pose estimation [25, 17].

Most object detection approaches face the problem of balancing between the false positive rate and the false-negative rate (See Subsection 4.1 for more about these metrics). This is largely a consequence of the high background-to-foreground ratio. Many techniques have been developed to combat this imbalance. Focal loss [22] lowers the loss for well-classified examples so as to put more focus on the hard samples. Since many background regions are easy to classify as background, these end up making up a lower part of the total loss.

[8]There are two architectures called CenterNet, unrelated to each other but using a similar approach.

Chapter 3

Design and Implementation

I have decided to design and implement a crate-counting solution for photos of blocks of matchboxes. I have chosen to use a CNN for detecting the matchbox keypoints in images and then a non-DL algorithm for interpreting the predictions to obtain the final crate count. The keypoints are corner points of the individual matchboxes built up into a cuboid block. The detection uses a classification-based CNN to detect the keypoints, as shown in Figure 3.1. Since the keypoints are located at the intersections of matchbox edges, they can be detected solely based on their local features and no global context is needed. The geometric nature of keypoints can also be considered as mostly scale-invariant, allowing the network to perform well at multiple scales without explicitly designing it to do so.

The CNN predictions are then converted to keypoints which are then processed by the crate-counting algorithm, outputting the final crate count.

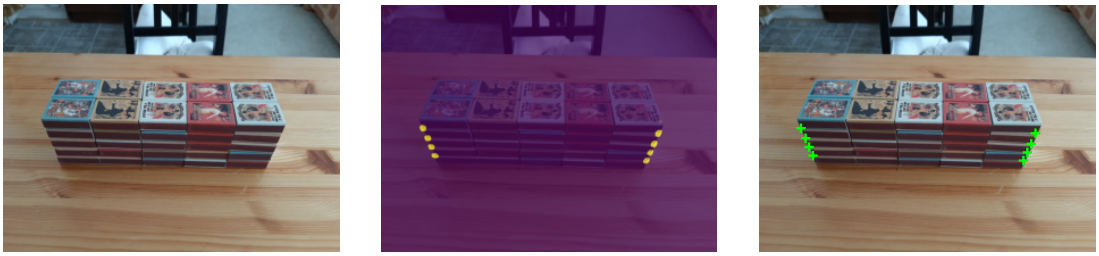


Figure 3.1: Example input and output of the keypoint detection network. CNN activations form blobs around keypoints of a given category (edge-side), keypoint estimation algorithm converts the prediction blobs to keypoints.

3.1 Dataset

The dataset consists of square cutouts (patches, samples) from 52 photos of matchboxes[1]. The matchbox photos' dimensions are 4032×3024 pixels. These photos were annotated

[1] A note on the deviation from the original assignment: The described dataset has been created as a toy dataset for initial experiments but ended up being the only dataset used, due to unforeseen circumstances. I originally planned to work with data from a beer brewery in Pilsen. There were supposed to be warehouse photos of blocks of beer crates available. I never got access to this data in the end. With the approval of my supervisor, I decided to focus on the functional part of the solution, even though the used dataset itself is tiny.

using the MakeSense.ai annotation tool[2]. The annotations mark the locations of 7 keypoint categories, covering different types of matchbox corners (vertices), according to their position in the block (cuboid) and they are shown in Figure 3.2 below. In addition, the total number of matchboxes in each photo is annotated on the side.

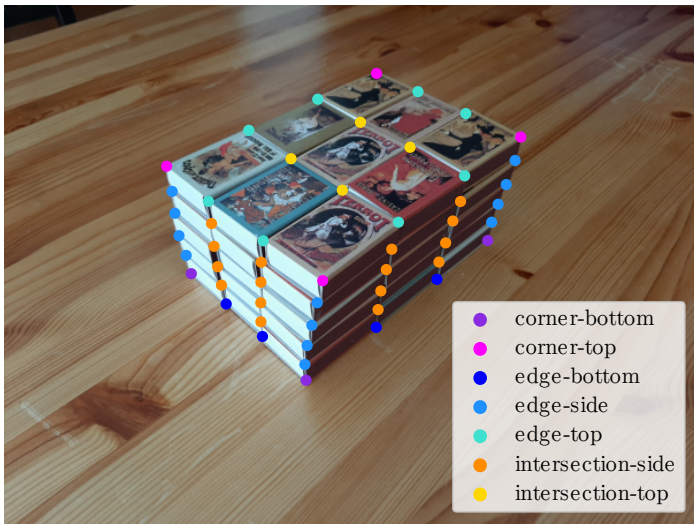


Figure 3.2: Example dataset source photo with annotated keypoint categories.

There are 7 categories of keypoints with over 1500 annotated keypoints in total. The dataset contains fixed-size cutouts created around the keypoints and cutouts randomly sampled from the background. Together with the background, there are 8 cutout classes in total. To clarify the use of the term background in this text, the background category represents any non-keypoint part of the image, so even the matchbox sides and edges are considered as the background category. A single sample represents the category of its center-point.

For generating a single cutouts dataset instance, the following parameters are used.

- **Cutout Dimension** – A cutout dimension from 32 to 128 pixels has been used.
- **Scale** – A scaling factor of 0.25 to 1.0 has been used. The same factor is also used when working with full images.
- **Background Sampling Density** – The number of background samples per photo ranges from 200 to 1000.
- **Minimum Background-to-Keypoint Distance** – A minimum distance of 16 pixels has been set.

The choice of the dataset parameters is a trade-off between performance and the quality of predictions. Over the course of development, experience has shown that models using larger cutouts perform better. On the other hand, this can be attributed to not only the larger available context but also the fact that these models contain more layers and parameters.

Since the original image resolution is way bigger than what usual object detection CNNs use, multiple downscaling values were used. A scale of 0.25 has been used in most cases, as it is the highest scale that did not exceed hardware limitations for practical use.

[2]Available at: https://www.makesense.ai/

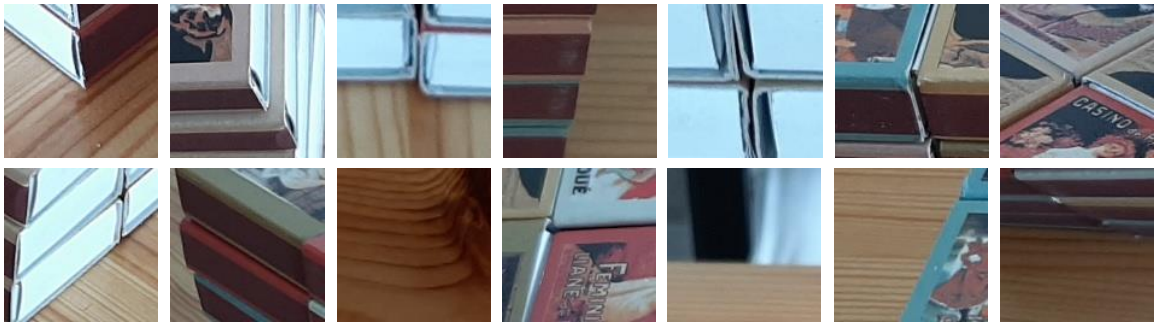


Figure 3.3: Example cutouts from the dataset. The top row contains all of the keypoint categories listed above, the bottom row shows various background samples.

The background sampling density turned out to be essential for the quality of full image predictions. Higher values make the dataset heavily imbalanced, but the full image predictions are way more robust when the cutouts cover the whole image densely.

For the rest of this text, the term dataset denotes the final combination of dataset parameters: 128×128 samples at a scale of 0.25, with 1000 background samples per image and a minimum of 16 pixels between a keypoint and a background position. The class imbalance created by the heavy background sampling is shown in Figure 3.4 below.

Figure 3.4: Number of samples per class in the used dataset version.

These samples have been split into training and validation subsets. Training data come from 46 photos and the validation data from the remaining 8 photos. A single photo contains either only training or validation samples, never both. The number of samples in the split is 46372 for the training and 8286 for the validation. There is no test subset of the dataset, as the full image predictions serve the

3.2 Keypoint Detection Network Design

A per-pixel classification approach has been chosen for the keypoint detection. As opposed to the regression of bounding boxes/keypoints, this approach is very simple, lightweight, and it requires less data for training than classical object detection. In its basic form, the classification-based approach can be thought of as yet another sliding window that extracts image features from given regions and outputs them as target class probabilities. The output tensors are dense activation maps of the target classes.

The architecture is a simple VGG-like, sequential classification CNN. The main building block of the architecture is a sequence of a convolution layer, a ReLU activation layer, and a batch normalization layer.

3.2.1 Patch-based Training

The model is trained using image patches of a fixed size. For training, the shape of a single sample is $D \times D \times 3$, where D is the cutout dimension and 3 is the number of channels (RGB). The output shape is $1 \times 1 \times C$, where C is the number of target classes. The goal is to train the model on small patches and then run inference on full-sized images – in a fully convolutional mode.

While classification is a widespread task and there exist many models, none of the classical architectures are suitable for this specific problem. All commonly used classification backbone models use strided operations for downsampling. For an arbitrary input of dimension $W \times H \times 3$, the shape of their output is close[3] to $\frac{W}{S} \times \frac{H}{S} \times C$, where S is the product of all strides used in the model layers.

For example, if a network has been designed for training on 64×64 patches and it runs inference on a 1000×750 input image, it will only output 15×11 activation maps. For a finer resolution output, a prediction could be run separately on every 64×64 cutout from the full image to produce a classification for every pixel, but that is not feasible performance-wise.

To overcome this problem, the model must not use any operations with a stride higher than 1. If the only reduction of dimensions in a model is cropping the input (subtraction instead of division), then the model output shape is only smaller than the input by a constant margin, not by a scaling factor. A summary of this concept, which is essential to my work, can be viewed in the Table 3.1 below.

Table 3.1: The effect of downsampling through $stride > 1$ operations compared to $stride = 1$ downsampling. Square input/output shape is considered. The number of channels is neglected here. D is the training patch dimension. K is an arbitrary scaling factor > 1.

	Input Dimension		Output Dimension
Downsampling through stride	D	$\longrightarrow$	1
	$K \cdot D$	$\longrightarrow$	K
No stride downsampling	D	$\longrightarrow$	1
	$K \cdot D$	$\longrightarrow$	$(K-1) \cdot D + 1$

[3] there are minor dimension changes caused by $stride = 1$ operations, too, e.g. convolutions with $kernel\_size > 1$.

If the model cannot use strided operations for downsampling, the downsampling must be caused by the kernel size of convolution and/or pooling layers. A convolution kernel of a dimension K subtracts $K-1$ from the input dimension. A quick way to downsample the image would be to use a few layers of large convolution kernels – for example, 7 layers of 10×10 convolutions. The shortcoming of this is that a larger convolution kernel results in an unnecessarily high computation volume (and number of parameters) compared to several smaller kernel convolution layers [15, 30][4].

On the other hand, using only 3×3 convolution layers would make the network unnecessarily deep. Even comparable modern classification networks like EfficientNet [32] use a baseline depth of 18 layers. Since the simple geometric structure of keypoints is much less abstract than usual classification target classes, clearly fewer layers should be enough.

3.2.2 Dilation Rate Growth

At the first layer of a network, a single value of a convolution kernel is only affected by a single pixel in the input image. As the convolution layers aggregate neighboring pixel values and max-pooling layers copy strong intermediate feature activations, there is an overlap in the receptive field of the individual kernel values. This overlap can be used for additional processing of the features or avoided to increase the receptive field faster.

The proposed model architecture uses dilated convolutions to downsample the image faster, in order to capture a higher receptive field within just a few layers. The dilation rate of a given layer should reflect the receptive field and the resolution of neighboring values in the previous layer output.

Searching for the theoretically optimal dilation rate growth scheme did not bring any results, so multiple candidates were evaluated and the best-performing one was chosen. All of the mentioned schemes use a base convolution kernel size of 3×3, followed by an activation function and a MaxPooling layer with a 2×2 kernel and a $stride = 1$.

Out of the many considered dilation rate growth schemes, the following 3 have been shortlisted. The descriptions contain the growth of the dilation rate between successive convolution layers.

- Increment the dilation rate by $1 \Rightarrow 1, 2, 3, 4, \ldots$
- Increase the dilation rate by $2 \Rightarrow 1, 3, 5, 7, \ldots$
- Multiply the dilation rate by a factor of 2, as proposed by Yu and Koltun [33] $\Rightarrow 1, 2, 4, 8, \ldots$

None of the schemes fit well to the model input size of 2^N, which was used because of the hardware efficiency when working with powers of 2. For this reason, the network design contains some irregularities in the convolution kernel size or the dilation rate.

During development, multiple models 32x models were used but experience has shown that models using larger input sizes perform better. For the 64x input size, the odd dilation growth rate performed the best. Soon after, other dilation rate schemes were abandoned and did not get tested any further.

[4] Computing two 3×3 convolution layers take $2 \cdot 3^2 C^2 WH$ operations, while a single 5×5 convolution layer takes $5^2 C^2 WH$ operations. The numbers of layer parameters share the same ratio. The receptive field is the same for both cases. Smaller kernels take fewer parameters, they are cheaper, and they make for deeper networks.

For the final network design, the odd dilation rate growth scheme has been used. The list of the model layers can be seen in Table 3.2 below.

Table 3.2: Overview of the 64x model with an odd dilation rate growth. C is the base number of channels.

Kernel size	Dilation rate	Output dimension	Output channels
3×3	1	62	$2C$
3×3	3	56	$4C$
3×3	5	46	$4C$
3×3	7	32	$8C$
3×3	9	14	$8C$
2×2	11	3	$16C$
3×3	1	1	$16C$
1×1	1	1	$16C$
1×1	1	1	8

The number of channels grows as the spatial dimensions decrease. Multiple values of the base number of channels were used to determine the optimal model capacity. In the end, values of $C \in \{16, 32\}$ were used.

Residual networks were experimented with during much of the development but did not get used in the final design. The simpler sequential models turned out to be good enough for the classification task and so the need for a more complicated model fell.

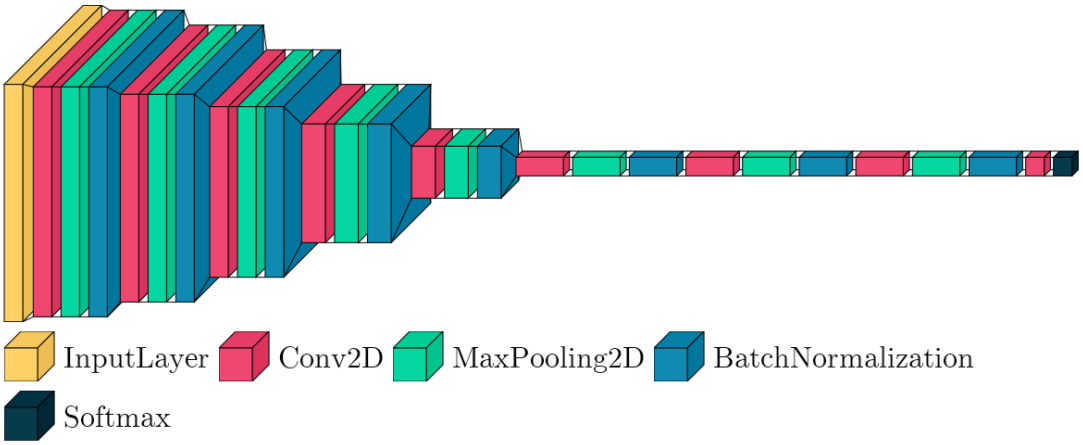


Figure 3.5: Model architecture visualization. Spatial dimensions range from 64×64 to 1×1. Number of channels ranges from 32 to 512.

3.3 Keypoint Detection Network Training

This section describes the training of the proposed model architecture.

3.3.1 Solving Class Imbalance

Since the background is densely sampled to capture the non-keypoint details thoroughly, the dataset is heavily imbalanced – for the final background sampling density of 1000 samples per image, the foreground to background ratio is less than $1 : 35$.

For the majority of the development, class weights have been used for training. Only upon closer evaluation, it became apparent that using the class weights had been decreasing the model performance significantly. The inverse frequency class weights have been dropped for uniform weights, which tremendously improved the model performance.

3.3.2 Training Details

The Adam optimizer has been used, as it is the standard choice in modern CNNs. The choice of an optimal learning rate affects not only the training time but also the performance of the model. To achieve optimal model performance, the learning rate is often varied throughout the training run – decreasing it allows the model to find smaller loss minima, and increasing it helps the model to surpass plateaus.

Multiple learning rates have been tested. A base learning rate value of 10^{-3} turned out to be among the fastest and consistent enough.

During a large part of the development phase, a learning rate scheduling with an exponential decrease was used. This turned out to be suboptimal, as it did not reflect the current model performance and the learning rate either dropped too soon or stayed high for unnecessarily long. Reducing the learning rate on plateaus has proved itself effective. When the model performance plateaus and does not improve on the training PR-value by more than 10^{-2} during 5 epochs, the learning rate gets halved.

To aid in avoiding overfitting, an early stopping rule has been set. If the validation PR-value performance does not improve by at least 10^{-2} during 15 epochs, the training is ended.

All models have been trained for 80 epochs unless cut short by the early stopping.

The training batch size can be set to a mostly arbitrary value, ranging e.g. from 32 to 1024, with higher typically being better. The upper bound on a batch size is usually hardware-imposed. A higher batch size leads to less varied distributions of the sample classes, which should have a positive impact on the training. On the other hand, an experiment with a batch size of 16384 showed that such an extreme value is actually detrimental to the training.

3.3.3 Data Augmentation

Data augmentation is a technique used to enhance datasets by introducing small perturbations to the input. These modifications result in data that is visually similar to the original and the model should learn to ignore these small variations. In many machine learning scenarios, data augmentation is the cheapest way of producing more training data, and subsequently, achieve higher levels of generalization.

Training on a small dataset with a limited number of keypoint samples calls for heavy use of augmentation. Additionally, the simple geometric nature of the keypoints allows for high intensity of augmentation. The list below explains the decision process behind the choice of augmentation types and their intensities. The values noted below were used in the All of them were applied with a random intensity bounded by 0 and the specified maximum value.

Since some augmentations cut off part of the training samples, the chosen set of maximum augmentation ranges has been set so that this cutoff does not introduce artifacts like black corners in a rotated sample. The generated sample dimension is usually twice the model input dimension (128× samples for a 64× model) and it gets cropped to the model input size after the augmentation process.

- **Translation** – Since the sample classification is interpreted as the classification of its center, a spatial shift of the sample may seem counter-intuitive. The convolution and pooling layers should also by design make the model robust towards translation.

  on the other hand, since the background samples are sampled from at least K pixels away from any keypoint, there is a circular gap around the keypoints where there are no training data. This gap can be closed by training the model to predict keypoint categories even in the close vicinity around the keypoint. Even if this makes the prediction less accurate from the distance-to-keypoint perspective, it is better than leaving these samples out of the dataset distribution and thus letting the model to make an uncontrolled prediction in these areas.

  Multiple translation ranges were evaluated experimentally and the maximum value of $\frac{1}{32}$ of the training sample size resulted in the best overall performance.

- **Rotation** – Samples are rotated around their center by up to 22.5° in either direction. Beyond this value, a sample can become hard to classify even for a human.

- **Zoom** – A maximum zoom range of $\pm\,50\,\%$ has been chosen, as it represents a reasonable variation of camera distance. Higher levels of zoom (especially zooming out) are impractical when using a dataset consisting of pre-generated samples, as the samples would need to be much bigger than the actual used size, in order to produce training samples without any artifacts.

- **Flipping** – Horizontal flipping gets applied with a $50\,\%$ probability. Vertical flipping does not get applied, as these samples fall out of the expected input distribution.

- **Color transformations** – Since the color properties of a dataset may often be too similar and the network should be able to generalize beyond the specifics of the dataset, many color transformations can be used to help with that.

  A **hue** shift of up to 0.2 has been chosen, along with a **saturation** scale factor of $[0.1, 1.5]$[5]. The color augmentation ranges are rather high because color information does not play a big role in this task.

  A **brightness** delta of up to 0.3 (symmetric) has been chosen. The **contrast** factor lies in $[0.35,\ 2.0]$[6].

3.4 Keypoint Estimation

The following processing converts the predicted activation regions to keypoints. It can be summarized by Figure 3.7 below.

[5] A scale factor of 1 represents the original saturation.
[6] A scale factor of 1 represents the original contrast.

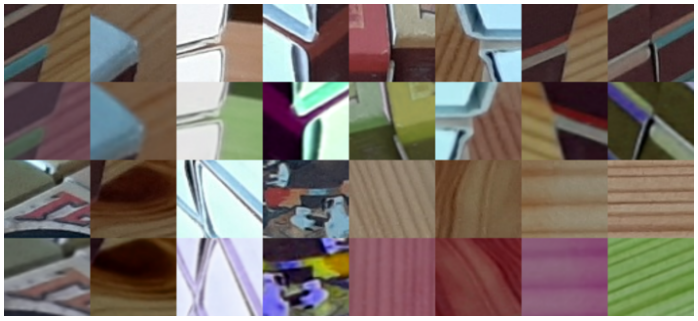


Figure 3.6: Example augmented samples. Odd rows contain original samples, even rows contain their augmented versions.

Figure 3.7: Estimating a keypoint from predicted activation regions. The keypoint position is the centroid of the whole blob area produced by the (keypoint-class-independent) connected component analysis. The majority blob class is used as the keypoint class.

First, thresholding is performed on the prediction with the threshold value of 0.9. This filters out small noise in the predictions but it does not change the predictions drastically, given the hard decision boundaries often learned by the network.

The thresholded prediction is channels-wise merged to a binary matrix, denoting any keypoint category activation. On this matrix, a connected component analysis is performed to find contiguous prediction regions. A 2-hops-connectivity (also called 8-connectivity) is used. The outputs are blobs of various sizes, independent of the prediction classes they originated from.

Since there may be some noise in the keypoint categories classification, a blob often contains a major chunk of the main predicted class and a minority of pixels from a different class. The blob category is set to the most prominent class in the original prediction.

To mitigate noisy keypoint predictions, blobs smaller than a certain number of pixels are discarded. This value has been set to $160S^2$, where S is the scaling factor (introduced in Section 3.1). For large enough blobs, the centroid of the blob area is used as the final keypoint location.

Basic outlier detection is also performed on the keypoints, so that further false-positive predictions are removed. It uses a simple heuristic based on the cuboid shape of the block. The mean distance from every keypoint to all other keypoints is calculated. If a keypoint has the mean distance of 1.8 times the average, it is removed.

While accurately estimating the keypoint positions is useful for further processing, the main decision factor is the correctness of the keypoint count and the keypoints' classification.

3.5 Crate-counting System

I have decided to build a crate-counting system based on geometric interpreting of blocks of crates.

After consulting the topic with my supervisor, I have decided to restrict the counting system to a single block of crates (matchboxes) with a regular cuboid shape (e.g. $5 \times 3 \times 3$). The dataset used does not break this assumption.

The counting algorithm first performs the view recognition based on the number of predicted keypoints of the corner-top and corner-bottom categories. The considered views of the blocks of crates are shown in Figure 3.8 below.

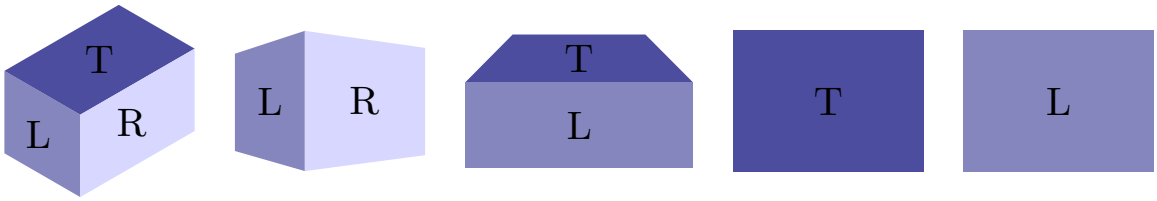


Figure 3.8: Different views of the blocks of crates. The geometric interpretation of keypoints recognizes these 5 views. T, L, and R represent top, left, and right respectively.

Since the keypoint categories do not capture all positional details of the keypoints position (front-back position, left-right position), the geometric interpretation does not make any assumptions about the block orientation or dimensions. The main problem that the geometric interpretation has to solve is to understand the structure of the block.

The following descriptions use the y-axis to denote the vertical direction. The horizontal directions denoted by x and z axes may get switched up depending on the orientation of the block, but it does not make a difference for the counting result.

Table 3.3: Determining the view from visible corner points. View abbreviations denote the visible sides, as shown in Figure 3.8.

Top corners	Bottom corners	View
4	3	TLR
4	2	TL
4	0	T
3	3	LR
2	2	L

When the view is not recognized or the computation fails at some point, the value -1 is returned.

Three-side View (TLR)

The following description of the geometric interpretation of the keypoints is also illustrated in Figure 3.9 below.

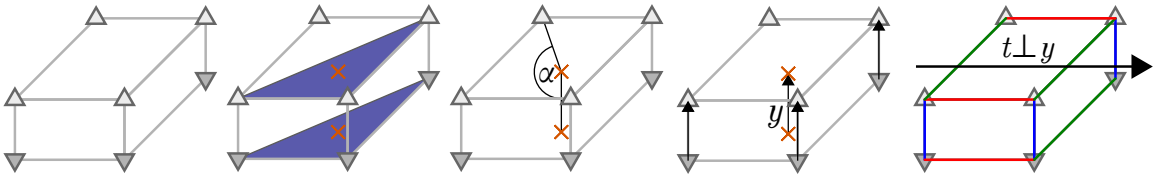
In the TLR-view, there are 4 top corner points and 3 bottom corner points visible. First, the back top corner point (one that does not share an edge with a bottom corner point) is determined. The decision is based on how well the front top corners separate the front bottom corners from the back top corner. When the points are separated well, the angle drawn in (3) will be close to 180°. The back top corner point is the one that causes this angle to be the closest to 180°. For the front (top and bottom) corner points, the centroids

of the triangles they form (2) are considered for measuring the angle. The step (3) has been introduced, since simply matching the top and bottom corners like in (4) showed failure cases due to perspective distortion.

Once the back top corner is determined, the front bottom corners are matched to the front top corners by shifting them by the y-axis vector (4).

The horizontal (x-axis and z-axis) ordering of the points is determined using a line t perpendicular to the y-axis vector (5). Midpoints of the vertical lines are orthogonally projected onto t and a left-front-right ordering of the vertical lines is found. At last, the corners are connected to form all edges of the block of crates.

To determine the final crate count, the number of points along each edge is counted. A point is said to lie on the edge if its distance from the edge line is at most 10 pixels.



(1) (2) (3) (4) (5)

Figure 3.9: Geometric interpretation of the keypoints – three-side view.

Two-side Views (TL, LR)

For the TL-view, the 2 front bottom corners are used as the x-axis vector. Top corner points are connected to create lines parallel to the x-axis, creating the remaining x-axis edges. A vertical ordering is found for the x-axis lines (front-bottom, front-top, back-top), based on which the remaining y-axis and z-axis edges are created. Along all the edge lines, the corresponding edge point categories are detected and counted.

For the LR-view, the procedure is based on the three-side view (TLR). The 3 bottom corners are matched to their top counterparts, they are ordered along the horizontal dimension and connected. Corresponding edge point categories are counted along all the edges.

One-side Views (L, T)

Since these views of the block do not contain complete information about the block structure, they only count the number of visible crates in the 2D grid.

For the L-view, the horizontal lines are the only possible lines. To ensure the correct order of corner points, both horizontal line vectors are made to point to the positive x direction. Corners are connected bottom to top based on their horizontal order and the points of corresponding edge categories are counted along all the edges.

For the T-view, a convex hull wrapper is found. Then, pairs of parallel lines are chosen, using an arbitrary order of x and z axes. Points of the edge-top category are counted along every edge.

The implemented counting approach does not make use of the intersection point categories. Many improvements and alternative approaches were considered but did not get

implemented in time – e.g. using the Hough transform to detect lines of keypoints and fallback scenarios for interpreting incomplete predictions.

3.6 Implementation Environment and Tools

Most of the development has been done on a Ubuntu 20.04 LTS laptop with a Nvidia 1050M 4GB GPU. For running numerous experiments, the Metacentrum computational grid[7] has been used.

The implementation was written in the Python programming language (v3.8). For the CNN development, the Tensorflow framework was used (v2.4). Image manipulation was done using OpenCV and NumPy. Further operations use the following libraries – SciPy, Scikit-Learn, and Matplotlib. For logging, visualization, profiling, and a general overview of CNN training, TensorBoard has been used.

3.7 Performance Overview

There are two performance[8] aspects to consider – the training requirements and the inference requirements. The main limiting hardware factors are the GPU memory and the computation time.

All the models can be trained on an average laptop GPU (Nvidia GTX 1050M 4 GB). Classification of small samples is fast and resource-light. The training of the largest tested model parameters-wise takes around 2 hours on a modern desktop GPU (Nvidia GTX 2080 8 GB).

For full image predictions, GPU memory is the main constraint and the noted laptop GPU was not enough to run full image predictions at a 0.5 scale (input dimensions 1500×2000). For this reason, earlier development evaluation runs often used only a center-cropped region of the image. At a 0.25 scale, the predictions can be run, taking ≈ 1.3 seconds per image. Because of the memory limit, a higher model input size (e.g. 128x) was not used, as making the full image predictions would not be possible on the development hardware.

To enable the fully convolutional mode of patch-based training and full image predictions, the model is defined without specific input dimensions. The Tensorflow framework allows this and creates the computational graph for given input dimensions on the first prediction (training step or inference). A (re-)tracing of the model operations is performed again whenever the input dimensions change. Retracing is an expensive operation, as illustrated by Table 3.4 below.

Table 3.4: The cost of model retracing. Timings were measured on the development laptop, using the final model architecture with ≈ 1 million parameters.

Input dimensions	First prediction [s]	Further predictions [s]
600×600	6.97	0.68
756×1008	19.94	1.31

[7] https://www.metacentrum.cz

[8] This is the only section to use the term performance to describe how fast the solution works, instead of how well.

All existing computation graphs are cached by the framework, so the tracing is only performed twice for a typical training run – once for the training sample size and once for the full image size.

Additionally, to optimize the training performance, a 16-bit precision mode has been used for training. The last layer (Softmax) is the only one using a 32-bit precision, so that small output confidence scores do not underflow to zero.

Chapter 4

Experiments

To find the best model version, multiple variants of the proposed design were evaluated. This chapter verifies some of the design choices with data and explores the effect of further design choices – e.g. the chosen metrics or the model input size.

Validation data were only evaluated every 5 epochs, since the difference between successive/consecutive epochs is insignificant. The full image metrics have also been evaluated every 5 epochs since running the full predictions for the whole training and validation dataset is rather long (≈3 minutes) compared to the epoch length (15 seconds to 1 minute).

4.1 Model Performance Evaluation

Multiple metrics have been chosen for the model prediction evaluation. Since the counting task is not solved directly but is instead built up from several stages of intermediate tasks, multiple aspects of the model performance must align for the counting to work well. The metrics monitor the model's performance at multiple levels. First, the sample level metrics evaluate the cutout classification predictions. Then the keypoint-level metrics evaluate the detection of keypoints in full images. Finally, the crate-counting metrics judge how well the solution works as a whole.

For all of the metrics, only the winning category of the model predictions are considered (hard classification), while ignoring the prediction confidence value and any other non-zero classes. Empirically, working with the confidence values did not show any benefit as all models tended to learn sharp decision boundaries.

4.1.1 Sample Classification Performance

There are multiple ways a classification prediction can be evaluated. A single metric can rarely express enough information to evaluate a model wholly.

For the precision and recall metrics (introduced below), the evaluation of the prediction is simplified to a binary classification problem. All keypoint categories are treated as positive cases and the background is considered a negative case. This is because a misclassification within keypoint categories is not as costly as a keypoint-to-background error (or vice versa). For binary predictions, a few commonly used terms are noted in Table 4.1 below.

- **Accuracy** – This metric expresses how often the predictions match the labels.

Table 4.1: Overview of binary classification evaluation terminology.

		True Label: Positive	True Label: Negative
Predicted Label	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Because of the strong class imbalance of the dataset, accuracy does not accurately reflect the quality of model predictions. Very high accuracy (97 %) can be achieved simply by always predicting the background class, but this is clearly not a useful prediction. Accuracy serves as a good baseline for the early stages of development, but it does not suffice by itself.

Since the biggest flaw of accuracy is the overly high contribution of the true negatives, metrics independent of the number of true negative cases have been chosen.

- **Precision** – Precision answers the following question:

  *How many selected items are **relevant**?*

  As can be seen in Eq. (4.2) below, precision is inversely correlated with the false positive rate.

  $$P = \frac{TP}{TP + FP} \tag{4.2}$$

  This is a key metric to evaluate how well the background is sampled in the dataset and how the model is expected to generalize from sample-based training to full image prediction. If the background patches are sampled densely enough, the model should confidently classify everything but the keypoints as background.

- **Recall** – Recall answers the following question:

  *How many relevant items are **selected**?*

  Recall (also called sensitivity) is inversely correlated with the false-negative rate (see Eq. (4.3) below).

  $$R = \frac{TP}{TP + FN} \tag{4.3}$$

  This metric expresses how well can the model detect the keypoint classes.

- **PR-value** – This is a custom metric combining precision and recall into a single value. It takes inspiration from the PR curve, but it is used as a scalar value only. This metric treats the false positive rate and the false negative rate as equally hurtful to the model performance (see Eq. (4.4) below).

$$PR = P \cdot R \tag{4.4}$$

  This is the main metric used to assess the model performance on the sample classification level. Since the classification must be nearly perfect to work well for the keypoint detection, this metric is an adequate measure that is hard enough to maximize. This metric is quite similar to the F-1 score. Having only learned about the F-1 score later, the PR metric was kept for consistent evaluation results.

All of the above metrics are calculated on a sample level and then aggregated over a body of samples. This body can span the whole dataset or a single target class. To better assess the model performance, both aggregation bodies are used.

Confusion Matrix

Confusion matrix C is a $i \times j$ matrix such that $C[i, j]$ is equal to the number of observations of the class i, predicted to be class j [27]. Confusion matrices are shown unnormalized in this text so that the proportions of the classes are clearly visible.

4.1.2 Keypoint Detection Performance

While the model is trained to classify image samples, its main task is to detect the keypoints of target classes in the full image. The following metrics evaluate the full image model predictions. For all the full image prediction metrics listed below, a lower value is better.

- **Pixel-wise distance error**

  Since the model prediction is a per-pixel classification, the most straightforward way to evaluate it is to do it at the pixel level. A mean square error (MSE) metric is used, which penalizes completely wrong predictions harder. For every predicted pixel of a keypoint class, the minimum distance to a keypoint of this class is calculated, as shown in the Eq. (4.5) below,

$$D = \sum_c \sum_i \min_k (\hat{y}_c^{(i)} - y_c^{(k)})^2, \tag{4.5}$$

  where y_c is a ground truth keypoint location of class c and $\hat{y}$ is the predicted pixel location.

  This error metric does not work with the background class, as the background prediction is complementary to the keypoint predictions. A fixed penalty P is used for classes with prediction pixels but no ground truth keypoints. The penalty value has been set at $P = 10^3$, which is punishing enough to render the prediction inaccurate[1]. This metric does not punish false negatives.

[1] Should the penalty value be interpreted as a misplaced prediction, it would mean a distance of $\sqrt{1000} \approx 31.6$ pixels to the nearest keypoint. This is a relatively close distance in a $\approx 1000 \times 750$ image. On the other hand, the total error is affected significantly because of the usually high prediction pixel counts. Higher penalty values had been used during development but they started to overshadow the distance error once the predictions have become less noisy.

The pixel-wise prediction is also affected by the scale at which the image is processed. Since the number of predicted pixels and their distance to the keypoints are both proportional to the input image size, the error is normalized by the square of the image scaling factor S (introduced in Section 3.1), to make the metric value scale-independent.

The result is also normalized by the total number of prediction pixels, so that larger activation regions do not get punished based on the predicted region size (quantity) but only based on the distance to the keypoints (quality).

$$MSE = \frac{D + P \cdot |\hat{Y}_{FP}|}{S^2 \cdot |\hat{Y}|}, \tag{4.6}$$

where $|\hat{Y}_{FP}|$ is the number of false-positive prediction pixels and $|\hat{Y}|$ is the number of total prediction pixels.

The resulting metric captures the scale-independent mean square distance to the nearest keypoint, additionally penalizing false positive predictions, normalized by the number of predicted pixels.

- **Keypoint-wise Distance Error**

  The distance error works similarly at the keypoint level but it can better evaluate false-positive and false-negative predictions. A MSE-based metric is used again. For every predicted keypoint, the minimum distance to a ground-truth keypoint of the same class is calculated. If there are fewer predicted keypoints than ground-truth keypoints, a fixed false-negative penalty is applied for each of the missing predictions.

$$D = \frac{1}{|C|} \sum_{c} \sum_{i} \min_{k} (\hat{y}_c^{(i)} - y_c^{(k)})^2, \tag{4.7}$$

  where y_c is a ground truth keypoint location of class c, $\hat{y}$ is the predicted keypoint location, and $|C|$ is the number of classes.

$$MSE = \frac{\overline{D}}{S}, \tag{4.8}$$

  where $\overline{D}$ is the mean of the total distance error, and S is the scaling factor.

  For simplicity, the Eq. (4.8) above omits the extra false-positive and the false-negative penalties (both set to 10^4). They are only used when there are keypoint predictions but no ground-truth keypoints or when there are ground-truth keypoints but no keypoint predictions. The penalties are multiplied by the absolute difference between the two.

  The keypoint-wise distance error is similar to the pixel-wise distance error but it is also averaged over the classes[2] and the scaling factor is used as a linear term, not quadratic.

- **Keypoint-counting Error**

[2]The additional averaging has been introduced erroneously but it only divides the output value by 7 (number of keypoint classes).

The keypoint counting error does not rigorously analyze false-positive and false-negative predictions. The metric captures the mean absolute error of the predicted per-class keypoint number, as shown in the Eq. (4.9). False-positive and false-negative predictions may cancel each other out.

$$C = \sum_{c} \Big|\Big||\hat{Y}_c| - |Y_c|\Big|\Big|, \tag{4.9}$$

where $|\hat{Y}_c|$ is the number of predicted keypoints and $|Y_c|$ is the number of ground-truth keypoints.

4.1.3 Crate Counting Performance

- **Crate-counting Failure Rate**

  When the crate counting fails completely, it returns the value -1. This case is monitored separately from an inaccurate count estimate, as it poses a bigger problem. The failure rate is expressed similarly to probability, with 1 being a failure for all predictions and 0 for none.

- **Crate-counting Error**

  This metric captures the mean absolute crate count prediction error. It only includes the non-failure prediction cases.

4.2 Applicability of Chosen Metrics

Since the metrics capture various aspects of the predictions and the prediction interpretations, they are best interpreted when viewed together.

Table 4.2: Comparison of the best models (columns) for each metric (rows) on validation data. The best value achieved for a given metric is in **bold**. Metrics are listed in the order in which they were introduced, their names have been shortened for readability. For the crate-counting MAE, models with a failure rate of over 37.5 % were excluded. *The model with the minimal Crate Counting Failure Rate is also the final model evaluated in Section 4.6.

	PR_{max}	$Pix\text{-}Dist_{min}$	$Pt\text{-}Dist_{min}$	$Pt\text{-}Cnt_{min}$	$Cnt\text{-}Fail_{min}$*	Cnt_{min}
PR-Value	**0.965**	0.919	0.941	0.941	0.957	0.953
Pix-Dist	1,460,504	**22,801**	40324	40324	64,935	103,440
Pt-Dist	15,251	15,089	**6455**	**6455**	11,417	13,566
Pt-Cnt	6.5	7.65	**3.5**	**3.5**	5.5	5.25
Cnt-Fail	0.25	0.75	0.5	0.5	**0.125**	0.375
Cnt-Err	8.625	42.75	7.5	7.5	16.429	**7.95**

As shown in Figure 4.1 below, the crate-counting failure rate and the crate-counting error are hard to minimize at the same time. Any total failure of the counting should be

unacceptable and an erroneous prediction is still better than none, which is why the final model in Section 4.6 has been chosen from the bottom row of the chart.

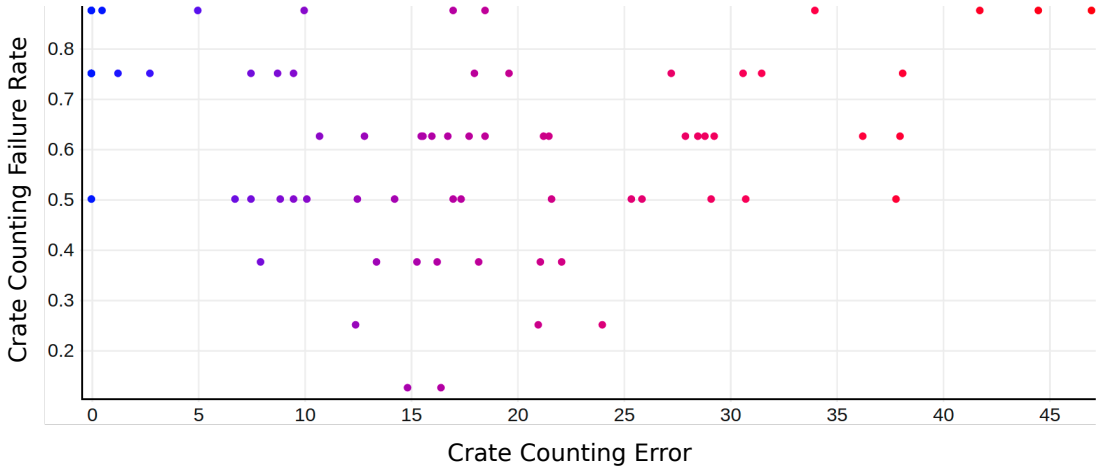


Figure 4.1: Scatter plot of crate-counting failure rate and error for validation data.

4.3 Model Input Size

The effect of the model input size has been investigated during the whole development. Even though only

Based on the odd dilation rate growth, further comparison models were designed. They use such model input sizes that make for regular repeated blocks based on 3×3 convolutions, similar to the main 64x model design. These models use the input sizes of 51x, 73x, and 99x. Together with the 32x and 64x models, 5 model input sizes are compared.

Many variants have been run and the best-performing models are presented in Table 4.3 below. Even though comparable models have been run, the best models for each input size are often not the largest ones. Only the crate-counting metrics are compared, as they are the most representative.

Table 4.3: Comparison of the best models for all tested input sizes. Models were chosen based on the validation data crate-counting failure rate.

Input size	Layers	Parameters	Crate-count MAE train	Crate-count FailRate train	Crate-count MAE val	Crate-count FailRate val
32x	12	0.45 M	13	0.72	23	0.25
51x	7	0.15 M	8.6	0.39	21	0.25
64x	9	4 M	**4.42**	0.17	16.43	**0.125**
73x	8	1.2 M	4.77	0.28	**12.41**	0.25
99x	9	0.45 M	9.63	**0.15**	24.01	0.25

4.4 Class Weights

During the development, it has shown that the inverse frequency (inv_freq) class weights do not help the model performance. Using uniform class (none) weights has become the default choice. To reevaluate these choices, together with the effective number of samples (eff_num) class weights scheme, experiments have been run.

For the effective number of samples class weights, $\beta = 0.999$ has been used. Using a β value based on the number of class samples has led to weights with the same ratios as the inverse frequency class weights and the weights were way lower, making the training even slower. The class weights comparison can be seen in Figure 4.2 below.

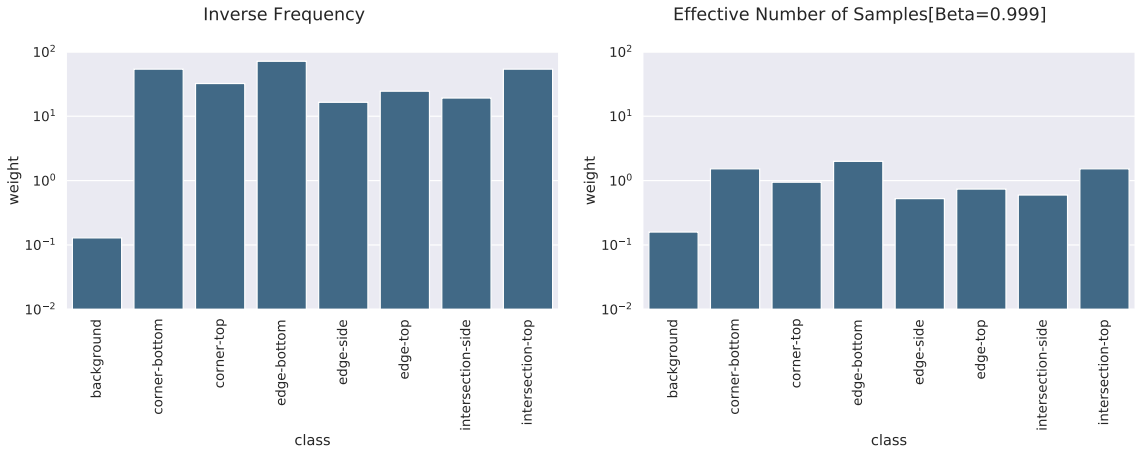


Figure 4.2: Comparison of used class weights.

As can be seen in Fig. 4.3, using the inverse frequency class weights scheme fails even at the sample level. The weights based on the effective number of samples have performed almost as well as using uniform weights. Closer inspection of the training logs has also shown that the Recall metric gets favored and usually reaches higher training values than the Precision. The differences are often negligible on the validation data, though. The best-performing model using the eff_num weights has been the best 73x model mentioned in Table 4.3

4.5 Scale-Invariance

Since the keypoints are mostly scale-invariant, the network should be able to detect them at various scales. The performance at various scales should also be improved by the zoom augmentation.

Predictions evaluated at multiple scales (shown in Fig. 4.4) show that the model is capable of detecting keypoints even at different scales, proving the original design assumption right. The background detection robustness does not transfer as well, making the predictions noisy at scales further from the original one.

4.6 Final Model Results

The final model uses the number of channels $C = 32$, resulting in total 4 million parameters. Uniform weights have been used for the model training.

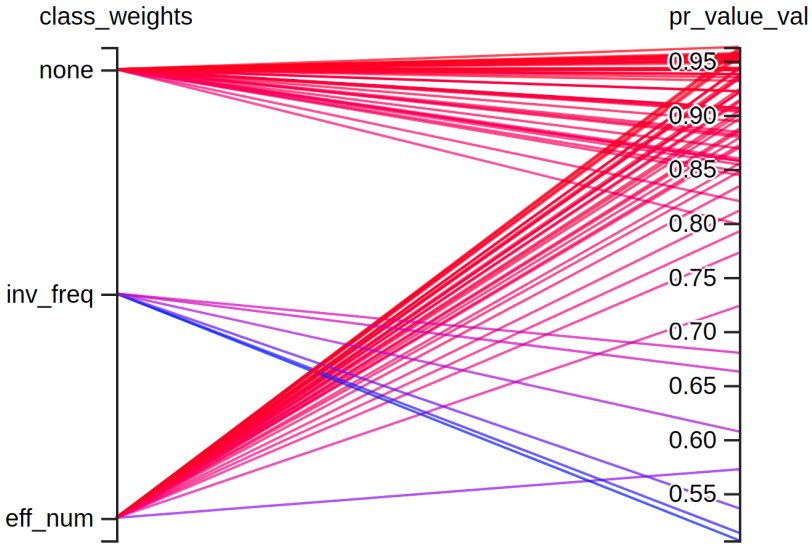


Figure 4.3: Parallel coordinates plot: comparison of the validation PR-value for models using different class weights. The comparison includes over 75 trained models.

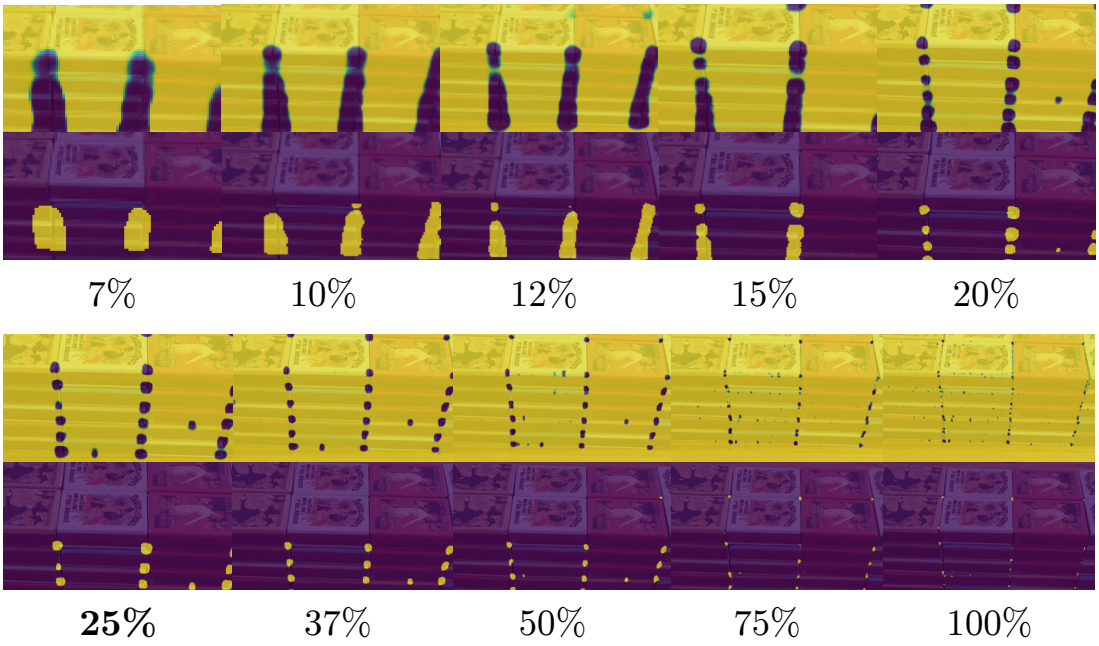


Figure 4.4: Comparison of background and intersection-side predictions for various input image scales. The predictions were run by the final model described in Section 4.6 and it has been trained on a scale $S = 0.25$ (in **bold**).

The final model was chosen based on the crate-counting metrics results. This has been the only model to consistently perform on both the training and validation dataset. As can be seen in Table 4.4 below, the only validation performance competitor fails twice as much at the training data. Values of the final model metrics are also shown in Table 4.2.

Table 4.4: Comparison of the 2 best models for the crate-counting.

Crate-count MAE train	Crate-count FailRate train	Crate-count MAE val	Crate-count FailRate val
4.43	**0.174**	16.4	**0.125**
8.38	0.348	**14.9**	**0.125**

Since the training run was performed, a minor outlier detection improvement has been added, which has pushed the validation crate-counting MAE down to 11.14 and the training crate-counting failure rate down to 10.8 %. The results suggest that further development of the counting algorithm could improve the results even more.

4.7 Summary

In the final experiment setting, only 2 out of the 76 runs achieved a final crate-counting failure rate of 0.125 on the validation data. Overall, the models are often inconsistent in their performances during training – this becomes visible when multiple metrics of a model are compared. The experiments have also shown that even models with fewer parameters can achieve competitive results. Still, the largest model (parameters-wise) wins by a sizeable margin at the main task of crate-counting.

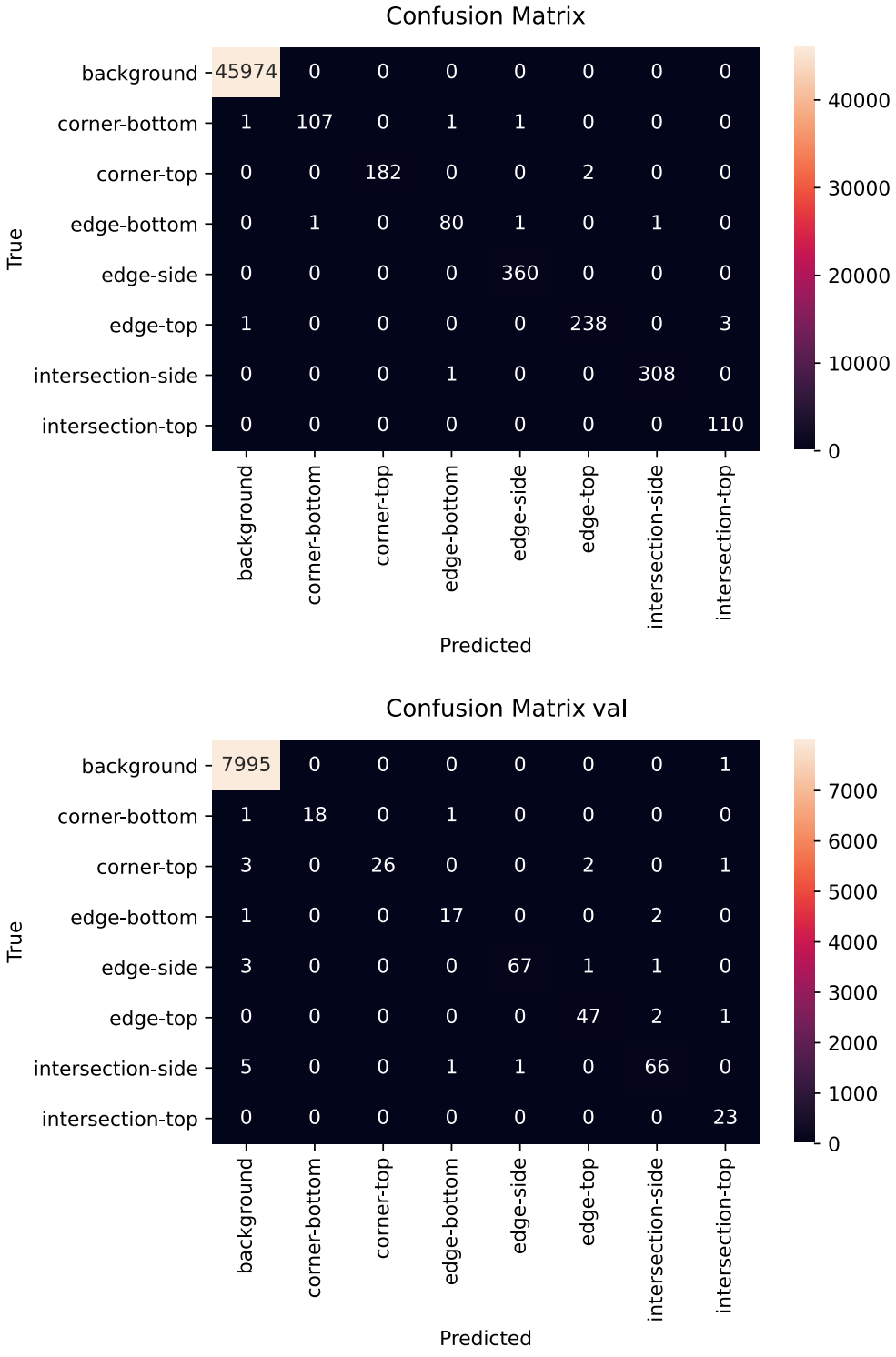


Figure 4.5: Confusion matrices for the final model. The top matrix is shown for training data, the bottom matrix for validation data.

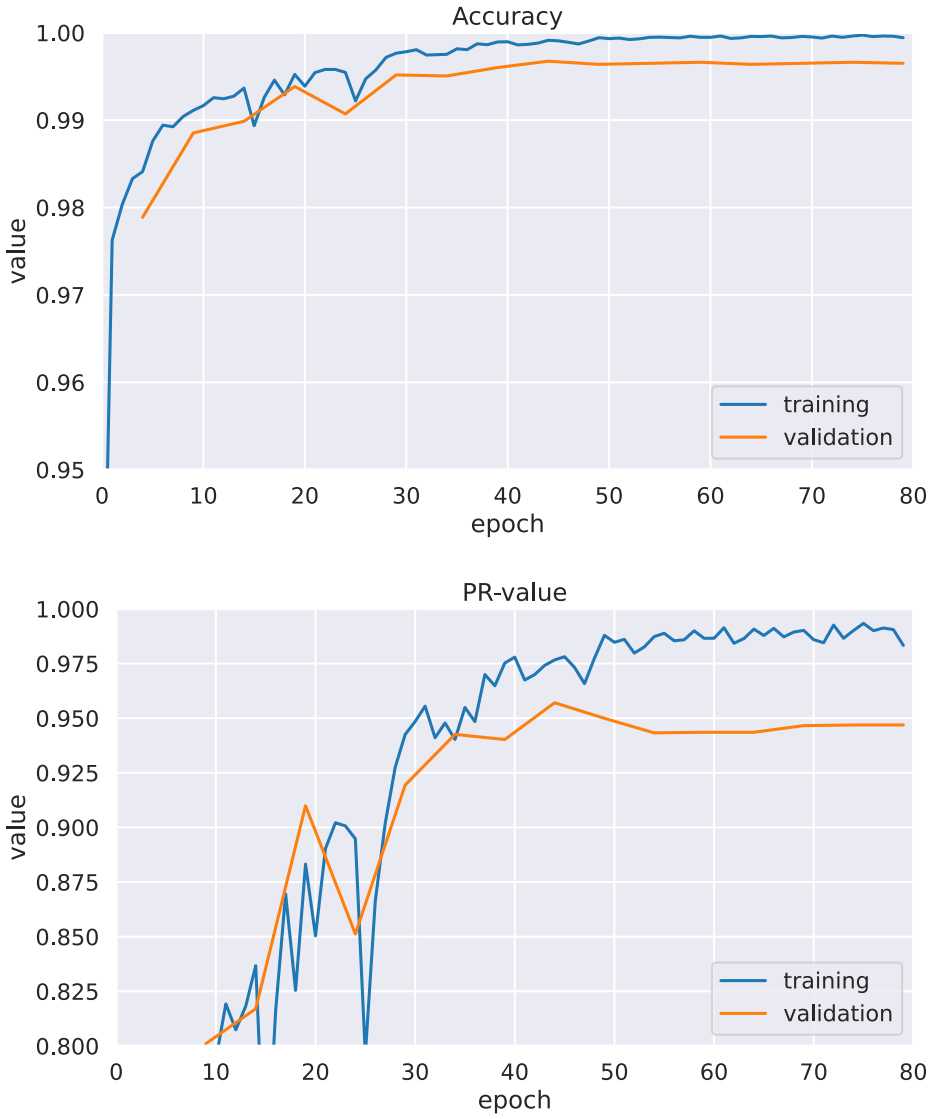


Figure 4.6: Sample-level metrics for the final model.

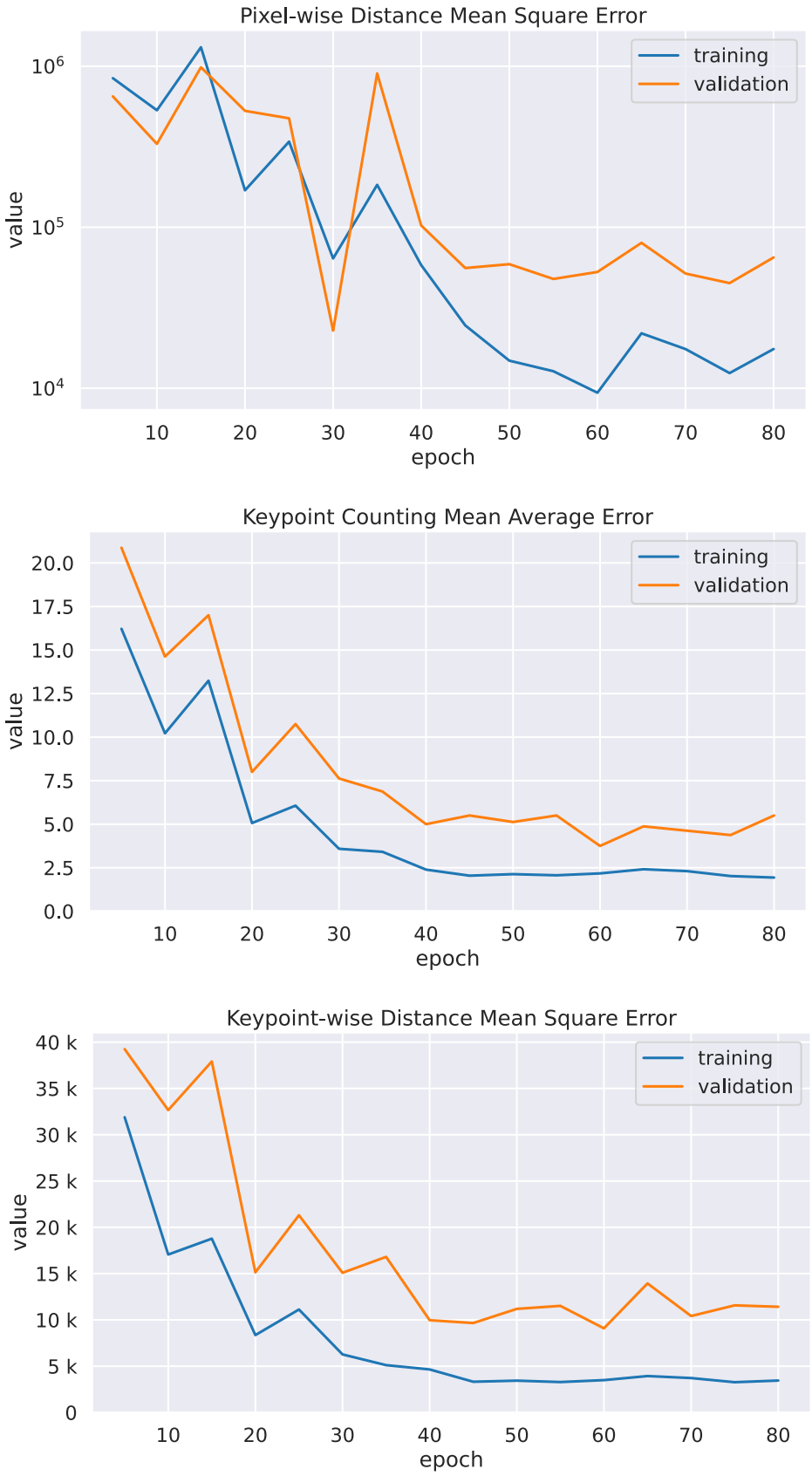


Figure 4.7: Keypoint detection metrics for the final model. Note: $k = 1000$ in the bottom chart.

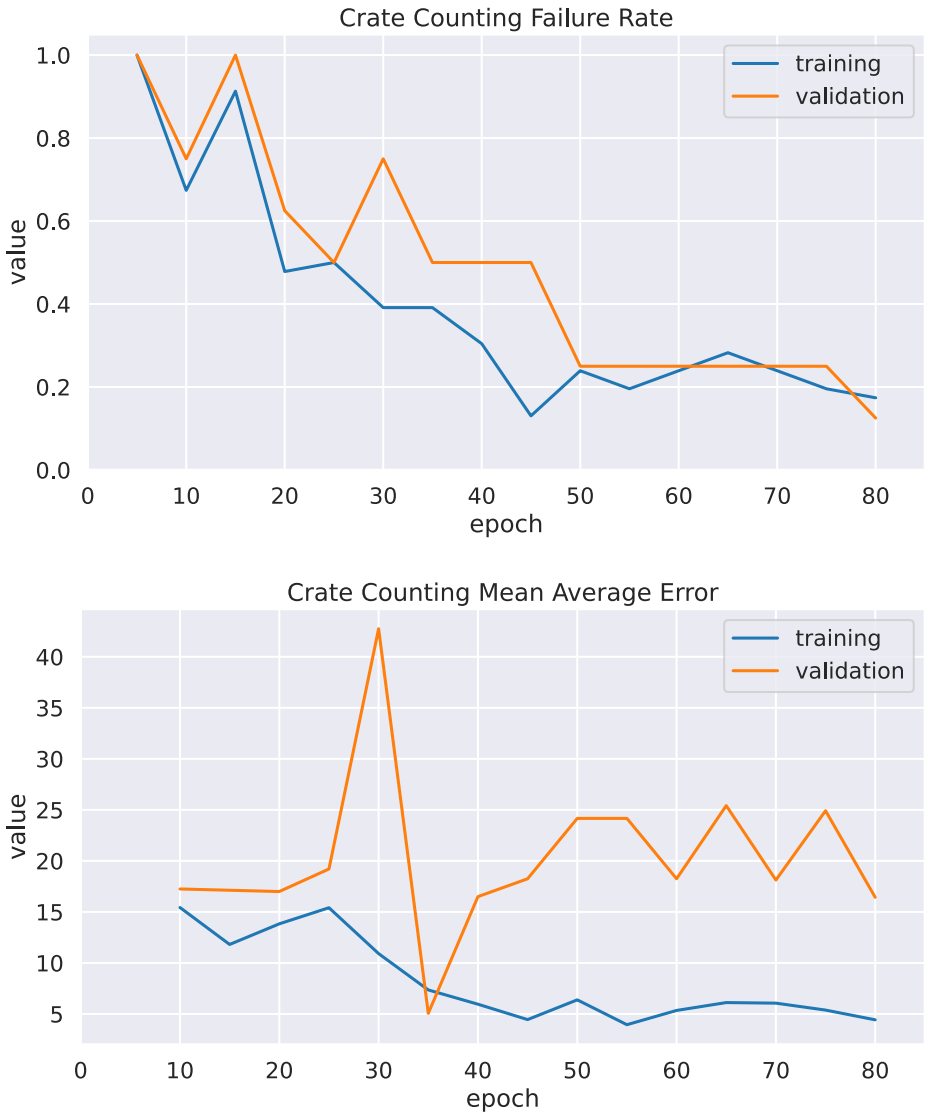


Figure 4.8: Crate-counting metrics for the final model. Counting error is not shown at epochs where the failure rate is 1.

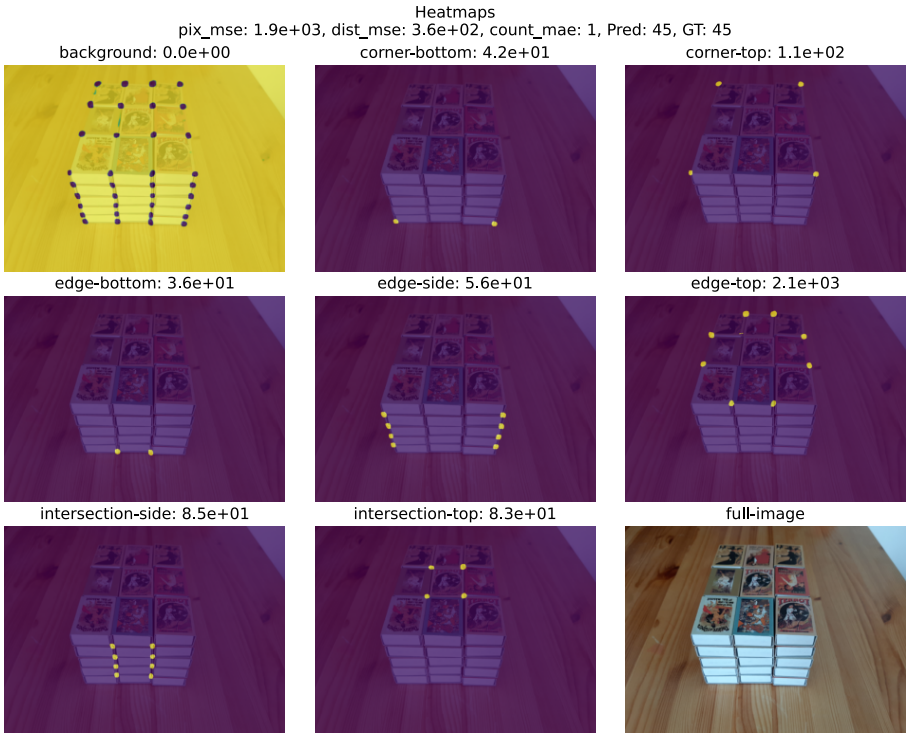


Figure 4.9: A successful prediction with correctly estimated keypoints.

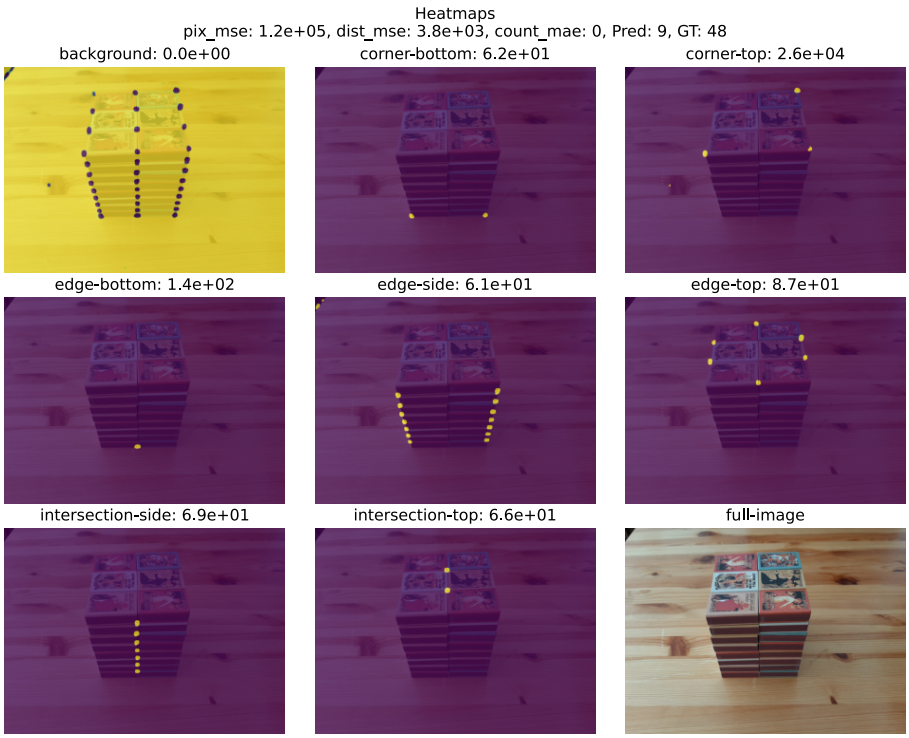


Figure 4.10: Prediction for which the crate-counting estimate is wrong.

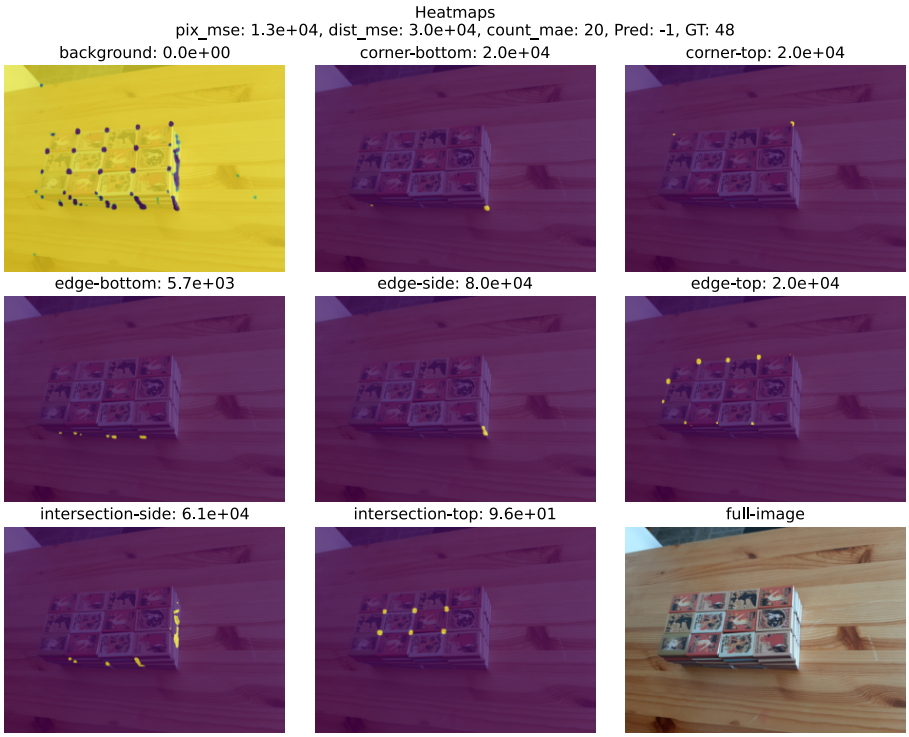


Figure 4.11: Prediction for which the crate-counting failed completely.

Chapter 5

Conclusion

Over the course of development, many aspects of CNNs were carefully studied and practically explored, leading to a strong baseline for the network design and implementation. A custom dataset of matchbox photos was created for the development purposes. A crate-counting solution was designed and implemented, using a keypoint detection network and a crate-counting algorithm. The keypoint detection network is an unconventional fully convolutional classification-based detection architecture, so it is worth noting that its performance on the task was not a given. The crate-counting algorithm stands on the quality of the keypoint predictions and it serves the task well. The best achieved results for the crate counting are a 12.5 % failure rate with a 11.14 MAE.

Various experiments have been performed with both the main design of the solution and its alternatives. Their detailed evaluation has confirmed the suitability of the final design choices.

There are many possible routes for future development. Since data is so key in deep learning, acquiring a proper dataset would be the first choice for further development. Possible improvements to the current CNN solution include using Focal loss or dataset resampling for improving the classification performance. Much experimentation has also been done using manual hyperparameter tuning – using a modern ML experiment management infrastructure with hyperparameter tuning or AutoML could have saved months of development time. Using online background sample generation would also make the dataset richer by using more than just a fixed subset of sampled background regions. Much development could be done in the crate counting. Apart from improving the algorithm to be more robust, a question can be raised, whether this part could be solved using deep learning too.

Given the limitations of the implemented solution, many considered approaches would mean building the whole system from the ground up again. Point-based object detectors could be used for the keypoint detection, similarly to this year's TetraPackNet [5]. Crate edges could also be detected instead of the corner keypoints, which would allow for easier interpretation of the block structure and simpler distinguishing between multiple blocks of crates. No specific further work is currently planned though, because of the lack of data.

Bibliography

[1] Bowyer, K. W., Chawla, N. V., Hall, L. O. and Kegelmeyer, W. P. SMOTE: Synthetic Minority Over-sampling Technique. *CoRR*. 2011, abs/1106.1813. Available at: http://arxiv.org/abs/1106.1813.

[2] Byrd, J. and Lipton, Z. C. *What is the Effect of Importance Weighting in Deep Learning?* 2019. Available at: https://arxiv.org/pdf/1812.03372.pdf.

[3] Cui, Y., Jia, M., Lin, T.-Y., Song, Y. and Belongie, S. Class-Balanced Loss Based on Effective Number of Samples. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019. Available at: https://openaccess.thecvf.com/content_CVPR_2019/papers/Cui_Class-Balanced_Loss_Based_on_Effective_Number_of_Samples_CVPR_2019_paper.pdf.

[4] Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q. et al. CenterNet: Keypoint Triplets for Object Detection. *CoRR*. 2019, abs/1904.08189. Available at: http://arxiv.org/abs/1904.08189.

[5] Dörr, L., Brandt, F., Naumann, A. and Pouls, M. *TetraPackNet: Four-Corner-Based Object Detection in Logistics Use-Cases*. 2021. Available at: https://arxiv.org/pdf/2104.09123.pdf.

[6] Gal, Y. *Uncertainty in Deep Learning*. 2016. Dissertation. University of Cambridge. Available at: https://mlg.eng.cam.ac.uk/yarin/thesis/thesis.pdf.

[7] Girshick, R. B. Fast R-CNN. *CoRR*. 2015, abs/1504.08083. Available at: http://arxiv.org/abs/1504.08083.

[8] Girshick, R. B., Donahue, J., Darrell, T. and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*. 2013, abs/1311.2524. Available at: http://arxiv.org/abs/1311.2524.

[9] Goodfellow, I., Bengio, Y. and Courville, A. *Deep learning*. 1st ed. MIT press, 2016. Adaptive computation and machine learning series. ISBN 9780262035613. Available at: https://www.deeplearningbook.org/.

[10] Géron, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*. 2nd ed. O'Reilly, 2019. ISBN 9781492032649.

[11] He, K., Gkioxari, G., Dollár, P. and Girshick, R. B. Mask R-CNN. *CoRR*. 2017, abs/1703.06870. Available at: http://arxiv.org/abs/1703.06870.

[12] He, K., Zhang, X., Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. *CoRR*. 2015, abs/1512.03385. Available at: http://arxiv.org/abs/1512.03385.

[13] Hinton, G. *Can the brain do backpropagation*. May 2019. Available at: https://can-acn.org/meeting-2019/2019-public-lecture-geoffrey-hinton/.

[14] Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L. et al. A Survey of Deep Learning-Based Object Detection. *IEEE Access*. Institute of Electrical and Electronics Engineers (IEEE). 2019, vol. 7, p. 128837–128868. DOI: 10.1109/access.2019.2939201. ISSN 2169-3536. Available at: http://dx.doi.org/10.1109/ACCESS.2019.2939201.

[15] Johnson, J. *EECS 498-007 / 598-005 Deep Learning for Computer Vision*. 2019. Available at: https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2019/.

[16] Khaki, S., Pham, H., Han, Y., Kuhl, A., Kent, W. et al. Convolutional Neural Networks for Image-Based Corn Kernel Detection and Counting. *Sensors*. 2020, vol. 20, no. 9. DOI: 10.3390/s20092721. ISSN 1424-8220. Available at: https://www.mdpi.com/1424-8220/20/9/2721.

[17] Kreiss, S., Bertoni, L. and Alahi, A. PifPaf: Composite Fields for Human Pose Estimation. *CoRR*. 2019, abs/1903.06593. Available at: http://arxiv.org/abs/1903.06593.

[18] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*. May 2017, vol. 60, no. 6, p. 84–90. DOI: 10.1145/3065386. ISSN 0001-0782. Available at: https://dl.acm.org/doi/10.1145/3065386.

[19] Law, H. and Deng, J. CornerNet: Detecting Objects as Paired Keypoints. *CoRR*. 2018, abs/1808.01244. Available at: http://arxiv.org/abs/1808.01244.

[20] LeCun, Y. *Self-Supervised Learning*. February 2020. AAAI-20. Available at: https://www.youtube.com/watch?v=UX8OubxsY8w.

[21] Li, Y., Zhang, X. and Chen, D. *CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes*. 2018. Available at: https://arxiv.org/pdf/1802.10062.pdf.

[22] Lin, T., Goyal, P., Girshick, R. B., He, K. and Dollár, P. Focal Loss for Dense Object Detection. *CoRR*. 2017, abs/1708.02002. Available at: http://arxiv.org/abs/1708.02002.

[23] Long, J., Shelhamer, E. and Darrell, T. Fully Convolutional Networks for Semantic Segmentation. *CoRR*. 2014, abs/1411.4038. Available at: http://arxiv.org/abs/1411.4038.

[24] Mendes, C. C. T., Frémont, V. and Wolf, D. F. Exploiting fully convolutional neural networks for fast road detection. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, p. 3174–3179. DOI: 10.1109/ICRA.2016.7487486. Available at: https://ieeexplore.ieee.org/document/7487486.

[25] Newell, A., Yang, K. and Deng, J. *Stacked Hourglass Networks for Human Pose Estimation*. 2016. Available at: https://arxiv.org/pdf/1603.06937.pdf.

[26] Oquab, M., Bottou, L., Laptev, I. and Sivic, J. Is object localization for free? - Weakly-supervised learning with convolutional neural networks. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, p. 685–694. DOI: 10.1109/CVPR.2015.7298668. Available at: https://www.di.ens.fr/~josef/publications/Oquab15.pdf.

[27] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, p. 2825–2830. Available at: https://github.com/scikit-learn/scikit-learn/blob/15a949460dbf19e5e196b8ef48f9712b72a3b3c3/sklearn/metrics/_classification.py.

[28] Ren, S., He, K., Girshick, R. B. and Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*. 2015, abs/1506.01497. Available at: http://arxiv.org/abs/1506.01497.

[29] Shrivastava, A., Gupta, A. and Girshick, R. B. Training Region-based Object Detectors with Online Hard Example Mining. *CoRR*. 2016, abs/1604.03540. Available at: http://arxiv.org/abs/1604.03540.

[30] Simonyan, K. and Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: Bengio, Y. and LeCun, Y., ed. *3rd International Conference on Learning Representations, ICLR 2015,San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. Available at: http://arxiv.org/abs/1409.1556.

[31] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E. et al. Going Deeper with Convolutions. *CoRR*. 2014, abs/1409.4842. Available at: http://arxiv.org/abs/1409.4842.

[32] Tan, M. and Le, Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *CoRR*. 2019, abs/1905.11946. Available at: http://arxiv.org/abs/1905.11946.

[33] Yu, F. and Koltun, V. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2016. Available at: https://arxiv.org/pdf/1511.07122.pdf.

[34] Zhang, A., Lipton, Z. C., Li, M. and Smola, A. J. *Dive into Deep Learning*. 2020. https://d2l.ai.

[35] Zhou, B., Khosla, A., A., L., Oliva, A. and Torralba, A. Learning Deep Features for Discriminative Localization. *CVPR*. 2016. Available at: http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf.

[36] Zhou, X., Wang, D. and Krähenbühl, P. Objects as Points. *CoRR*. 2019, abs/1904.07850. Available at: http://arxiv.org/abs/1904.07850.