Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Detection of Particular Objects in Images

Askar Kassymgaliyev

Supervisor: doc. Georgios Tolias, Ph.D.
May 2024

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kassymgaliyev Askar**	Personal ID number:	**510643**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence and Computer Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Detection of Particular Objects in Images

Bachelor’s thesis title in Czech:

Detekce konkrétních objektů v obrázcích

Guidelines:

Recent advances in deep learning for computer vision have significantly improved the performance of object detection. This is the case for learning detection of generic object classes, such as bicycle, horse, chair, monitor, etc., while less attention is given to particular objects, i.e., this bicycle, this chair. This is exactly the focus of this project, to learn how to detect particular objects given one or few examples. The goal will be pursued by training Convolutional Neural Networks. The starting point of the thesis is the OS2D-ECCV2020, and the overall goal is to seek performance improvements and/or simplifications of the overall pipeline. Particular attention is to be paid to the costly part of the spatial transformer and ways of dispensing with its necessity.
1. Read OS2D and understand it in detail. The thesis should provide a complete description, trying to be more detailed and/or clear than the original description, wherever this is needed.
2. Using the publicly available implementation and the instance-level datasets used in the ECCV paper, reproduce the reported experiments.
3. Seek performance improvements and/or simplifications of the overall pipeline. This goal is set due to the increased complexity, in terms of computing and design, of the original approach. A candidate direction, which may vary during the semester, is to remove the spatial transformer and replace it with deterministic augmentations of the class images at the input.

Bibliography / sources:

[1] Osokin et al, , ECCV2020, OS2D: One-Stage One-Shot Object Detection by Matching Anchor Features
[2] Dwibedi, Misra, Hebert, ICCV'17, Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection.
[3] Ammirato, Fu, Shvets, Kosecka, Berg, arxiv'18, Target Driven Instance Detection

Name and workplace of bachelor’s thesis supervisor:

doc. Georgios Tolias, Ph.D. Visual Recognition Group FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **12.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

doc. Georgios Tolias, Ph.D. Supervisor’s signature	prof. Dr. Ing. Jan Kybic Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I want to express my deepest appreciation to my supervisor, doc. Georgios Tolias, for his guidance and support. I am also thankful to my parents and friends, who supported me throughout my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 24th of May 2024

Signature. . . . .

Abstract

In recent years, many advancements have been made in object detection, yet there is still no complete solution as the problem is deeply ingrained in the "understanding" of what makes an object an object. The scope of this work lies in examining instance-level object detection as opposed to semantic or category-level object detection. We also built a detector of such objects based on an existing method that uses a geometric alignment module by removing that module in favor of a simpler technique.

Keywords: instance level object detection, one shot, feature matching

Supervisor: doc. Georgios Tolias, Ph.D.

Abstrakt

V posledních letech bylo dosaženo mnoha pokroků v detekci objektů, ale stále neexistuje úplné řešení, protože problém je hluboce zakořeněn v „pochopení“, co dělá objekt objektem. Rozsah této práce spočívá ve zkoumání detekce objektů na úrovni instancí, na rozdíl od sémantické nebo kategorické detekce objektů. Také jsme sestavili detektor takových objektů na základě existující metody, která využívá modul geometrického zarovnání, tím, že jsme tento modul odstranili ve prospěch jednodušší techniky.

Klíčová slova: instance level object detection, one shot, korelační párování

Překlad názvu: Detekce konkrétních objektů v obrázcích

Contents

Figures

Tables

Chapter 1
Introduction

Object detection is a fundamental task in computer vision. The goal is to find all objects of interest in an image and to classify them. It is efficiently solved by deep Convolutional Neural Networks (CNNs), due to their ability to learn relevant features from images. The success of CNNs, however, relies heavily on a huge amount of training data with accurate annotations.

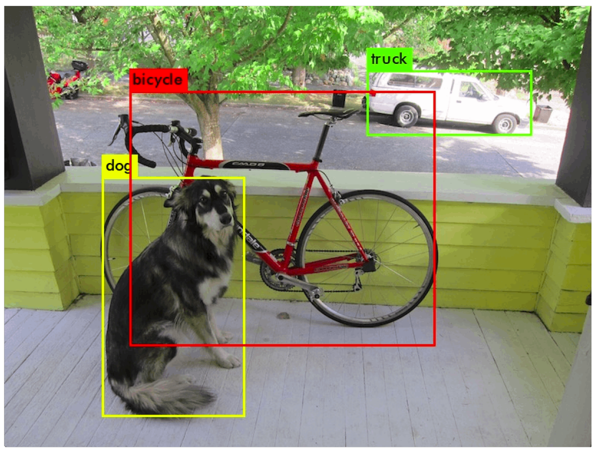


Figure 1.1: Object detection example, picture by Josef Redmon[1]

1.1 Problem definition

Instance-level object detection extends this task by recognizing a particular instance of an object, instead of just the category to which it belongs. As an example, instead of labeling a building as a "tower", instance-level label would be "*The Eiffel Tower*", or "*Fender Frankenstrat*" instead of "guitar".

Basic level	Building	Musical instrument
Fine grained	Tower	Electric guitar
Instance level	The Eiffel Tower	Frankenstrat

This context is especially relevant for practical applications encountered in real-world scenarios. Examples include robotic manipulation in factories, distinguishing between various traffic signs and lights for self-driving cars, or identifying individual groceries on supermarket shelves. In these environments, the objects involved are typically individual instances rather than broad categories.

Instance-level labels within the same category (my cup, your cup, their cup) sometimes show little variation, challenging the human eye to differentiate between them. Additionally, it is hard to come across a dataset appropriate for a certain environment, because each new environment with new instances requires expensive data collection and annotation. In contrast, one of the most widely used state-of-the-art datasets for general object detection, COCO[2], contains 1.5 million object instances and 80 object categories, which on average equals 18 750 objects per category.

The problem essentially boils down to the fact that there are many instances, and it is hard to differentiate between them. They might look dissimilar when they are actually the same due to factors like illumination, viewpoint, deformation, etc., or they might look very similar when they are, in fact, not the same. Furthermore, there is often little data about them.

The rest of the paper is organized as follows: Ch. 2 discusses topics related to one-shot object detection, we examine the work by Osokin et al.[3] in Ch. 3, and lastly propose a method that gets rid of the spatial transformer network in the said work in Ch. 4.

1.2 Related work

The field of instance-level object detection is not studied as thoroughly as regular object detection. The work by Ammirato et al.[4] focuses on instance-level object detection by performing feature comparisons between the target

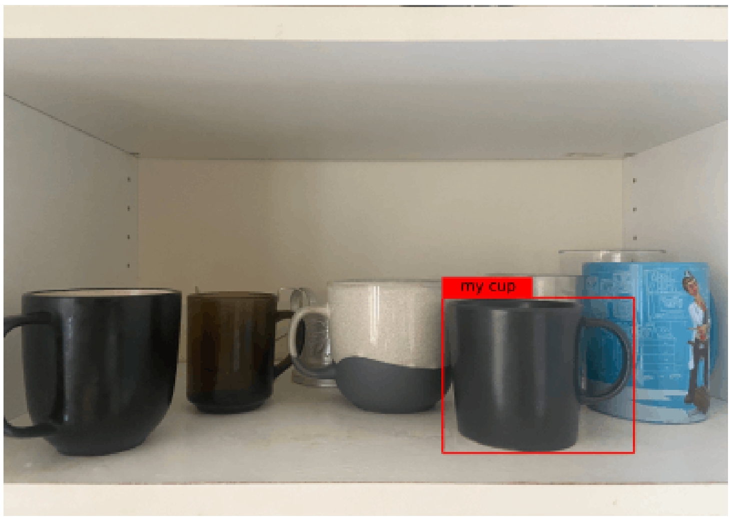


Figure 1.2: Instance-level detection example. The task is to detect my cup among other cups

and input feature maps to focus on the specific classes targeted in the image.

The work by Hsieh et al.[5] tackles the problem of one-shot object detection in a two-stage manner, using an attention mechanism to influence the proposed regions by the Regional Proposal Network (RPN). Their model not only demonstrates strong performance on the PASCAL Visual Object Classes (VOC)[6] and COCO[2] datasets but also achieves decent results when trained on instance-level datasets.

Dwibedi et al.[7] approach the problem of sparse data directly. They introduce a simple yet effective method for generating annotated instance datasets for object detection by "cutting" and "pasting" objects on random backgrounds. This newly generated data improves performance when combined with real images.

Chapter 2

Theoretical Background for One-Shot Object Detection

One-shot object detection is a method for detecting an object within an image with as little as one example. By combining it with metric learning, it has the potential to better generalize to new, unseen data. These qualities make them beneficial for the task we are dealing with.

Similarly to general object detectors based on deep Neural Networks (NNs), one-shot detectors fall into one of two categories: one-stage and two-stage. A common practice in one-shot detection is using weights from a model trained on a larger dataset and incorporating them into a backbone network for extracting features from images. Consequently, the weights of the transferred model can be unfrozen and further fine-tuned during training.

2.1 Transfer learning

Transfer learning is a machine learning technique in which a model created for one job is used in another job as the foundation for a new model.

Given the vast resources required to develop and train NN models for these problems and the enormous jumps in skill that they provide, it is not uncommon to use pre-trained models as the starting point in deep learning.

The authors of[8] describe three common measures by which transfer might improve learning:

- The initial skill of the model.
- The time it takes to train the model.
- The converged performance of the model.

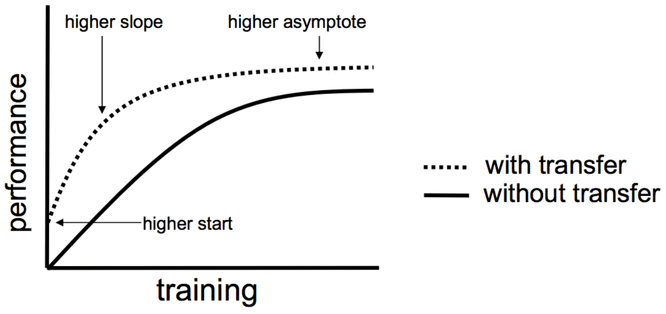


Figure 2.1: Three ways in which transfer might improve learning, picture taken from[8]

2.2 Metric learning

Contrary to classification, which primarily learns to distinguish between classes, metric learning focuses on understanding the relationships and similarities between different classes. It does so by learning the distance between them in the feature space as shown in Fig. 2.2.

As a result, it moves similar classes closer together and pushes others further apart. This makes it handy for open-set problems, where the testing sets involve unseen data during training.

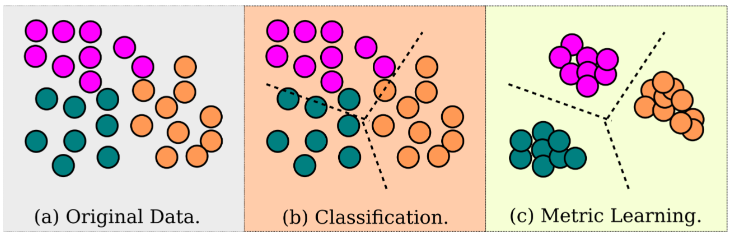


Figure 2.2: Difference between classification and metric learning approaches[9]

2.3 Two-stage detectors

Two-stage detectors operate in two stages. The first one, often referred to as the region proposal stage, narrows down the potential space where an object could be located in an image, and the second stage is a CNN that classifies each proposed region. These detectors achieve high accuracy at the cost of speed due to the sequential nature of the architecture.

In 2014, Girshick et al.[10] introduced R-CNN (Regions with CNNs) at that time, achieving mean Average Precision (mAP) of 58.5% on the PASCAL VOC 2007 dataset, which was a 24.8% increase from the previous best method[11]. For proposing regions, it uses a selective search algorithm. After that, a CNN extracts feature vectors from the proposed regions. Then, each feature vector

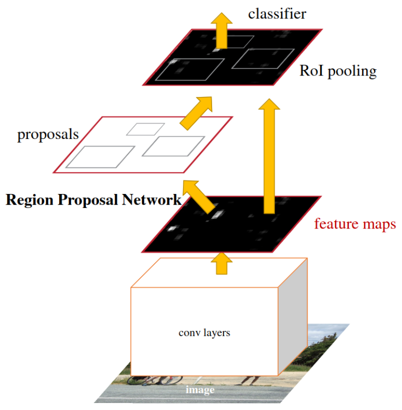


Figure 2.3: Faster R-CNN[13] architecture

is fed into a support vector machine for classification. Since the selective search algorithm generates approximately 2 000 region proposals, R-CNN is quite slow and unsuitable for real-time object detection.

Following R-CNN, Fast R-CNN[12] improves upon R-CNN by using a shared convolutional layer to process the entire image and all regions of interest rather than processing each region independently.

Faster R-CNN[13] further increased both the speed and the accuracy, achieving 73.2% mAP on the Pascal VOC 2007 dataset. The main idea of Faster R-CNN is to use Regional Proposal Network (RPN) instead of the selective search algorithm in previous ones. The architecture of Faster R-CNN can be seen in Figure2.3. Other relevant methods include Mask-RCNN[14], useful for instance segmentation.

2.4 One-stage detectors

One-stage detectors combine the previous two steps into one, they require only a single pass through the NN and predict all the bounding boxes in one go.

Redmon et al.[15] presented the YOLO (You Only Look Once) algorithm in 2016. It could process images in real-time at 45 frames per second and score 52.7% mAP on Pascal, which was more than twice as accurate as prior work in real-time detection[15]. These detectors have a reputation for achieving higher inference speeds at the expense of accuracy. Despite this, a newer version of YOLO released recently in 2022 outperformed most of the existing object detectors in terms of both accuracy and speed[16].

The key idea here is the use of a grid with anchor boxes. The input image is divided into smaller parts called grid cells. Anchor boxes are multiples of grid cells' widths and heights. Typically, a grid cell contains several anchor boxes of different proportions. The grid puts a constraint on where an object

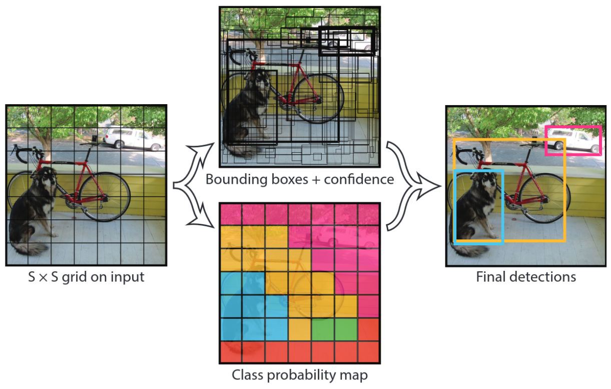


Figure 2.4: YOLO[15] model

could be, and anchor boxes help the cell predict the shape of an object.

After extracting a feature map $F \in \mathbb{R}^3$ from an image $I \in \mathbb{R}^3$, each grid cell $(x, y) \in F$ with A anchor boxes is responsible for detecting:

- N numbers for N classes ranged in between [0, 1] such that their sum is

  $$\sum_{i=1}^{N} n_i = 1, \;\; n \in N$$

  It is used as a conditional class probability, $P(Class_i|Object)$.

- A * 1 confidence score in range [0, 1]. They indicate how confident the detector is about the presence of an object at that position for each anchor box, $P(Object)$.

- A * 4 bounding box offsets $\Delta x, \Delta y, \Delta w, \Delta h$.

Class-specific confidence scores are then calculated for each predicted box.

$$P(Class_i|Object) * P(Object) * IOU_{pred}^{truth}$$

where IOU stands for Intersection Over Union of the predicted box and the ground truth box.

An important note here is that since there are 4 offset parameters for each anchor box for a given grid cell, the model makes predictions about the bounding box relative to its anchor box. If the cell is offset from the top left corner of the image by c_x, c_y and the anchor box has width and height a_w, a_h, then the predictions correspond to

$$x = \sigma(\Delta x) + c_x$$
$$y = \sigma(\Delta y) + c_y$$
$$w = a_w e^{\Delta w}$$
$$h = a_h e^{\Delta h}$$

Where $\sigma(z) = \frac{1}{1+e^{-z}}$ is a logistic function, which maps all real numbers to range $[0, 1]$

This procedure is relevant for the older YOLO detectors, which viewed object detection as a multi-class problem instead of a multi-label one. It figuratively resembles how other successful one-stage methods such as SSD[17], Retina Net[18], and the newer versions of YOLO work.

Chapter 3

One-Stage One-Shot Object Detection by Matching Anchor Features

OS2D[3] is a one-shot, one-stage object detector based on using a CNN for geometric matching that will be referred to as TransformNet by Rocco et al.[19][20]. Its components are differentiable, which allows for end-to-end training. It uses two distinct models with pre-trained weights, as described earlier in Sect. 2.1.

The architecture consists of the following steps: 1) feature extraction, 2) correlation matching, 3) spatial alignment by using transformNet and 4) computing the recognition scores and bounding boxes based on the successful matches.

This chapter will provide an overview of the pipeline and emphasize some important aspects where the original paper's description was rather shallow, namely the alignment and resampling.

3.1 Feature extraction

Given an input image and a class image, Residual Network[21] (ResNet) pre-trained on ImageNet[22] classification dataset with shared weights extracts dense feature maps from both of them. $I \in \mathbb{R}^{w_i \times h_i \times d}$ and $C \in \mathbb{R}^{w_c \times h_c \times d}$ are extracted feature maps for input and class images respectively.

Due to the architecture of transformNet, one of the feature maps has to be fixed to a certain size. The input feature maps are of high resolution, and rescaling them would result in a major loss of details and aspect ratio, thus, class feature maps C are rescaled to a fixed size of $15 \times 15 \times d$ instead. This is done through bilinear resampling.

Bilinear resampling is a simple, fast, and differentiable method for transforming images using repeated linear interpolations in two dimensions. It can also be applied to feature maps.

It uses the 4 nearest pixel values located in diagonal directions from a given position to interpolate the intensity values of a pixel at that position. For example, when stretching an image, new pixels appear as a result. The values

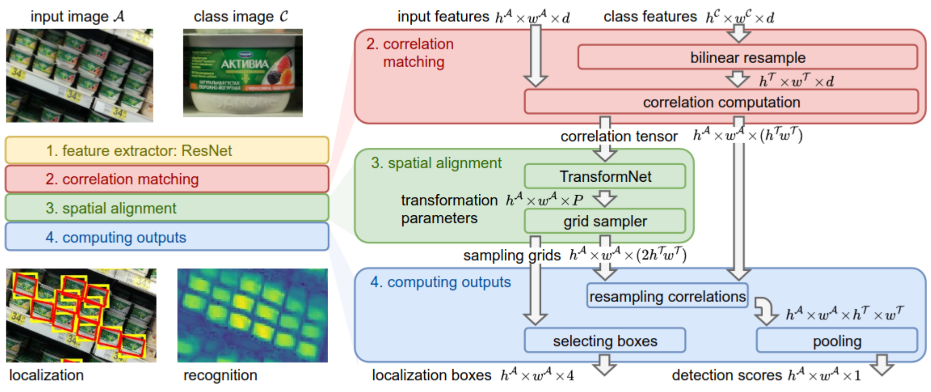

input image $\mathcal{A}$
class image $\mathcal{C}$
1. feature extractor: ResNet
2. correlation matching
3. spatial alignment
4. computing outputs
localization
recognition
input features $h^A \times w^A \times d$
class features $h^C \times w^C \times d$
2. correlation matching
bilinear resample
$h^T \times w^T \times d$
correlation computation
correlation tensor $h^A \times w^A \times (h^T w^T)$
3. spatial alignment
TransformNet
transformation parameters $h^A \times w^A \times P$
grid sampler
sampling grids $h^A \times w^A \times (2h^T w^T)$
4. computing outputs
resampling correlations
$h^A \times w^A \times h^T \times w^T$
selecting boxes
pooling
localization boxes $h^A \times w^A \times 4$
detection scores $h^A \times w^A \times 1$

Figure 3.1: OS2D model architecture, picture borrowed from the paper[3]

of these new pixels are "guessed" by using bilinear interpolation.

3.2 Correlation matching

After extracting both feature maps, the correlation tensor $c \in \mathbb{R}^{h_i \times w_i \times h_c \times w_c}$ is computed, which is a product of two feature maps divided by their l_2 norm along the feature dimension so that their length is equal to 1:

$$c[i,j,x,y] = \frac{\langle I_{ij}, C_{xy} \rangle}{\|I_{ij}\|_2 \|C_{xy}\|_2}$$

The higher the score, the more similar the two positions of the feature maps are. The maximum length is 1, hence the values are in the range [-1, 1].

3.3 Alignment

TransformNet is a Spatial Transformer Network(STN)[23], which acts as a differentiable visual attention mechanism. It is employed as follows:

The correlation tensor is passed through a Rectified Linear Unit (ReLU), discarding all the negative matches and fed into transformNet in the form of $c \in \mathbb{R}^{h_i \times w_i \times (h_c \times w_c)}$ in a fully-convolutional way. TransformNet then outputs $\tau_{ij} \in \mathbb{R}^{h_i \times w_i \times \theta}$ parameters for transformations at each location of the input feature map I. The number of predicted parameters θ can be either 4 or 6[1]. Those parameters are entries of a transformation matrix T_θ, that is responsible for all kinds of affine transformations such as translation, scaling, rotation and shearing.

[1]4 for simple affine transformations, meaning only translation and scaling are applied. In this case, θ_{12} and θ_{21} responsible for rotation are set to 0. 6 for full affine transformations

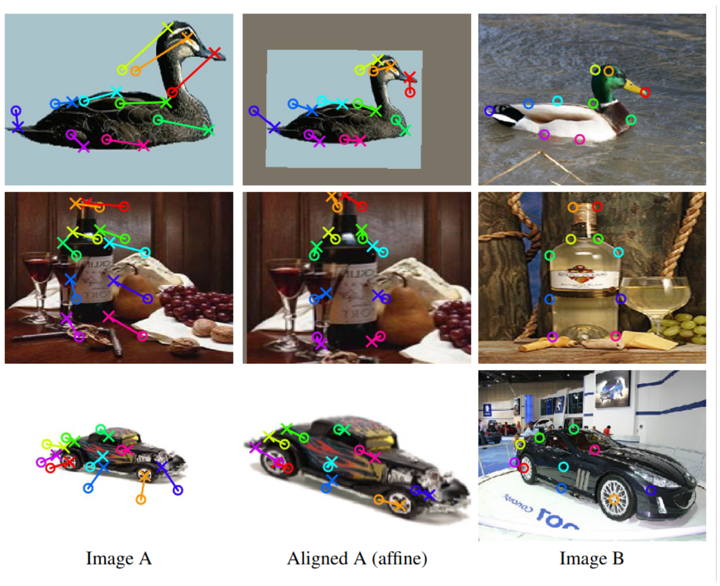


Figure 3.2: An example of how transformNet works. Ground truth matching keypoints are depicted as crosses and circles for images A and B, respectively. Keypoints of the same color are supposed to match after image A is aligned to image B. Picture source[19]

$$T_\theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix}$$

The weights released by Rocco et al. define the direction of transformations in the opposite direction, i.e., they align input I to class C. In order to use the weights, each transformation T_θ needs to be inverted. This is done by augmenting the matrix into 3×3 form and calculating its inverse:

$$T_\theta^{-1} = \begin{bmatrix} \theta'_{11} & \theta'_{12} & \theta'_{13} \\ \theta'_{21} & \theta'_{22} & \theta'_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

3.4 Resampling process

First, a parameterized sampling grid[2] $g \in \mathbb{R}^{h_i \times w_i \times h_c \times w_c \times 2}$ is created by applying the transformations τ_{ij} to a meshgrid[3] of size $w_c \times h_c$, the same size as C. The values in the meshgrid represent normalized coordinates of the class feature maps C ranging from -1 to 1. The values in the parameterized sampling grid then consist of those coordinates transformed, where each pair denotes the location of a pixel to sample from.

[2]https://pytorch.org/docs/stable/generated/torch.nn.functional.affine_grid.html
[3]Meshgrid is a matrix of coordinates.

In other words, g does not contain the actual values. The values for each individual pixel are then taken from and interpolated from the correlation tensor c in accordance with g.

A transformation T_{θ}^{-1} of a coordinate pair (x_m, y_m) from the meshgrid can be written as:

$$\begin{bmatrix} \theta'_{11} & \theta'_{12} & \theta'_{13} \\ \theta'_{21} & \theta'_{22} & \theta'_{23} \end{bmatrix} \begin{pmatrix} x_m \\ y_m \\ 1 \end{pmatrix}$$

1 is added to the coordinate pair to make it homogeneous. The last column of the transformation is responsible for the translation.

Each transformation is defined with respect to the local coordinate system. Operations below convert them to normal, with respect to input feature map I:

$$x_{global} = x_{local} * \frac{w_c}{2} + x_{centerpoint}$$

$$y_{global} = y_{local} * \frac{h_c}{2} + y_{centerpoint}$$

The translation factors are the center points of the anchor boxes with respect to I.

Then, bilinear sampling is performed at each channel layer $(h_c \times w_c)$ in $c \in \mathbb{R}^{h_i \times w_i \times (h_c \times w_c)}$. This procedure can be thought of as aligning each position of C to I. The resulting tensor of matches is summed across its last dimension to get the recognition scores $s \in \mathbb{R}^{h_i \times w_i}$, indicating how likely the location has a detection.

And lastly, the bounding box coordinates are obtained by taking the maximum and minimum of the grid tensor $g \in \mathbb{R}^{h_i \times w_i \times h_c \times w_c \times 2}$ along its third and fourth dimensions. This results in two points, the leftmost-topmost corner (x_{min}, y_{min}) and the rightmost-bottommost corner (x_{max}, y_{max}) of the bounding boxes.

3.5 Training

For training, the input images are divided into several smaller parts by randomly cropping an area to avoid downsampling. Otherwise, scaling them down to a fixed size would result in a loss of aspect ratio and the smallest objects would be represented by too few pixels. For augmentation, a small amount of scale jitter is applied to the cropped parts. For each image, a batch of class images is collected from those present and annotated in the image.

TransformNet specifically was trained only on positives, because when training it on negatives the network started to move the transformations in random directions. This can be explained in the following way:

When transformNet encounters a hard negative example (one that is difficult to distinguish from a positive one), it tries to align it.

What makes a hard negative example "hard" is that it is very similar to a positive example - they are hard to tell apart and share many similarities, like color, shape, text, etc., and only differ from positives slightly in some parts, not entirely.

The job of transformNet is to align the areas that are the same. Therefore, when it aligns a hard negative example, it is actually doing its job correctly by aligning the parts in C to parts in I that match. By getting penalized for this, it gets confused. That is why a copy of its output gets detached from the computational graph when training.

3.5.1 Loss function for localization scores

The localization loss used here is smooth L1 loss. It acts as L1 loss for values outside of [-1, 1]. If the absolute element-wise error falls below 1, it uses a squared term. For one object, the loss can be described as:

$$l_{loc}(y,\hat{y}) = \textstyle\sum_{c=1}^{4} \begin{cases} \frac{1}{2}(y_c - \hat{y_c})^2 & if\ |y_c - \hat{y_c}| < 1, \\ |y_c - \hat{y_c}| - \frac{1}{2} & otherwise. \end{cases}$$

where $y, \hat{y} \in \mathbb{R}^4$ are the predicted and ground truth boxes in the format of $x_{min}, y_{min}, x_{max}, y_{max}$. It is less sensitive to outliers than mean squared error and in some cases prevents exploding gradients[12].

3.5.2 Loss function for recognition scores

Object detection often faces the difficulty of a non-balanced number of positives and negatives. To overcome this, Ranked List Loss(RLL) by Wang et al.[24] is used. It is a margin-based loss for metric learning, which aims to learn the relative distances between inputs, instead of simply classifying them. It is defined as:

$$l_{rec}^{pos}(s) = max(m_{pos} - s, 0),\ \ l_{rec}^{neg}(s) = max(s - m_{neg}, 0),$$

$$L_{rec}^{RLL} = \textstyle\sum_{i:t_i=1} \frac{1}{\tilde{n}_{pos}} l_{rec}^{pos}(s_i) + \sum_{i:t_i=0} w_i^{neg} l_{rec}^{neg}(s_i),$$

$$w_i^{neg} \propto exp(T l_{rec}(s_i, 0))[l_{rec}^{neg}(s_i) > 0].$$

$\tilde{n}_{pos}$ is the number of non-trivial positives, positives such that $l_{pos}^{rec}(s_i) > 0$.

For negatives, the weights w_i^{neg} are normalized so that they sum to 1 over all the negatives for each image-class pair. T is a temperature parameter that indicates how peaky the weights are.

T is chosen in a way that puts 10^3 times larger emphasis on losses that are further away from the negative margin m_{neg} than those closest to the negative margin m_{neg}. This makes RLL work in a way, that would be similar to combining a classical margin-based loss function, like contrastive loss, with mining hard negative examples.

3.5.3 Total loss

The total loss is a combination of those two:

$$L_{total} = L_{rec}^{RLL} + L_{loc} * loc_{scale},$$

where loc_{scale} indicates the scaling factor for localization loss.

3.6 Datasets

The main dataset, GroZi-3.2k[25](grozi), comprises all kinds of groceries from Swiss supermarket shelves. In total, there are 680 images, 8 921 objects for 1 063 classes. There are two splits of this dataset. The first consists of 622 objects of 185 classes in 84 images, with classes that were ignored during training. For the second split, those same 84 images were used, but with 518 objects of classes that were seen during training. The first split is used as the main validation split and the second as the secondary validation split, named grozi-val-new and grozi-val-old respectively. Additional test sets were collected named dairy, paste-f, and paste-v, with the latter consisting of unusual rotations. The setting is similar to grozi - supermarket shelves with groceries as classes.

Other datasets include Instance-level visual object Retrieval and Recognition[26] (INSTRE). The dataset is made up of 28 543 images and 200 object classes. These classes are split into two: INSTRE-S1, representing 100 classes of objects found in the lab of the dataset creators, and INSTRE-S2, representing 100 classes of buildings, logos and common objects gathered online.

3.7 Performance

Overall, OS2D proved to work well, with their best model achieving a high mAP of 90.6 on the main validation subset grozi-val-new, beating the two-stage detector baselines and outperforming the work by Hsieh et al.[5]. It also works well in domains found in the INSTRE dataset, given additional rotations are applied to the class images during evaluation, since features extracted by CNNs are not rotation invariant and INSTRE contains such rotations.

3.7.1 The metric

Mean average precision(mAP) is a metric for evaluating the performance of object detection models. In this case, mAP at the IoU threshold of 0.5 is used. It incorporates the trade-off between precision and recall, which considers false positives and false negatives. Precision and recall are defined as:

$$Precision = \frac{TruePositives}{TruePositives+FalsePositives}$$

$$Recall = \frac{TruePositives}{TruePositives+FalseNegatives}$$

Precision and recall are needed to calculate the Average Precision (AP), by plotting precision against recall at different thresholds, where thresholds are the confidence scores sorted in descending order. The AP is then calculated as the area under the curve. AP is calculated for each class separately.

Mean average precision is the mean of the AP over all classes:

$$mAP = \frac{1}{n}\sum_{k=1}^{n} AP_k$$

3.8 Reproduction of experiments

Experiments were conducted by training models with various methods and evaluating them on different datasets. The results can be seen in Table 3.1 for models trained on grozi dataset and Table 3.2 for models trained on INSTRE dataset. The hard patch-mining that the first table refers to is a method that chooses hard negative and hard positive regions from the *input* images I for the model to train on, while the RLL loss mentioned previously does hard negative mining on *class* images, based on the predictions.[4]

The models were trained using the Stochastic Gradient Descent (SGD) optimizer for 200 000 iterations. The initial learning rate was set at 0.0001, with a weight decay of 0.0001 and a momentum of 0.9. The rate of learning slowed down by a factor of 10 after 100 000 and 150 000 iterations. As in the original study, the input was groups of 4 pictures, each cropped to a size of 600x600 pixels. Each group could at most contain 15 different class images.

There are three main differences between the V1 and V2 models:

1. V1 uses simplified affine transformations with P = 4, whereas V2 uses P = 6 with rotations.

[4] offline vs online, input image crops vs class images

	grozi-val-new	grozi-val-old	dairy	paste-f	paste-v
$V1$	88.5	89.3	71.7	48.8	61.4
$V2_{init}$	86.0	80.0	66.0	48.5	68.5
$V2_{w/o\ hard-patch}$	88.9	83.8	69.0	51.3	71.1
$V2_{with-hard-patch}$	90.3	84.9	71.7	54.5	73.9

Table 3.1: The results of different versions of OS2D trained on grozi dataset. Training with hard-patch mining is slow, but it does slightly improve the performance

	INSTRE-S1	INSTRE-S2
$V2_{initresnet50}$	71.9	64.5
$V2_{initresnet101}$	69.7	63.2
$V2_{w/o\ hard-patchresnet50}$	88.5	77.4
$V2_{w/o\ hard-patchresnet101}$	88.2	79.1
$V1_{resnet50}$	83.6	73.7
$V1_{resnet101}$	87.0	75.8

Table 3.2: Models' performance on the INSTRE dataset(also trained on it) initialized with a different feature extractor for each version. Resnet101 worked slightly better for the V1 model while having little to no effect on V2

2. TransformNet is initialized from scratch in V1, which incidentally allows for naturally training the transformation directions without having to invert them. Moreover, since V2 is not initialized from scratch, it is applicable without any training and produces decent results.

3. V1 is trained under full supervision while V2 is trained under weak supervision due to rotations.

The input images in the grozi dataset were fixed to a size of 1 280, paste-v and dairy to 3 500, and paste-f to 2 000 with additional class image rotations at 90, 180, and 270 degrees for the latter. The evaluation was done at 7 scales of the input image: [0.5, 0.625, 0.8, 1, 1.2, 1.4, 1.6]. Overall, the results were very close to the ones reported on all datasets.

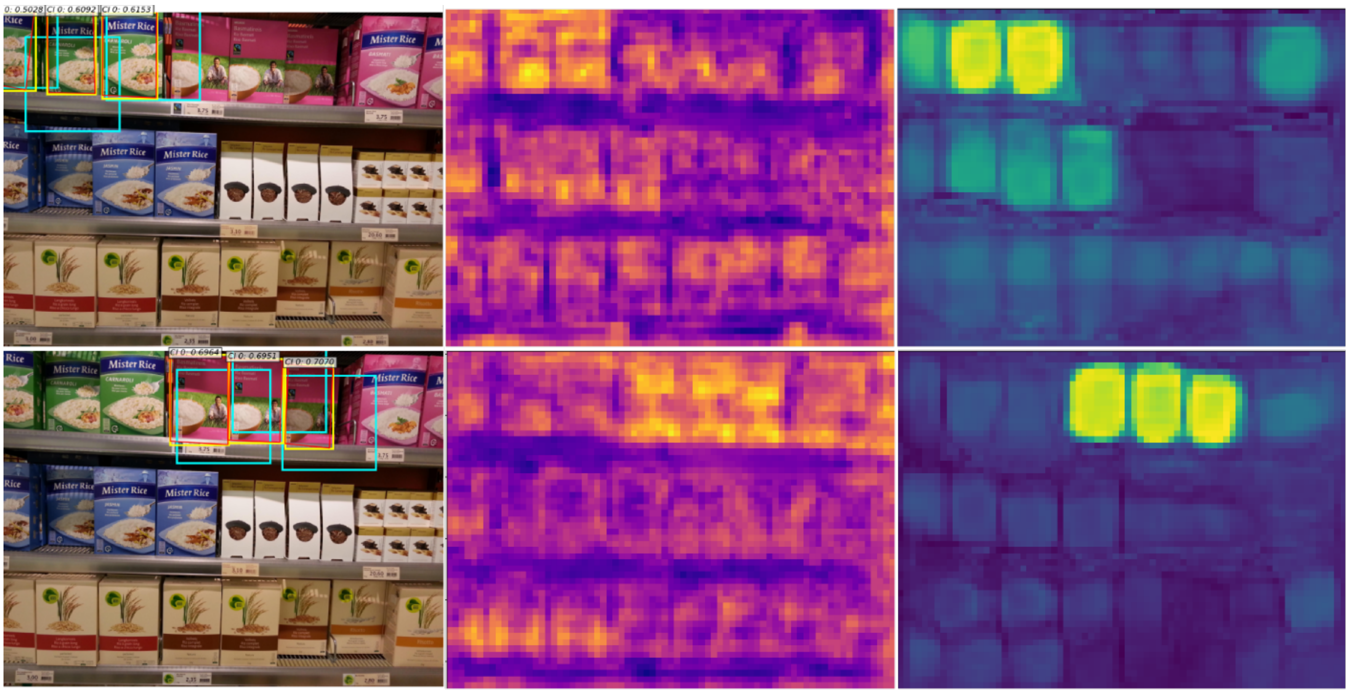


Figure 3.4: The images above portray a couple of simple examples from the training subset of grozi. On the left are the results with anchor boxes depicted in light blue, parallelograms constructed from the corners of transformations in red, and predicted boxes in yellow. Images in the center depict correlation tensor $c \in \mathbb{R}^{h_i \times w_i \times (h_c \times w_c)}$, pooled along the last dimension. The rightmost image is a heatmap of recognition scores, i.e., the results after resampling the correlation tensor.

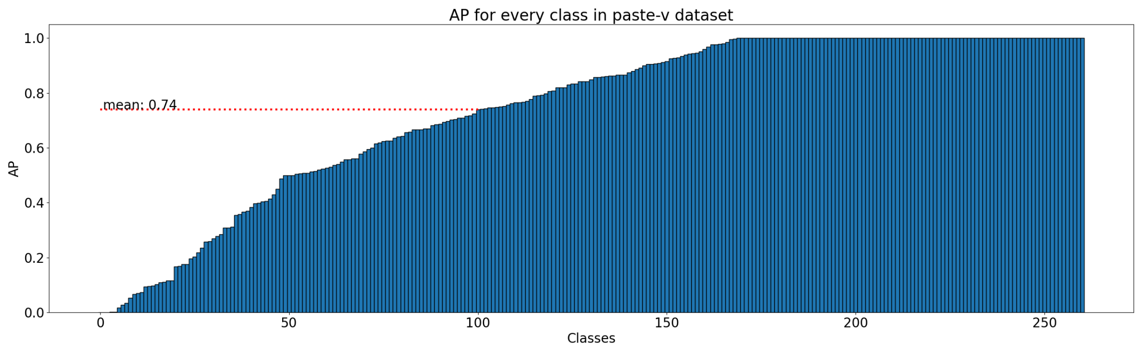


Figure 3.3: Average Precision per class in the paste-v dataset, evaluated on the best V2 model

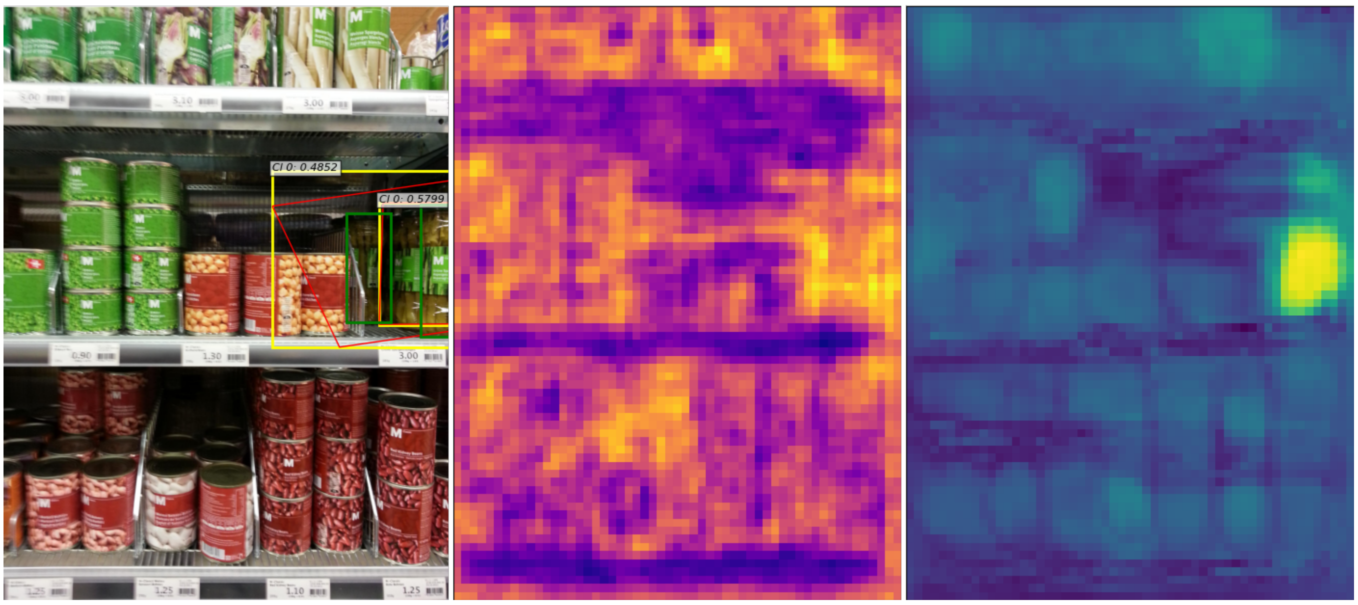


Figure 3.5: The same setting as previously, but with anchor boxes excluded, and instead, the annotated boxes are drawn in a dark green color. The glass jars of asparagus that stand in front of the others partially cover their labels, but not entirely, making it difficult to distinguish between them.

3.9 Conclusion

By observing the pictures and the tables, we can conclude that OS2D behaves as intended and excels when the objects of interest are clearly recognizable from their class images but not when they are rotated in 3D space or somehow overlapped. And differently from two-stage object detectors, it can recognize objects that consist of two distinct parts as one instead of splitting the detection as shown in a qualitative example in Fig 3.8.

Aside from the sensitivity to rotations, another issue noticed when conducting experiments, namely on dairy and paste datasets that are of higher resolution, is the drastic drop in inference time compared to grozi and INSTRE datasets. As the number of classes to detect increases and the input images get higher in resolution, OS2D becomes unfit for real-time detection[5].

[5]The detector is actively employed by mirum.io in Russia, the website suggests that detections occur in minutes

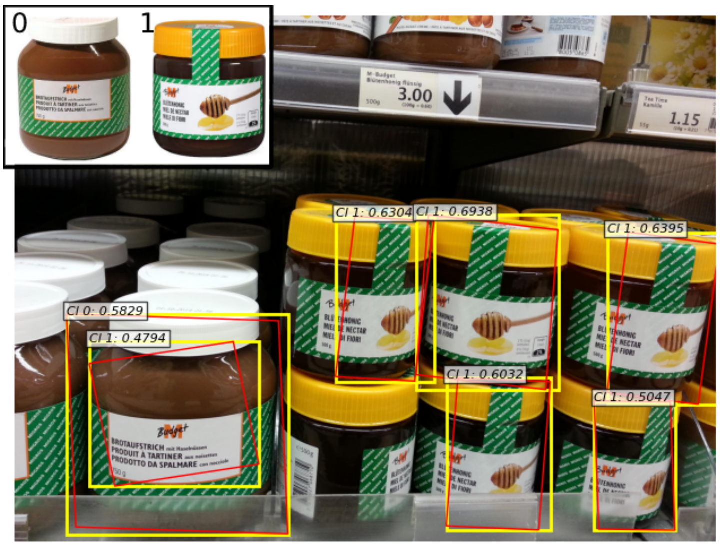


Figure 3.6: This example demonstrates a difficult case when objects are rotated in 3D space. Since the method matches the whole jar to the labels, the more the jars are rotated around their own axis, the more difficult it is to match them correctly.

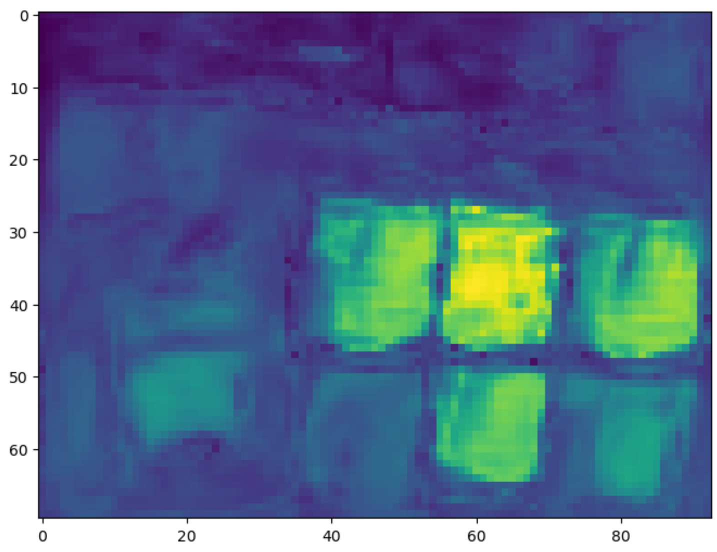


Figure 3.7: Recognition scores for class 1 shown in Fig. 3.6. Note how the highest scoring region corresponds to an object turned with its front facing forward, identical to its class image in the top-left corner.

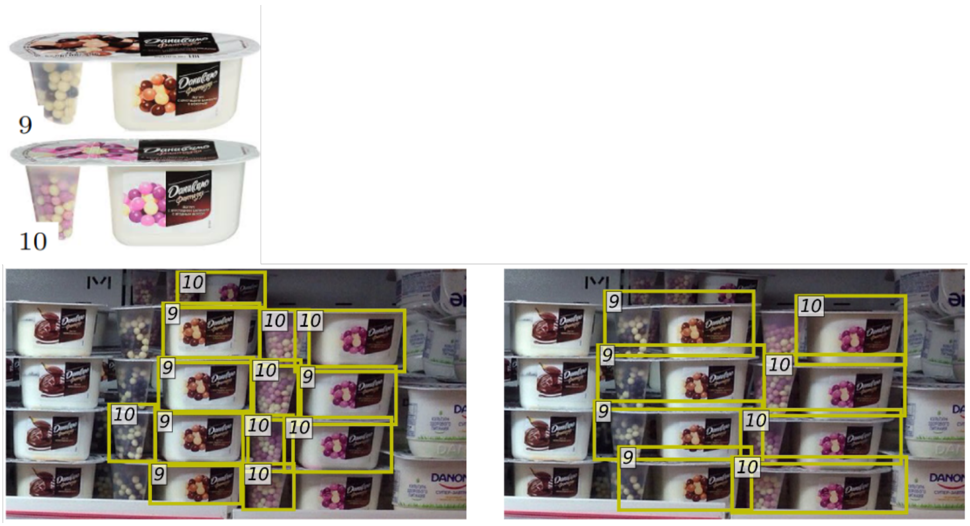


Figure 3.8: A qualitative comparison of OS2D model with a baseline similar to two-stage detectors(region detector + retrieval), for objects comprised of two parts and not seen during training. The baseline system has to finalize the object bounding box without knowing the classes of objects it is supposed to detect. Thus, if the object detector fails to detect two parts as one, the retrieval system cannot fix the boxes and recognize them correctly. On the other hand, the OS2D model is aware of the fact that the objects consist of two parts and detects them correctly. This image is borrowed from the paper[3]

Chapter 4
Model

Based on the available implementation of OS2D, we aim to remove the transformNet due to its computational and design complexity. To achieve this and maintain a one-stage detection framework, we propose the following changes:

- We borrow the idea of coordinate regression layers from one-stage detectors like YOLO and SSD described in Sect. 2.4. These layers directly predict bounding box coordinates from feature maps, omitting the need for transformNet.
- To address the problem of CNNs not being inherently scale-invariant, we apply a fixed number of rescalings to the class images whilst keeping their aspect ratio intact. The scales were chosen based on empirical observations of the dataset.
- In an attempt to make the network learn the specific features of a given class, we employ a simple method used in Target Driven Instance Detection (TDID) by Ammirato et al.[4].

4.1 Class image augmentation

After normalizing the images by subtracting the mean and dividing by the standard deviation, N number of rescalings are applied to the class images, approximately corresponding to the common sizes of objects found in the input images of the grozi dataset. By experimenting with different numbers of N, we stopped at 3 to not overload the memory and slow down the computations too much.

Then, ResNet50 extracts the weights from all the class and input images at its fourth residual block. The feature maps are then $I \in \mathbb{R}^{d \times h_i \times w_i}$ and $C = \{c_1, c_2, \ldots, c_N \mid c_j \in \mathbb{R}^{d \times h_{c_j} \times w_{c_j}}\cdot\}, \quad |C| = N$, for input and class respectively.

4.2 Feature comparison

Following Ammirato et al.[4] constructed a joint embedding that combines feature correlation and feature differencing between the class images and input images.

First, global max pooling is applied to all class feature maps C, yielding feature vectors with the most dominant features for every feature map across every channel, $V \in \mathbb{R}^{N\times d\times 1\times 1}$.

Next, each of these vectors are subtracted from the input images at every pixel to make an embedding $DIFF \in \mathbb{R}^{N\times d\times h_i\times w_i}$.

$$DIFF_j = I - v_j, \text{ for every } v_j \in V$$

Then, input feature maps I are multiplied by each one of the vectors in V as if applying convolutional filters to I. This results in a correlation embedding $CORR \in \mathbb{R}^{N\times d\times h_i\times w_i}$. Contrarily to computing cosine similarity in OS2D in Sect. 3.2, the feature maps are not normalized and the multiplication is done element-wise.

$$CORR_j = I * v_j, \text{ for every } v_j \in V$$

These embeddings were tested together and separately. In the end, simple feature differencing performed better on its own. So we omit the correlation tensor.

4.3 Multilayer Perceptron

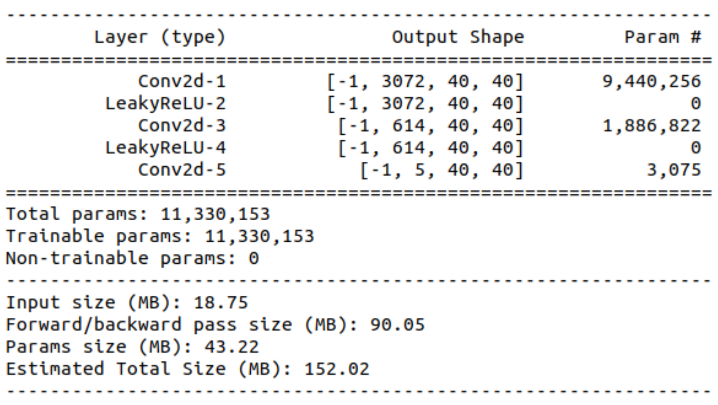
After obtaining the embedding, it is reshaped into the form $DIFF \in \mathbb{R}^{(N\times d)\times h_i\times w_i}$ and passed through a Multilayer Perceptron(MLP) in a fully-convolutional way using point-wise convolution.

In a point-wise (1x1) convolution, each output channel of a pixel is computed as a weighted sum of its input channels. If we consider each pixel as a separate instance and the channels as features of these instances, then a point-wise convolution is equivalent to applying a dense layer on the channels for each pixel.

Convolutions with a 1x1 kernel are compact, requiring much fewer parameters than convolutions with a bigger kernel[1], the spatial dimensions stay intact without any padding. They are often used in lightweight object detectors, such as MobileNetV2[27] and SqueezeNet[28].

In total, there are three layers of point-wise convolutions with leaky ReLU[29] activation functions between them. The output is then of size $5 \times h_i \times w_i$. These 5 parameters predict the box width, height, center location

[1]For instance, when a 3x3 kernel is applied to N input channels producing M channels, 3x3xNxM has 9 times more parameters than 1x1xNxM



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1        [-1, 3072, 40, 40]       9,440,256
         LeakyReLU-2        [-1, 3072, 40, 40]               0
            Conv2d-3         [-1, 614, 40, 40]       1,886,822
         LeakyReLU-4         [-1, 614, 40, 40]               0
            Conv2d-5           [-1, 5, 40, 40]           3,075
================================================================
Total params: 11,330,153
Trainable params: 11,330,153
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 18.75
Forward/backward pass size (MB): 90.05
Params size (MB): 43.22
Estimated Total Size (MB): 152.02
----------------------------------------------------------------
```

Figure 4.1: A summary of the MLP, given an input of size (1024 * 3, 40, 40)

on the x-axis, the center location on the y-axis, and the confidence scores. They are denoted as Δw, Δh, Δcx, Δcy and $\Delta conf$.

The predictions are then processed as follows:

$$\begin{aligned} box_w &= a_w e^{\Delta w} \\ box_h &= a_h e^{\Delta h} \\ box_{cx} &= ReLU(sigmoid(\Delta cx) - 0.5 + c_x) \\ box_{cy} &= ReLU(sigmoid(\Delta cy) - 0.5 + c_y) \\ box_{conf} &= sigmoid(\Delta conf) \end{aligned}$$

where c_x, c_y are the grid cell's center coordinates, and a_w, a_h are the anchor box's width, and height.

There are 5 predictions for every pixel in I. Since we do not use a softmax layer, the entire process is approached as a multi-label problem instead of a multi-class one. In the context of object detection, it means that each predicted bounding box can have multiple classes.

To convert the localization predictions to global coordinates, box_w, box_h, box_{cx}, and box_{cy} are multiplied by the receptive field of the feature extractor.

4.4 Loss

The loss functions used were available in the OS2D implementation. They are smooth L1 loss described in Sect. 3.5.1 for localization and contrastive loss[30] with hard-negative mining for confidence scores. Contrastive loss is another margin-based loss used for metric learning, defined like this:

$$l_{rec}^{pos}(s) = max(m_{pos} - s, 0), \;\; l_{rec}^{neg}(s) = max(s - m_{neg}, 0),$$

$$L_{rec}^{CL} = \frac{1}{n_{pos}} \sum_{i:t_i=1} l_{rec}^{pos}(s_i)^2 + \frac{1}{n_{pos}} \sum_{i:t_i=0} l_{rec}^{neg}(s_i)^2$$

During training, since some *instances* within a category (toothpastes of *different flavors*) may look quite similar, the classifier can easily overlook hard negative examples and output lots of false positive detections. The classical hard negative mining approach is to take those false positives, explicitly create negative examples out of them, and add those examples to the training set.

In our case, though, they are not added to the training set, but rather a fixed number of them is mined for each positive in a batch, and other negatives are discarded. This is essentially done to control the imbalance between negative and positive examples, which can lead to a model that is biased towards predicting negatives, as it might find the task of minimizing the error on the negative examples easier.

4.5 Other methods

The initial proposed method was different. Its main focus was making more transformations of the class images, such as scaling and rotation, to make the MLP learn the box positions and confidence scores from them. The correlation was computed differently too:

$$v_{gap} = GAP(\tfrac{C}{\|C\|_2}), \quad \in \mathbb{R}^{d \times N}$$
$$I_{norm} = \tfrac{I}{\|I\|_2}, \quad \in \mathbb{R}^{w_i \times h_i \times d}$$
$$cos_{sim} = I_{norm} \times v_{gap}, \quad \in \mathbb{R}^{w_i \times h_i \times N}$$

Where I stands for image feature maps, and v_{gap} stands for feature vectors of classes. v_{gap} is obtained by performing global average pooling on normalized class feature maps. Normalization is performed along the feature dimension d. Then, cos_{sim} would have been further passed into an MLP to get the same format of predictions, $w_i \times h_i \times 5$.

Unfortunately, this did not work. The reason why it did not work in my opinion is because of all the information getting lost when reducing the feature dimension to 1 for each transformation N. All it had was the intensities for each position of the image feature map I times N. A scenario where it might excel is instance segmentation, but not detection.

4.6 Performance

Performance-wise, the method was not so successful. Scoring 26.0 mAP on grozi-val-new and 44.7 mAP on grozi-val-old sets. The main issue was the lack of accuracy, that is to say, the number of false positive predictions outnumbered true positives.

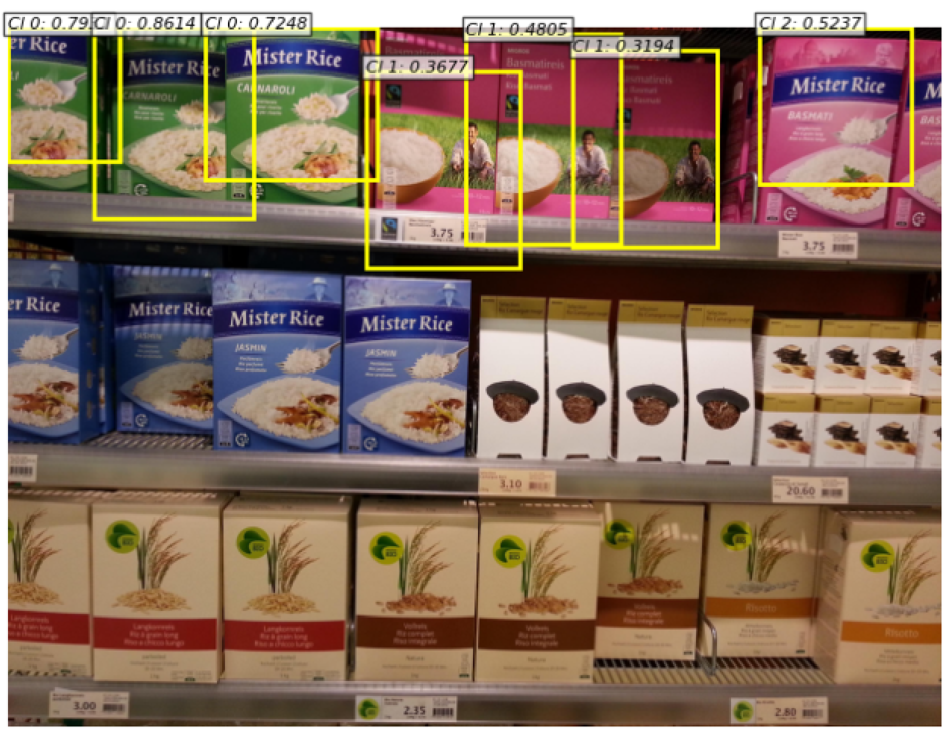


Figure 4.2: An example with rice boxes that were seen during the training. The class labels are correct, but the boxes are a little off and confidence scores are low for the 3 pink boxes in the middle.

An experiment with N = 1 (meaning the class images are rescaled to one fixed size) showed, that the number of rescalings matters as the performance dropped to 14.4 mAP on the grozi-val-new set. However, naively increasing N is not a solution, as it takes a toll on the inference time and memory consumption.

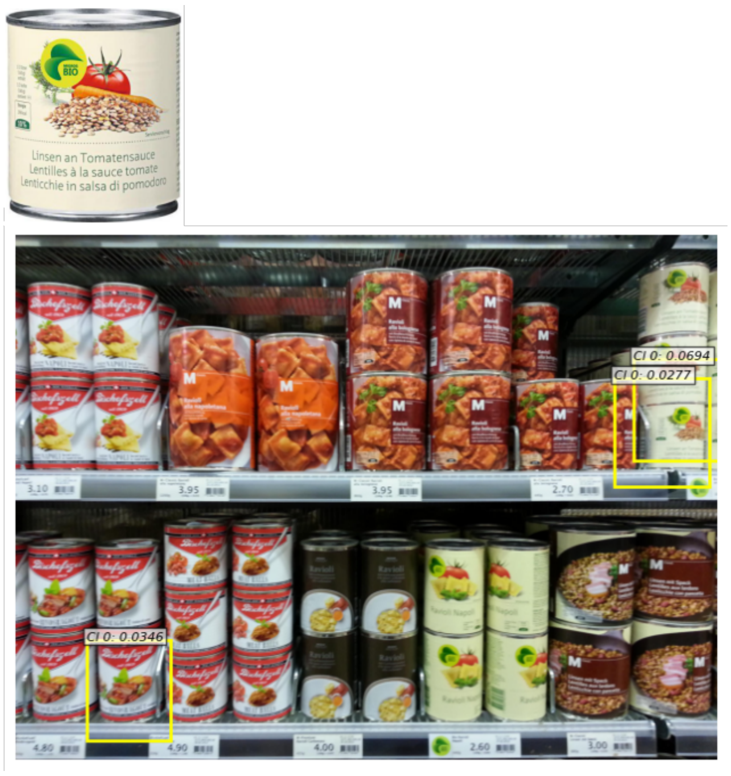


Figure 4.3: Detections for a class unseen in training. While the model has a general understanding of where the cans of lentils are located, the results are not quite accurate. The confidence scores are low too.

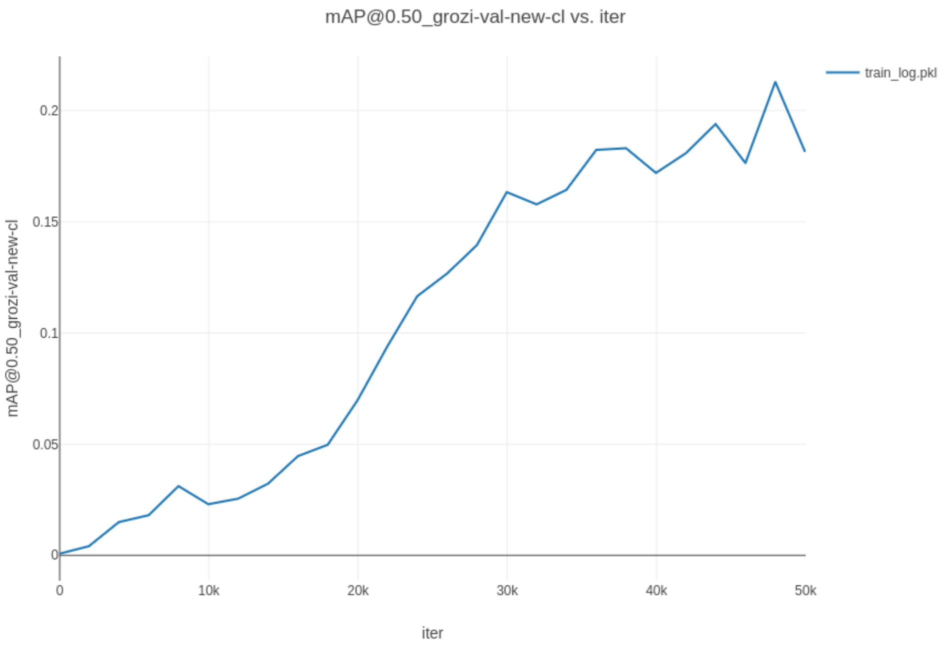


Figure 4.4: Mean average precision over iterations on grozi-val-new subset. After it enters the area between 20 to 25, it would oscillate in that area without any significant improvement.

Chapter 5

Conclusion and further improvements

A thorough study of the work by Osokin et al.[3] was conducted, and their experiments on the instance-level datasets were successfully replicated. The method proposed by Ammirato et al.[4] was adopted to deal with the same task, and we observed that this method resulted in a decrease in performance when compared to OS2D. However, there is room for further improvement, with several possible paths:

The first is to minimize the size of the network, as it is overparametrized at this stage. Alas, this was noticed too late, and no further experiments were conducted. Reducing the number of parameters would allow for more class image augmentations, and with that, the MLP might be able to "choose" the appropriate transformation. This has the potential to work, as reducing the number of rescalings correlates negatively with performance.

The second is to address the problem of low accuracy. The loss function emphasizes hard negatives. In spite of that, the model still learned mostly from easy examples, resulting in high recall and low accuracy. This is likely due to the suboptimal ratio of positives to negatives, where there were too many negatives. The reason for such a ratio is that when we tried reducing the number of negatives per batch, the model could not learn from those few negatives.

Last but not least is to follow the progress of regular one-stage object detectors. That is, to predict not one but many anchor boxes of varying proportions and do detections at multiple grid levels by utilizing several blocks of the feature extractor at once. This will further aid the MLP in handling differently shaped objects.

Bibliography

[1] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement”, *arXiv preprint arXiv:1804.02767*, 2018.

[2] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, “Microsoft coco: Common objects in context”, in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, pp. 740–755.

[3] A. Osokin, D. Sumin, and V. Lomakin, “Os2d: One-stage one-shot object detection by matching anchor features”, in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV 16*, Springer, 2020, pp. 635–652.

[4] P. Ammirato, C.-Y. Fu, M. Shvets, J. Kosecka, and A. C. Berg, “Target driven instance detection”, *arXiv preprint arXiv:1803.04610*, 2018.

[5] T.-I. Hsieh, Y.-C. Lo, H.-T. Chen, and T.-L. Liu, “One-shot object detection with co-attention and co-excitation”, *Advances in neural information processing systems*, vol. 32, 2019.

[6] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge”, *International journal of computer vision*, vol. 88, pp. 303–338, 2010.

[7] D. Dwibedi, I. Misra, and M. Hebert, “Cut, paste and learn: Surprisingly easy synthesis for instance detection”, in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1301–1310.

[8] E. S. Olivas, J. D. M. Guerrero, M. Martinez-Sober, J. R. Magdalena-Benedito, L. Serrano, *et al.*, “Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques: Algorithms, methods, and techniques”, in IGI global, 2009, ch. 11, pp. 242–265.

[9] F. A. Faria, L. H. Buris, F. A. Cappabianco, and L. A. Pereira, “Combining deep metric learning approaches for aerial scene classification”, *arXiv preprint arXiv:2303.11389*, 2023.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[11] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model", in *2008 IEEE conference on computer vision and pattern recognition*, Ieee, 2008, pp. 1–8.

[12] R. Girshick, "Fast r-cnn", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", *Advances in neural information processing systems*, vol. 28, 2015.

[14] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn", in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[16] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 7464–7475.

[17] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector", in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, Springer, 2016, pp. 21–37.

[18] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection", in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[19] I. Rocco, R. Arandjelovic, and J. Sivic, "Convolutional neural network architecture for geometric matching", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6148–6157.

[20] I. Rocco, R. Arandjelović, and J. Sivic, "End-to-end weakly-supervised semantic alignment", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6917–6925.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[23] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, “Spatial transformer networks”, *Advances in neural information processing systems*, vol. 28, 2015.

[24] X. Wang, Y. Hua, E. Kodirov, G. Hu, R. Garnier, and N. M. Robertson, “Ranked list loss for deep metric learning”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5207–5216.

[25] M. George and C. Floerkemeier, “Recognizing products: A per-exemplar multi-label image classification approach”, in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II 13*, Springer, 2014, pp. 440–455.

[26] S. Wang and S. Jiang, “Instre: A new benchmark for instance-level object retrieval and recognition”, *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 11, no. 3, pp. 1–21, 2015.

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[28] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size”, *arXiv preprint arXiv:1602.07360*, 2016.

[29] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, “Rectifier nonlinearities improve neural network acoustic models”, in *Proc. icml*, Atlanta, GA, vol. 30, 2013, p. 3.

[30] P. Khosla, P. Teterwak, C. Wang, *et al.*, “Supervised contrastive learning”, *Advances in neural information processing systems*, vol. 33, pp. 18 661–18 673, 2020.

Appendix A
The use of AI

List of AI tools used:

- Microsoft Copilot for machine translation, literature search, and consulting problems. Outputs were double-checked.
- QuillBot for grammar and flow.
- Grammarly for grammar and flow.

Appendix B
Contents of attachment

```
/
├── data
├── os2d
├── demo.ipynb
├── readme.txt
└── INSTALL.md
```

Appendix C

Glossary

grozi GroZi-3.2k dataset comprised of groceries on supermarket shelves. 16

pixel A pixel is referred to as an element of a feature map, not necessarily an image. 13

TransformNet Spatial transformer network used in OS2D. 11