ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE

AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Machine Learning for Computer Vision

CONVOLUTIONAL NEURAL NETWORK ARCHITECTURES FOR TEMPLATE MATCHING

CANDIDATE
Federico Spurio

SUPERVISOR
Prof. Samuele Salti

CO-SUPERVISORS
Dott. Angelo Carraggi
Dott. Maurizio De Girolami

Academic year 2021-2022

Session 2nd

Contents

List of Figures

List of Tables

Introduction

Computer Vision is the ensemble of algorithms and techniques that allows computers to reproduce functions and processes of the human visual apparatus. The aim of this branch of research is to extract information from images at increasing higher levels of abstraction and understanding. These information allows Computer Vision algorithms to solve many tasks, for example to detect objects, people, animals inside photos and videos, to recognize faces, emotions and more.

In recent years, the advent of increasingly advanced Machine Learning techniques have increased the interest in Computer Vision, since they have allowed to achieve performance comparable to human standards.

The objective of this thesis is to analyze existing algorithms and propose new methods for solving the task of Template Matching. The Template Matching problem is defined as follows: given two images, one called *template* and the other *query*, the objective is to find the ”object” (could be a person, an animal, an everyday object) represented in the template image inside the query image (that usually contains other objects in addition to the searched one).

Template Matching has countless applications in different fields: in the manufacturing (as part of the quality control), a way to navigate a mobile robot, for solving games (like ”Where’s Waldo”, cited in the following chapters) and many more.

Machine Learning and Deep Learning have the spotlight on by the industry and in the research field, even by public opinion, because they have allowed to solve task deemed impossible for computers. Even thinking about the hot

topic of autonomous driving.

For this reason, this work focuses on overcoming the limitations of classical Computer Vision techniques (SIFT, similarity scores,...) for solving Template Matching using Neural Networks. The main challenges in Template Matching are scale changes, rotation, different light intensities, affine and non-affine transformation of the template in the query image.

This thesis is the result of my curricular internship done in Datalogic S.p.A. [1], an Italian company that operates worldwide in the fields of automatic data acquisition and process automation [2].

From the point of view of the structure, this thesis is divided in five chapters. The first chapter is an overview (advantages and limitations) of the classical algorithms of Computer Vision for solving the Template Matching task. Chapter two presents existing Neural Networks specifically designed for solving Template Matching (WaldoNet), and Neural Networks partially or not initially designed for this specific task (PatchNet and FlowNet).

Chapter three is rather technical, where all changes and tests I have made to the mentioned networks to make them suitable for the task of Template Matching and to achieve better results are explained. Then, in Chapter four, the choices that I made for the training (dataset, loss function, metrics for evaluation) are described. In the last chapter, the results on virtual experiments done on the custom networks, and the evaluation (with the chosen metrics) of both classical algorithms and deep learning methods are reported.

Chapter 1

Classic computer vision for Template Matching

Template Matching is a well-known problem in the literature of classical Computer Vision. For many years researchers and companies rely on classical algorithm to solve this problem. Since it is not a new challenge, the methods described in this chapter have been studied for years. There exists both published algorithms, taught also in University courses (e.g. ZNCC, SIFT, ...) and patented or not-disclosed algorithms (e.g. property of companies, like Datalogic's Blueye).

In this section, a brief explanation of disclosed algorithms is given.

1.1 (Dis)similarity functions

In an ideal setting where the template is present in the image with the same size and without any rotation, Template Matching could be solved by simply slid the template image across the query image and compared at each position (as shown in Figure 1.1). Then a (dis)similarity function is computed. The objective is to find the maximum similarity score, or the minimum dissimilarity score.

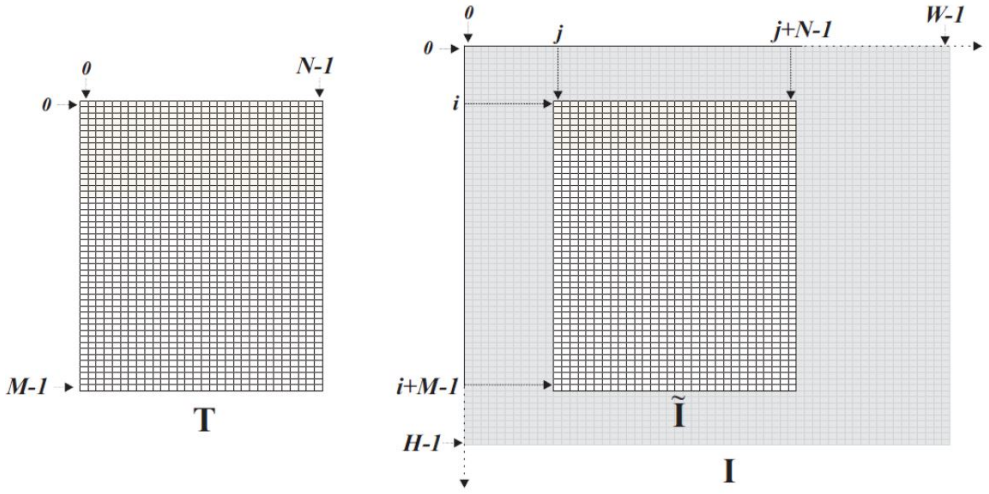


Figure 1.1: Example of sliding window of same size of the template across the whole query image

1.1.1 Compare intensities

One way to solve this problem is to compare intensities of the template and the query. This could be done via **Sum of Squared Differences (SSD)** or **Sum of Absolute Differences (SAD)**. For definition these methods are not invariant to intensity changes. If there is no changes in the light from template to query then these methods are fast and efficient.

1.1.2 (Zero-mean) Normalized Cross-Correlation (Z)NCC

The **Normalized Cross-Correlation** (NCC) is a way to handle linear intensity changes, but it turns out to be computationally slow. A more robust version of the NCC is the **Zero-mean NCC** (denoted as ZNCC), which is invariant to affine intensity changes, thanks to the subtraction of the local mean [3]. It is slightly more computationally expensive with respect to the NCC, because it needs to compute the mean of the template and the mean of the sliding window, but can be speed up by using box-filtering.

Starting from a query image I of sizes $H \times W$ a sub-template image T of size $M \times N$ and , for each position (i, j), the similarity score between the

template image T and a query image I can be computed through the (Zero-mean) Normalized Cross-Correlation as in the following equation 1.1:

$$\text{ZNCC}(i,j) = \tag{1.1}$$

$$= \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(I(i+m,j+n)-\mu(\tilde{I}(i,j)))\cdot(T(m,n)-\mu(T))}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(I(i+m,j+n)-\mu(\tilde{I}(i,j)))^2}\cdot\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(T(m,n)-\mu(T))^2}}$$

where:

$$\mu(\tilde{I}(i,j)) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m,j+n)$$
$$\mu(T) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} T(m,n)$$

The sums over the integers $m < M$ and $n < N$ in the above equations represent the operation of the *sliding window* of the image I and template T matrices. After the sliding window summation, with $\tilde{I}(i,j)$, it is denoted the subset of the image corresponding to the template at position (i,j). Finally, the sliding window output is normalized by the size $M \times N$ and this is symbolically represented by the mean operator $\mu(\cdot)$.

ZNCC-based Template Matching finds the same optimal solution as a full-search process and allows for significant computational savings.

1.1.3 Fast Template Matching

All the methods above described can be very slow. As a matter of fact, their complexity is $\mathcal{O}(M \cdot N \cdot W \cdot H)$, where $H \times W$ is the dimension of the query image, $M \times N$ is the dimension of the template.

Template Matching can be solved by ***approximate methods*** or ***exact methods***.

Among the approximate methods there is the *Image Pyramid*, in which the idea is to shrink the image, building an image pyramid. It is a very fast approach, but the number of level of the pyramid needs to be chosen carefully, to avoid losing too much information.

Instead, as said before, NCC and ZNCC are exact methods. They can be made faster by carrying out the Template Matching in the Fourier domain (due to the Convolution Theorem [4]) and thanks to box-filtering. Box-filtering is an incremental calculation scheme which makes the calculation independent of the template area and requires only four elementary operations per image position. [5]. With these two methods the computational complexity boils down to $\mathcal{O}(W \cdot H \cdot log_2(W \cdot H))$.

1.2 Scale Invariant Feature Transform (SIFT)

The **Scale Invariant Feature Transform (SIFT)** is a computer vision algorithm to detect, describe and match local features in images [6].
In Template Matching, SIFT could be used to match the feature of the template image with the feature of the query image. This task boils down to a classical Nearest Neighbour (NN) Search problem. The SIFT descriptors are matched computing a distance function, usually the Euclidean distance. The matches then need to be validated, to avoid false matches and matches when the template is not present.
The algorithm as above described is quite slow, but there exists techniques to speed it up (e.g. indexing techniques).
Establishing correspondences between features of the template and the features of the query image is done in three steps:

- **Detection** of keypoints
- **Description** - computation of suitable descriptor based on a neighbourhood around a keypoint

- **Matching** descriptors between images

As said before, this process should also be invariant to many transformation.

1.2.1 Detection

A good detector should find the same keypoints in different views of the same scene, despite transformations (property of repeatability). It should also find keypoints surrounded by informative patterns (property of saliency). One way could be to use the corner detectors like Harris. However, corners are rotation-invariant, but not scale-invariant.
To obtain scale-invariant features, the detector should work on the scale-space. The key of the scale-space is to apply the detector on scaled and increasingly blurred version of the input image. The most used scale-space filter is the Laplacian of Gaussian (LoG), changing the value of the standard deviation σ. This means, for instance, that Gaussian kernel with low σ gives high value for small blobs, while Gaussian kernel with high σ fits well for larger blobs. So, the local maxima across scale and space gives a list of (x, y, σ) values, that are the possible candidates for a keypoint at (x, y) at σ scale.
The computation of LoG is costly. Instead, SIFT uses the **Difference of Gaussian (DoG)**, that provides a computationally efficient approximation of LoG. To localize keypoints more accurately, DoG function is approximated around each extrema by its second degree Taylor expansion.

1.2.2 SIFT Descriptor

The SIFT descriptor is computed in a 16×16 oriented pixel grid around each keypoint (Figure 1.2 shows an example of keypoint descriptor). This is further divided in 16 sub-blocks of 4×4 size. For each region, a gradient orientation histogram with 8 bins is created. Gradients are rotated according to the canonical orientation of the keypoint. Each pixel in the region contributes to its designated bin according to gradient magnitude as well as to a Gaussian

weighting function centred at the keypoint. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation, etc...

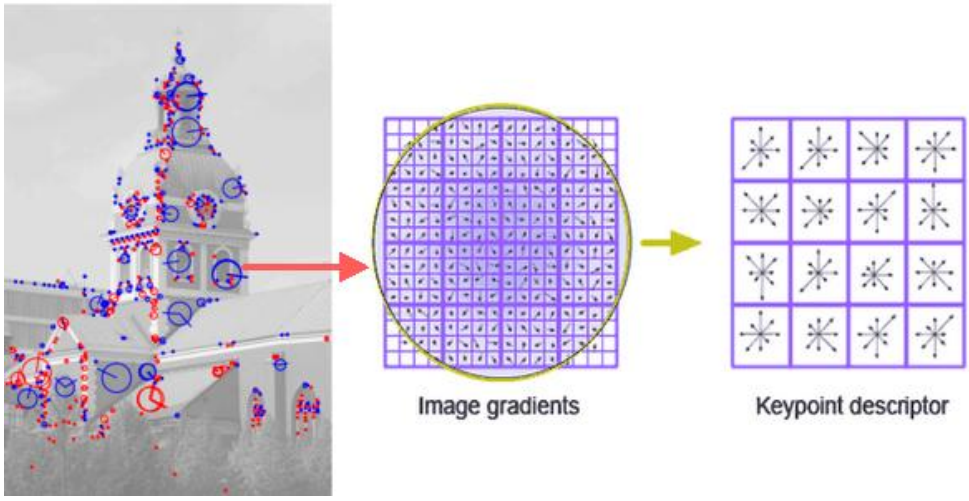


Figure 1.2: An example of SIFT keypoint descriptor, from [7]

1.2.3 Matching

The descriptors are compared across two images to find corresponding keypoints. This is usually done by **Nearest Neighbour (NN) Search**. The NN, because of noise or some other reasons, could be a non-valid correspondence. To avoid false matches, two criteria are used:

1. $d_{NN} \leq T$ (the NN distance is lower than a threshold T)
2. $\frac{d_{NN}}{d_{2-NN}} \leq T$ (the ratio of first-closest distance over second-closest distance is lower than a threshold T)

With $T = 0.8$, it is shown that 90% of wrong matches are rejected, while only 5% of correct matches are missed.

1.3 Implementations

Implementations of Template Matching and SIFT algorithm can be found, for Python and C++ programming languages, in the OpenCV library (open

source).

Template Matching is also an hot-topic in industries, and some of them have their own software to achieve this task. An example is the Datalogic software, **Blueye**. Blueye is a Template Matching algorithm designed to be fast for system with low computational power, while trying to keep an high and repeatable accuracy. Furthermore, it is designed to be robust to noises, in addition to small deformation of the query image, scale variation and/or rotations, and even to different polarity of the template to search.

1.4 Limitations

The SIFT algorithm was successfully applied in many computer vision tasks: object recognition, panoramic image stitching, gesture recognition, video tracking, identification of wildlife, etc. The drawback of this wide flexibility it is mathematically complicated and computationally heavy. One important feature of SIFT is that it derives descriptors which are scale and rotation invariant. The drawback is that these descriptors are created only for "interesting" local regions. Another disadvantage is that a large number of values to represent the image by the descriptors is needed. In general, the SIFT algorithm was found problematic when using for sub-image searching and for affine [1] transformations with rotation angles larger than $\sim 50^\circ$ [6]. The algorithm works with keypoints representations of images which are not convenient for near-homogeneous images.

The ZNCC is invariant to constant brightness changes, but it is not defined for constant intensity images, and shows close to one correlation between approximately white and black images. In addition, results of ZNCC are not rotation, scale and affine transformation invariant.

[1] In Euclidean geometry, an affine transformation is a geometric transformation that preserves lines and parallelism, but not necessarily distances and angles.

Chapter 2

Deep Learning Template Matching

More and more often, machine learning and deep learning are used to solve classical problems that rely on classical algorithms, and sometimes they overcome limitation of these methods. This is also the case of Computer Vision, where, for example, Object Detection is solved more effectively by deep learning approaches than by traditional algorithms.

Despite this, machine learning techniques are quite young and constantly updated. It is only recently that machine learning solutions have been implemented to solve the Template Matching task.

In this thesis I explore two of the solutions proposed explicitly for Template Matching that are *"Where's Waldo?*[1] *A Deep Learning approach to Template Matching"* [8] (from now on referred as *Hossler paper*), that makes use of Hypernetwork to carry out the spatial information of the template to the query image (from now on referred as **WaldoNet**), and *"PatchNet - Short-range Template Matching for Efficient Video Processing"* [10] (from now on referred as *Mao et al. paper*), that exploits the division in patch of the template, for a more efficient matching in the query (from now on referred as **PatchNet**).

[1] Also known as *"Where's Wally"*. It is a British series of children's puzzle books created by English illustrator Martin Handford [9]

Then a third method is investigated, not usually used for Template Matching, but for flow estimation. The method in question is **FlowNet**, from *"FlowNet: Learning Optical Flow with Convolutional Networks"* [11] (from now on referred as *Fischer et al. paper*), that has a correlation layer used for matching features of different frames to estimate the flow, but that could be converted and use for matching features of template and query image.

2.1 Convolutional methods

I decided to group the WaldoNet and PatchNet under the section **Convolutional methods** because, differently from FlowNet, these methods use standard convolutional layer to compare the features extracted from the query and from the template.
Nevertheless, these networks use the convolution differently, as shown in the following sections.

2.1.1 WaldoNet

WaldoNet is a Convolutional Neural Network (CNN), that given in input two images, the template image of size 224×224 pixels and the query image of size 326×224 pixels, produce in output a binary image of size 326×224 pixels (same as the input) with value 1 where the template is detected in the query image and 0 elsewhere.

The network is structured as follows (in Figure 2.1 is shown a summary image): there are two CNNs, trained simultaneously, one extracts the features from the template (Hypernetwork) and one extracts the features from the query image (Main network). The feature extraction is done with notorious CNNs and, in the case of the Hossler paper [8], VGG11 is used for both template and query. VGG (Visual Geometry Group, from the name of the department where it was developed) is a Neural Network whose purpose is to classify objects in a image. The number 11 indicates that there are eleven layers with

learnable parameters (8 convolutional layers and 3 fully connected layers) in the network.

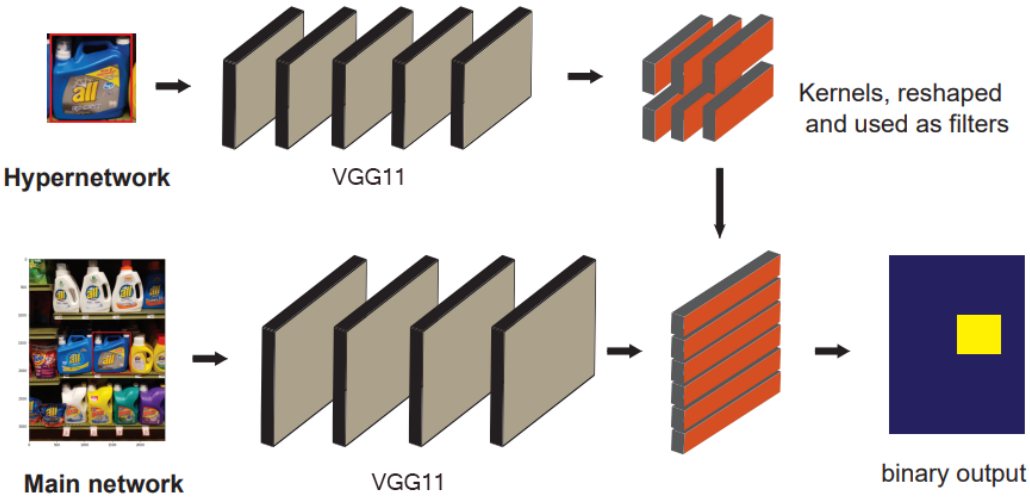


Figure 2.1: Architecture of WaldoNet from [8]

Hypernetwork and hyperlayer

The feature map produced by the Hypernetwork is used as weights of the last convolutional layer of the main network. In this way, the two feature maps are convoluted one with the other.

This operation is done in the following way: the last convolutional layer store its parameters into a kernel. The kernel K contains $N_{in} \times N_{out}$ filters, and each filter has dimensions $f_{size} \times f_{size}$. In this way, the kernel K storing all these parameters belongs to a space of dimensions $\mathbb{R}^{N_{in} \times N_{out} \times f_{size} \times f_{size}}$. The matrix K can be broken down as N_{in} slices of smaller matrices with dimensions $N_{out} \times f_{size} \times f_{size}$. Each slice of the kernel is denoted as $K_i \in \mathbb{R}^{N_{out} \times f_{size} \times f_{size}}$.

In the following of this document, the layer that turns the tensor produced by Hypernetwork into the kernel K is denoted as *hyperlayer*. Therefore, in the approach of this thesis, the hyperlayer is a two-layer linear network. [2]

[2]The Hossler paper [8] have called Hypernetwork all the layers that convert the tensor image into the kernel of the last convolutional layer of the Main network. The Hypernetwork, in the original paper of Ha et al. [12], corresponds to the two layers that I have called hyperlayer.

The template image is processed by VGG11 and produce a tensor $z \in N_{in} \times w' \times h'$, where $w' \times h'$ is the resolution of the image after being processed. Then, the tensor z is linearly projected into N_{in} inputs, and then flattened, so to have $z_i \in \mathbb{R}^{N_z}$, where N_z is just $w' \cdot h'$.

A complete overview of the process that brings the input vector z to the kernel K of the convolutional layer (notice that the operations described in the following are computed for all z_i, so computed N_{in} times) can be seen in the following three equations:

$$a_i = W_i z_i + B_i \qquad \forall i = 1, ..., N_{in} \tag{2.1}$$

$$K_i = \langle W_{out}, a_i \rangle + B_{out} \qquad \forall i = 1, ..., N_{in} \tag{2.2}$$

$$K = (K_1 K_2 \cdots K_i \cdots K_{N_{in}}) \tag{2.3}$$

Equation 2.1 refers to the the first layer of the hyperlayer that is applied to z_i with weights $W_i \in \mathbb{R}^{d \times N_z}$ and bias vector $B_i \in \mathbb{R}^d$, where d is the size of the hidden layer, and thus is an hyperparameter. A vector $a \in \mathbb{R}^d$ is produced.

Equation 2.2 refers to the second and final layer of hyperlayer. It corresponds to a linear operation which takes the input vector a_i of size d and linearly projects this into K_i, using a common tensor $W_{out} \in \mathbb{R}^{N_{out} \times f_{size} \times f_{size} \times d}$ and bias matrix $B_{out} \in \mathbb{R}^{N_{out} \times f_{size} \times f_{size}}$. The symbol $\langle W_{out}, a_i \rangle$ is the tensor dot product between $W_{out} \in \mathbb{R}^{N_{out} \times f_{size} \times f_{size} \times d}$ and $a_i \in \mathbb{R}^d$. It result is $\langle W, a \rangle \in \mathbb{R}^{N_{out} \times f_{size} \times f_{size}}$.

Finally, in equation 2.3 all the matrices K_i are concatenated to obtain the kernel K.

2.1.2 PatchNet

In the original Mao et al. paper, PatchNet is proposed to match objects in adjacent video frames, to solve both video object detection and visual object tracking tasks. Figure 2.2 represents a comparison between Correlation filter methods (that include also the WaldoNet), Siamese Networks (the idea at the

base of FlowNet) and their PatchNet architecture.

The architecture of PatchNet (shown in detail in Figure 2.3) is composed by two subnets: patch correlation layer and aggregation subnet.

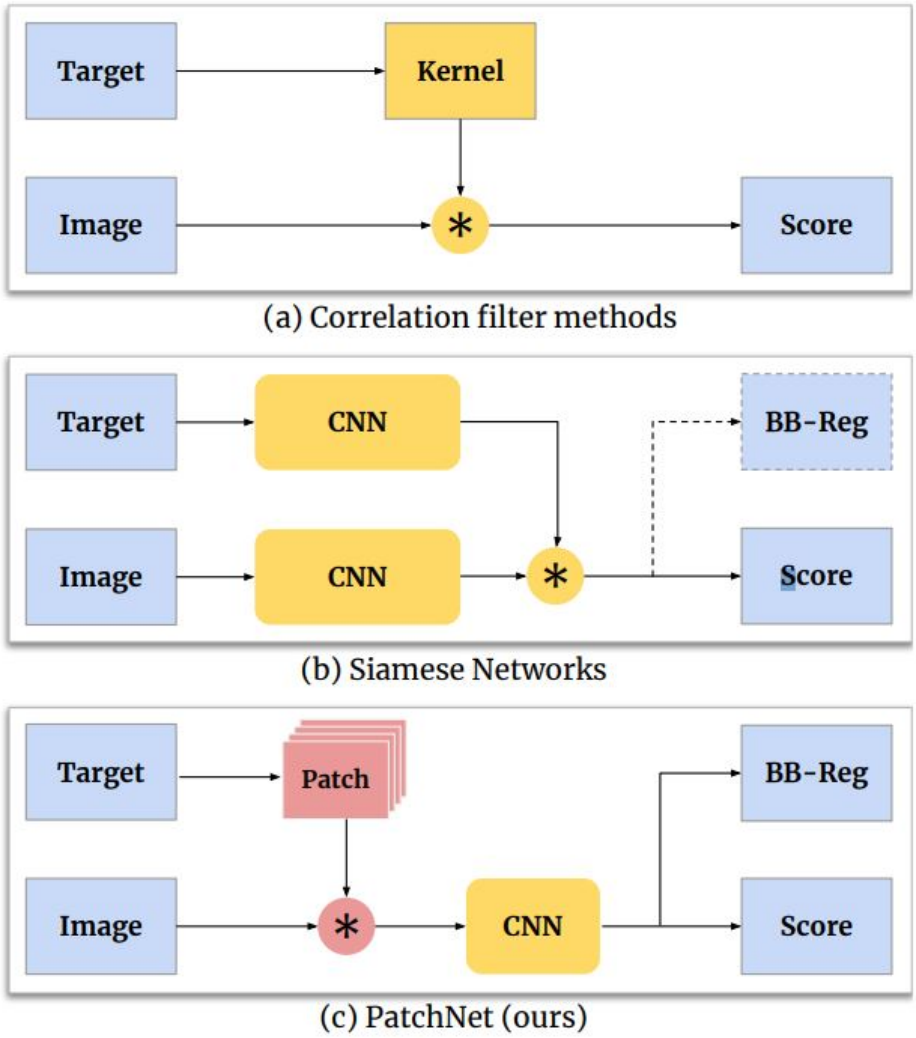


Figure 2.2: Comparison between PatchNet and other known architecture from [10]

Patch Correlation Layer

Given a fixed-size template image, it is split into a fixed number of patches. Each patch serves as convolutional filter, and concatenated together, they form a $4D$ tensor as convolutional weight. Then a $3D$ correlation map is produced by a standard $2D$ convolution, that perform patchwise correlation. This $3D$

map is treated as feature map and fed into the aggregation subnet, to localize the object center and regress the bounding box.
To avoid false responds to background, the authors propose a learning-based method: PatchNet selects weighted Fourier-domain features rather than plain image features. The Fourier coefficients are learned.

Aggregation Subnet

The Aggregation Subnet performs target localization and hierarchical bounding-box regression. The bounding box estimation is done in a few layers, and it is based on the observation that patch distribution information can inform bounding box estimation. Then, this information is hierarchically aggregated from smaller patches to larger patches, by convolution and modified max pooling.

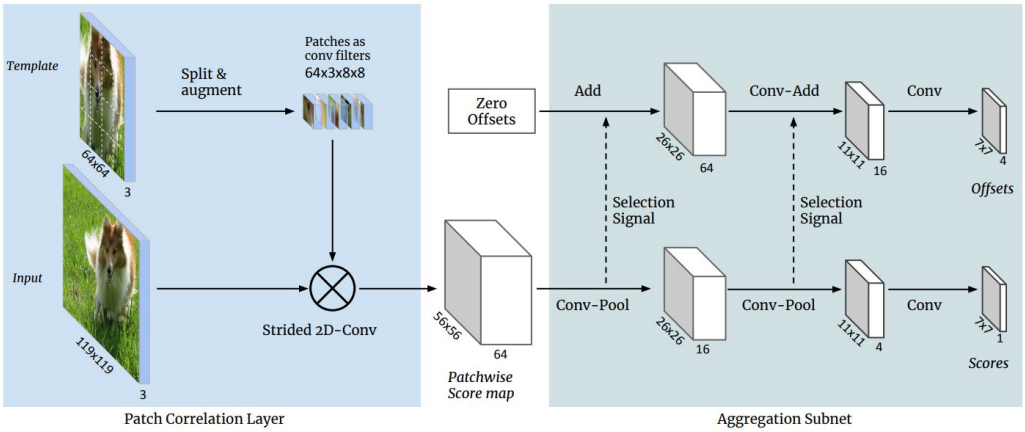


Figure 2.3: Architecture of PatchNet in details. It is composed by two subnets: Patch Correlation Layer and Aggregation Subnet from [10]

2.2 Optical Flow method

The network FlowNet was designed to solve the Optical Flow estimation task. The definition of this problem is estimating the apparent motion of all pixels between two frames of a video. Therefore, a network for solving this problem should take in input two images (in this case two frames) and produce in output

the flow map.

2.2.1 FlowNet

The architecture of FlowNet is similar to U-Net, since it has an encoder part and a decoder part, to gently upscale the tensor produced by the encoder. It takes in input two images of the same size and produces the flow map, that has two channels.

Encoder

The encoder part of FlowNet (shown in Figure 2.4) is divided in the Siamese feature extractor, the Correlation layer and a further processing. The Siamese feature extractor takes the two input images and extracts their features with three convolutional layers with shared weights. Then the features extracted from both image one and two are mixed together through the Correlation layer. This layer mimic the computation of matching costs performed by traditional flow pipelines. The matching cost at location (u, v) in the feature map f^1 (obtained from image x^1) for displacement (du, dv) in the feature map f^2 (obtained from image x^2) is the equation 2.4.

$$c(u, v, du, dv) = \sum_k f_k^1(u, v) f_k^2(u + du, v + dv) \tag{2.4}$$

That is the dot product between f^1 and f^2 (so the unnormalized NCC). This layer has no learnable parameters, and the only hyperparameters are the number of displacement $D = 2\lfloor d/s \rfloor + 1$, where d is the radius of the probing window, and s is the stride used to sample it (applied to prevent excessive computational cost).

The output has dimension $D \times D \times H \times W$, but it is reshaped into a $3D$ tensor to be further processed with $2D$ convolutions. So the final dimension is $D^2 \times H \times W$.

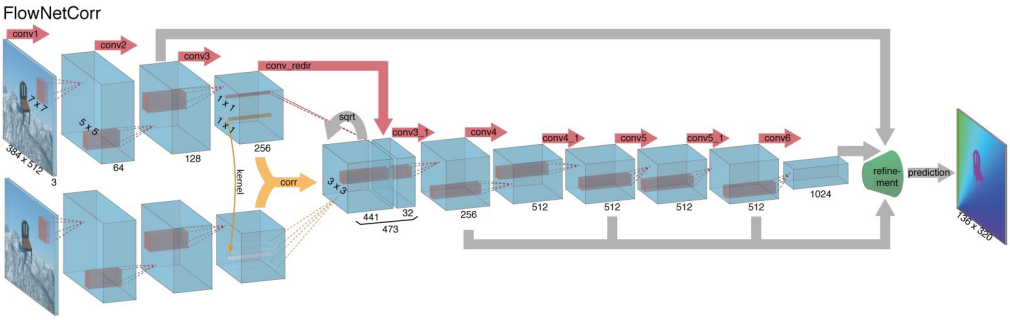


Figure 2.4: Encoder part of the FlowNet architecture. It takes two frames of a video and produce an estimated flow map from [11]

Decoder

The decoder part of FlowNet (shown in Figure 2.5) is very similar to U-Net [13], with upconvolutions (or transposed convolution, upsamples feature maps with locally connectivity and shared, learnable parameters) and skip connection. However, differently from U-Net, supervision is provided at every resolutions.

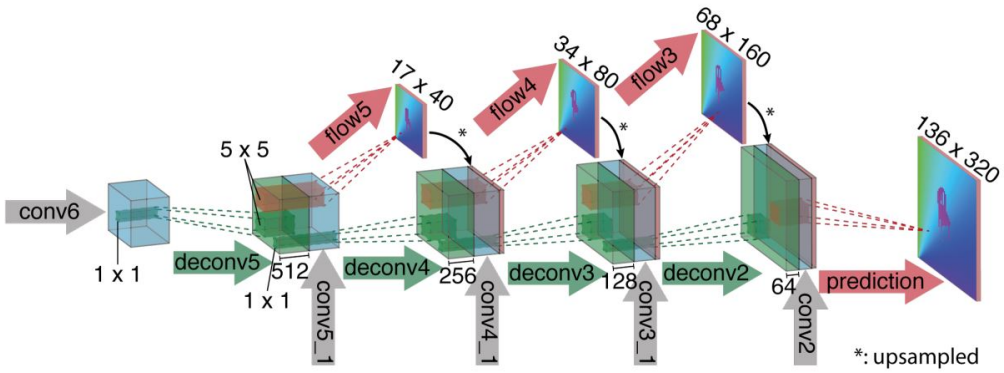


Figure 2.5: Decoder part of the FlowNet architecture. It upscale the map produced by the encoder part to an higher resolution from [11]

Chapter 3

Custom Implementation of Selected Neural Networks

As mentioned in the previous chapter, three different architectures where tested for the task of Template Matching: **WaldoNet, PatchNet** and **FlowNet**. All these architectures have been modified during my internship to make them suitable for the task and to achieve better results and predictions.

3.1 WaldoNet

I start implementing this network from the work of Xingrui Wang [14].

The overall architecture is composed by two networks: one that processes the query image (main network) and one that processes the template one (Hypernetwork). Different combination of object detection networks have been used (first is the main network and second is the Hypernetwork):

- VGG11 - CNN5
- VGG11 - VGG11
- VGG19 - VGG11
- VGG19 - VGG19

- ResNet50 - VGG11

CNN5 is a custom Convolutional Neural Network composed by five convolutional layers (convolution - Batch Norm - ReLU).

When VGG11 is used, the first five convolutions are frozen, while the last three are fine-tuned. Differently, when VGG19 is used, the first ten convolutions are frozen, while the last six are fine-tuned.

Experiments are done also with a decoder part, to gently brings the resolution back to the input one. This idea is the same at the base of FlowNet.

The key part of the architecture is how the main network and the hyper network are combined. The output of the last layer of the hyper network is used as weight of the kernel of the last convolution of the main network (the architecture proposed by Hossler paper is shown in Figure 2.1). Then, the tensor produced is bi-linearly interpolated to get back the original resolution. Finally, a Sigmoid layer is applied, to limit the range of value from 0 to 1. In doing so, each pixel of the binary mask predicted takes on a value between 0 and 1: 1 means that the template is detected in that pixel, 0 means that the template is not detected in that pixel. An example of predicted output is shown in Figure 3.1

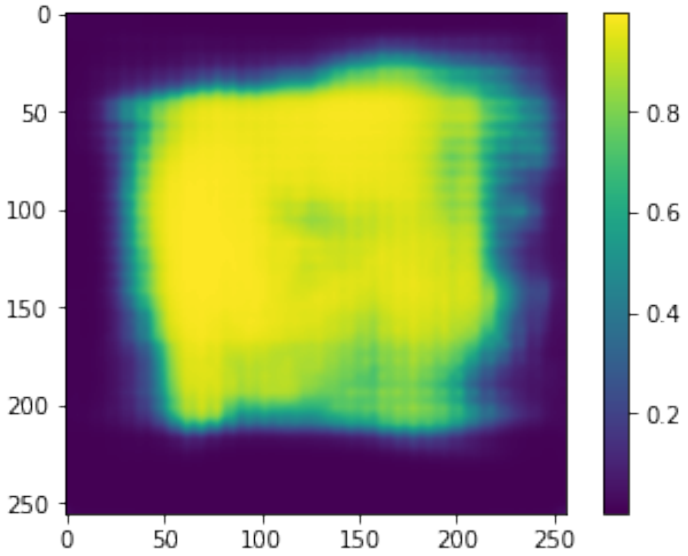


Figure 3.1: Example of prediction of WaldoNet. Each pixel takes value between 0 and 1 as shown on the color scale on the right.

3.2 PatchNet

The Mao et al. paper that describes the architecture of PatchNet has also the link for the GitHub that contains the code [15]. So, the implementation is taken from this repository. All the implementation below described are a custom version of the original one. Small changes are made to better fit the task. Since there is no need to compute bounding boxes, the bounding box regression part is removed. The decision to remove the box estimation is due to the fact that both WaldoNet and FlowNet do not compute them. Therefore, to be able to directly compare the results between these architectures, I decided to make this change in PatchNet.

- PatchNet
- PatchNetDec
- PatchNetBigPatches

PatchNetDec, as the name suggests, is a custom version of PatchNet that also

implement a decoder, that as the same structure of ModFlowNet. Since the results of PatchNetDec were better than PatchNet without it, PatchNetBigPatches has a decoder part as well.

The difference between PatchNet (and also PatchNetDec) and PatchNetBigPatches is only on the size and the numbers of patches. In PatchNet there are 256 patches of dimension 16×16, while PatchNetBigPatches processes 64 patches of dimension 32×32.

The number of patches and their dimensions are strictly related, as shown in Figure 3.2. As a matter of fact, for PatchNet with $N = 16$ and $K = 16$ (N^2 is the number of patches, $K \times K$ is the dimension of the patch), the template has dimension $3 \times 256 \times 256$; then the Patch Kernels have dimension $256 \times 3 \times 16 \times 16$.

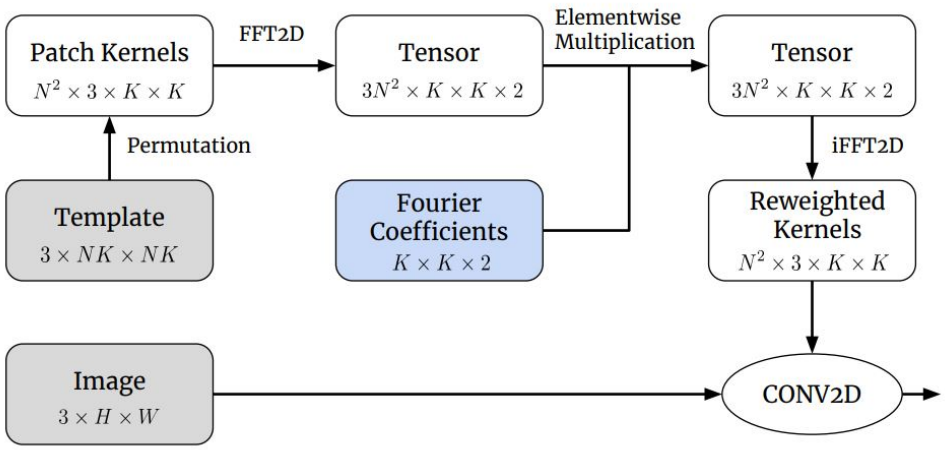


Figure 3.2: How PatchNet process the input image in the Patch Correlation Layer. The template is split into patches, re-weighted in the Fourier domain and converted to convolutional filters. From [10]

3.3 FlowNet

The implementation of FlowNet is taken from the official GitHub of NVIDIA [16] (in particular the implementation of FlowNetC) and then is customized.

The original FlowNet architecture processes two images at the time: they are

processed in a Siamese network (three convolutional layers) that extracts the features of both images. Then the features are put together thanks to a correlation layer. The activation is further processed by six convolution. To obtain the flow in output, a U-Net like structure is exploited: five upconvolutional layers brings the activation to bigger dimensions. For each upconvolution, a flow is predicted to help these convolutions to better reconstruct the flow.
I made three different custom architecture starting from FlowNet:

- ModFlowNetC
- ModFlowNetC2
- ModFlowNetC2v2

All the custom FlowNet implementations have generally the same structure of the original FlowNet, but there are few changes to adapt these architectures to the task. First, the resolution in output should be the same of the input. Then, in the decoder part of the network, is now predicted a binary mask and not anymore a flow. ModFlowNetC has this exact structure.

On the other hand, in the encoder part of ModFlowNetC2, the last convolutional layer was removed. This is done because a large reduction of the spatial dimensions decreases the performance of the network (probably it is harder to reconstruct the segmentation mask from a too low dimension activation).

Both ModFlowNetC and ModFlowNetC2 upscale their resolution, in the decoder part, until $1/2$ of the original resolution; then the tensor is upsampled by an Upsample layer with bi-linear interpolation. ModFlowNetC2v2, instead, use also the skip connection from the first convolutional layer to upscale the tensor to the original resolution.
In all the models a final Sigmoid layer is applied, to limit the range of value from 0 to 1. In doing so, each pixel of the binary mask predicted takes on a value between 0 and 1: 1 means that the template is detected in that pixel, 0 means that the template is not detected in that pixel.

Chapter 4

Definition and optimization of the training environment

During my internship in Datalogic, I have performed many virtual experiments using WaldoNet. This was the first network that I have modified and worked on as described in the previous chapter. The choice of many hyperparameter values common to the three networks, the loss function and other parameters have been tested on this network. In any case, different combinations of parameters and functions have been tested on FlowNet and PatchNet, in order to confirm the choices made with virtual experiments on WaldoNet.

A smaller number of virtual experiments have been done with PatchNet for two main reasons. First, the results seem to be not as promising as the one obtained with FlowNet and WaldoNet; second, it takes very long time to be trained, and the resources at my disposal were limited.

4.1 Training setting

In this chapter I will explain the setting and the configuration used for training the networks. First of all, the networks are trained on a cluster of GPUs kindly offered by Datalogic. The GPUs are NVIDIA® with 8 Gigabytes of Virtual RAM (VRAM). This specification is important because it limited some

choices (for example, the batch size).

The chosen programming language is Python, since it has libraries for building Neural Networks and its flexibility suits perfectly the exigences of this work. However, Python is slower to run than C++ and it is more prone to casting errors. The library chosen for build the Neural Networks is PyTorch [17], a choice made by my team. After some use I was totally convinced by this choice, because it is more easily customizable than TensorFlow [18].

4.2 Dataset

The choice of the dataset is a crucial part of the Neural Network training. There does not exists a specific datasets for the Template Matching task. However, the datasets for object detection could be easily adapted for this kind of problem. In addition, the networks are not trained on a company dataset for two reasons: the absence of ground truths and the fact that the network is thought to work on different industrial dataset, so a possible generalization is one of the key aspect of this development.

Hence, the dataset used to train the networks is *Pascal VOC 2010* [19] (from now on referred as *VOC* or *the dataset*). The train set is composed of 4998 images and the validation set has 5105 images. The dataset is quite general purpose, because it contains mainly images of animals, objects and persons.

A VOC sample is defined as a tuple composed by an image and an annotation file referred to it. In the annotation file, the coordinate of the corners of the bounding boxes containing the ”objects” (e.g., an animal, a person or a generic stuff) present in the image are stored.

A dataset for this task should contains triplets composed by the query image, the template and the ground truth. To arrange the VOC to match these specifications, the objects of Pascal VOC are used as templates (each template is the part of the query image inside a bounding box). The ground truth is a

binary image, with value 1 inside the bounding box of the template, and 0 otherwise.

Finally, the dataset class yields three images: the query image, the template image and the ground truth binary mask (as shown in Figure 4.1). The query image is resized to a resolution of $3 \times 256 \times 256$ pixels (3 is the number of the color channels, in this case RGB, then 256×256 is $width \times height$), keeping the aspect-ratio, and use PAD (value zero) as fill. The original sizes of the template image are kept and padded to reach $3 \times 256 \times 256$ pixels; if the template exceeds this sizes, the image is downsized to $3 \times 256 \times 256$, always keeping the original aspect-ratio. Both the query and the template image are RGB images. The ground truth has dimension $1 \times 256 \times 256$, and has only binary values.

Selected networks yield as prediction a segmentation mask. Unfortunately, it is not possible to retrieve a segmentation mask as ground truth using Pascal VOC. As future improvement, segmentation binary masks should be used as ground truth.

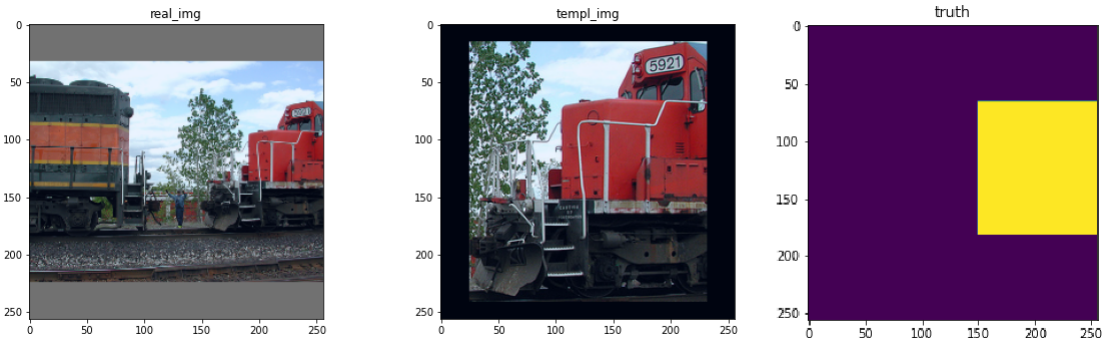


Figure 4.1: Example of datum of the Dataset: on the left the query image, in the middle the template image and on the right the ground truth

4.2.1 Data augmentation

In industrial settings the same template is used to match many images: for this reason, it is almost impossible that the template is present in the query

without any transformation (rotation, different scale, perspective changes...). Therefore, all the networks are trained on an augmented dataset. The validation set is augmented at instantiating time, so that the model is evaluated always on the same set; the training set, instead, is augmented at each epoch. This contributes also to make the network robust to all these transformations and allows to overcome the limits of the classical algorithms.

The augmentation is done in the following way: any transformation described below has a probability of 50% to be applied. The possible transformations are:

- **random rotation** [20]: It performs a rigid rotation of a given angle, which can assume values between -90ř and 90ř;
- **random crop** [21]: It crop a random portion of image and resize it to a given size. The area of the crop can have a scale value between 0.5 and 2.0. The scale is defined with respect to the area of the original image;
- **random perspective** [22]: It performs a random perspective transformation of the given image with a given probability. The degree of distortion can assume values between 0.0 and 0.1;
- **horizontal flip** [23]: It makes a horizontal flip of the given image randomly with a given probability;
- **vertical flip** [24]: It makes a vertical flip of the given image randomly with a given probability.

Then, to make the networks also robust with respect to intensity and light changes, additional color jitterring transformations [25] are applied:

- **random brightness**: the brightness factor is chosen uniformly from 0.5 to 1.0;
- **random contrast**: the contrast factor is chosen uniformly from 0.5 to 1.0;

- **random saturation** the saturation factor is chosen uniformly from 0.0 to 1.0;
- **random hue**: the hue factor is chosen uniformly from -0.2 to 0.2.

An example of an augmented image with all the transformations mentioned above is shown in Figure 4.2.

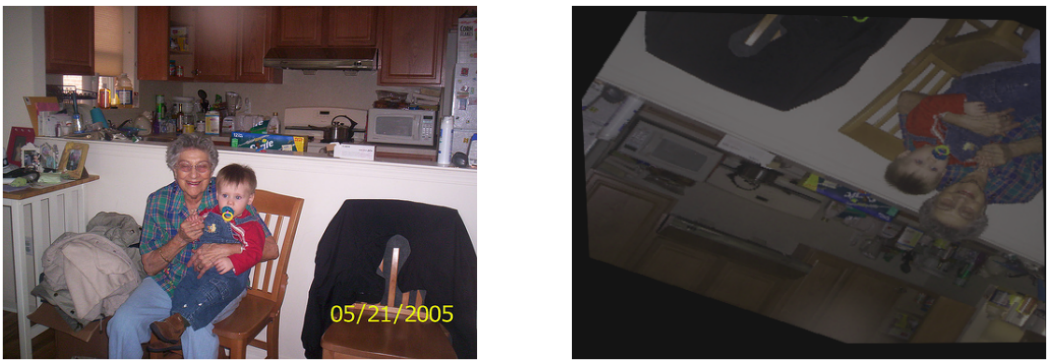


Figure 4.2: Example of augmented image. On the left the original image, on the right one of the possible version of the augmented ones

Data augmentation is key to obtain high performance. For instance, with WaldoNet, in the configuration VGG11-VGG11 (explained in section 3.1), the model trained with a non-augmented dataset reached 56.4% of IoU, while the same model trained with the augmented dataset reached 63.3% of IoU. IoU (Intersection over Union) is a score that measure the quality of the network, and it is defined below in section 4.3.

4.2.2 Data normalization

Normalizing the image data in the training is an important step. It ensures that each pixel in input has similar distribution, allowing the network to converge faster during the training [26].

The normalization [27] is done by subtracting the mean for each channel and then divide for the standard deviation, as formally described in equation 4.1.

$$output_c = \frac{(input_c - \mu_c)}{\sigma_c} \tag{4.1}$$

The equation refers to one channel, that of index c, but it is applied to all the RGB channels. The channel $output_c$ is that of the output image produce by the normalization; the channel $input_c$ is that of the input image. The mean and the standard deviation values are taken from the dataset ImageNet, and their values are; mean $\mu = [0.485, 0.456, 0.406]$; standard deviation $\sigma = [0.229, 0.224, 0.225]$, respectively for the R, G and B channels.
To make sure that normalizing the data helps the network to converge faster, I made an empirical experiment. With **ModFlowNetC2** the IoU reached with normalized training and validation is 67.9%, while, with the same configuration, the IoU reached with not-normalized data is 62.5%. Note that this is not a formal proof, but only an empirical test of the theory.

4.3 Metrics

Another crucial aspect of the training of a network is the choice of the metrics. Qualitative speaking, metrics give an indication of "how good" the network is performing.

To compare the results obtained with other papers and works, two metrics are used in this work:

- **Generalized Intersection over Union** (Generalized IoU);
- **Average Precision** (AP).

The Hossler paper suggests to use the Mean Squared Error (MSE) as metric to evaluate the quality of the networks. This metric is computed as in equation 4.2, where N is the total number of pixels, $\hat{y_i}$ is the $i-th$ predicted pixel, while y_i is the i-th value of the ground-truth.

$$\text{MSE} = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y_i})^2 \tag{4.2}$$

I choose to use Generalized IoU instead of MSE because, as shown later, IoU considers only pixels with value equal to 1, while MSE computes the average over all the pixels. For this reason, if the size of the object in the image is small (its dimensions are $\ll N$), the MSE value is close to zero both if the prediction is "accurate" or if only some pixels are right. In addition, as explained in the paper by Zhang et al. [28], any bounding box predicted where the second corner lies on a circle with a radius centered on the corresponding corner of the ground truth will have the same MSE value. This is shown in Figure 4.3, where the green box is the ground-truth bounding box, while the black one is the predicted bounding box. The MSE values for the two configurations shown in 4.3a) and b) are the same, while the IoU values are different.

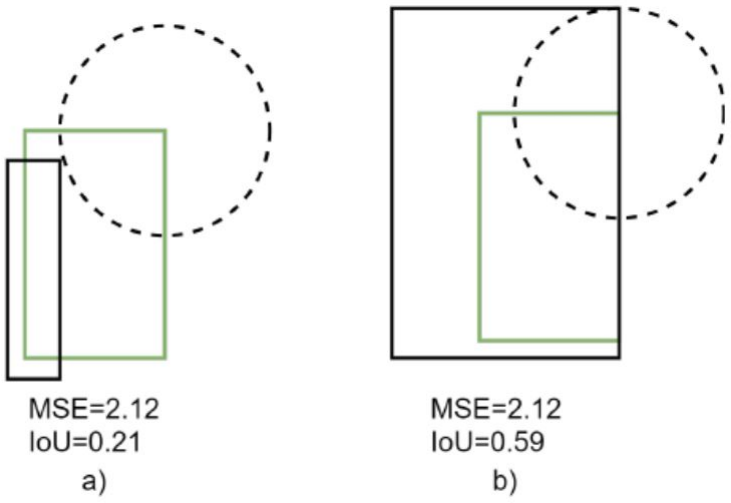


Figure 4.3: MSE and IoU scores of two predictions (black rectangles) on the same ground truth (green rectangles). From [28]

4.3.1 Generalized IoU

As said before, the Template Matching task is quite similar to the Object Detection one. Therefore, one choice could be to use the same metric, that is Intersection over Union (IoU). But since all the networks yield pixel segmentation instead of bounding boxes, the chosen final metric is the Generalized

IoU, that is measured at pixel level instead of box level.

The IoU measures the overlapping between the box predicted (BB_P) and the ground truth box (BB_{GT}), and is computed as in equation 4.3:

$$\begin{aligned} \text{IoU}(BB_P, BB_{GT}) = \frac{area\,of\,intersection}{area\,of\,union} = \\ = \frac{|BB_P \cap BB_{GT}|}{|BB_P| + |BB_{GT}| - |BB_P \cup BB_{GT}|} \end{aligned} \tag{4.3}$$

To better understand this score, qualitative speaking:

- IoU~ 0.55 corresponds to a partial overlap
- IoU ~ 0.75 corresponds to a good overlap
- IoU ~ 0.90 corresponds to a perfect overlap.

Given the definition of IoU, the Generalized IoU is computed as the mean of the IoU referred to a class (IOU_c):

$$\text{IoU}_c = \frac{area\,of\,intersection}{area\,of\,union} \tag{4.4}$$

The $area\,of\,intersection$ (AoI_c, i.e. the true positives, in the following also referred as TP_c) is computed as in equation 4.5.

$$AoI_c = \sum_{images} \#pixels\ where\ y_{uv} = c\ and\ \hat{y}_{uv} = c \tag{4.5}$$

While the $area\,of\,union$ (AoU_c) is computed as in equation 4.6.

$$AoU_c = \sum_{images} (\#pixels\ where\ y_{uv} = c\ + \#pixels\ where\ \hat{y}_{uv} = c) - AoI_c \tag{4.6}$$

Finally, the Generalized IoU (mIoU) is the mean across all the classes of IoU_c, as show in equation 4.7.

$$m\text{IoU} = \frac{1}{C}\sum_{c=1}^{C} IoU_c \tag{4.7}$$

From now on the Generalized IoU is referred simply as IoU.

4.3.2 Average Precision

There are two conflicting requirements in finding the template in the query image:

- Detect the highest number of pixels belonging to the template, i.e., achieve high True Positive Rate (or Recall);
- Do not detect "false" pixels in other regions, i.e., achieve high Precision.

The objective is to find a good trade off between these two requirements. In addition, the number of True Positive and False Negative is influenced by varying the minimum confidence to accept a detection. In this task, the threshold states whether the predicted pixel probability from the network is 0 (below the threshold) or 1 (above threshold). But in my thesis work, this threshold is given by implementation constraints, that is to say the function *round* of PyTorch, since is well optimised to work on GPU. Therefore, the threshold chosen is 0.5 for the training process. During the test, on the other hand, all the tensors are passed into the CPU, so the threshold could be also different from 0.5.

The evaluation of the detector is done across all its possible Recall / Precision regimes, changing the probability threshold. In this way, a precision-recall curve is obtained. The area under this curve is the Average Precision. For multiple classes classification, a mean is done across classes to obtain mean Average Precision (mAP). But in the case of this thesis, the classification is binary, and for this reason only the AP is taken.

The practical implementation of the AP is just an approximation of the real value, because it is not possible to compute with very high accuracy an integral

in Python. For this reason, for each threshold i, the Precision (Pr_i) and Recall (Re_i) values are produced. Then, the final AP is computed in equation 4.8 [29]

$$AP = \sum_i (Re_i - Re_{i+1}) \cdot Pr_i \tag{4.8}$$

4.4 Loss function

The objective of the training of Neural Networks is to minimize the loss function. The loss function is a proxy measure which avoids to directly optimize the accuracy (in our case the IoU or the AP). It is easier to optimize, but it still correlated with how good the classifier is.

The task of Template Matching, as described above, is a case of Binary Classification: as a matter of fact, our objective is to determine, for each pixel, if it is a zero or a one. Usually, for this kind of task, the loss used is the Binary Cross-Entropy (BCE) loss. In the equation 4.9 of the BCE loss, y_i is the truth value for the $i-th$ pixel, while $\hat{y_i}$ is the prediction for the $i-th$ pixel.

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y_i})] + [(1-y_i) \cdot \log(1-\hat{y_i})] \tag{4.9}$$

Since the loss function is a proxy function of the accuracy (in our case the IoU), I have introduced an inductive bias, since what I really want to maximize is the IoU score [30]. This was possible only because the Template Matching task for Neural Networks is casted as Binary Classification problem. Each pixel can assume value 0 (no template) or 1 (template found).

For this reason, I also employed a custom loss, called *IoU loss*, that is $= 1 -$IoU, where IoU is the same quantity computed in equation 4.3. The "one minus" in front is just because the loss function is minimized, so to maximize the IoU this term is necessary. In addition, virtual experiments are done also with a weighted combination of IoU loss and BCE loss. This is the one largely employed in the training of all the networks, because it yields the best results

(with them unevenly weighted: 60% the IoU loss and 40% the BCE loss). To sum up, three different losses have been tested:

- BCE loss;
- IoU custom loss;
- IoU + BCE custom loss.

4.5 Training

To make the experiments repeatable, all the random function are fed with a static seed (42), and the behaviour of the networks is set to *"deterministic"*. With all these precautions, the results yield after each restart of the machine are the same. All the networks are trained up to 100 epochs, with Adam [31] as **optimizer**, with a weight decay of 5×10^{-4}. The batch size chosen is 32.

To avoid overfitting and to allow the learning to converge faster, some regularization techniques are used.

First of all, I implemented **early stopping**, a method that treats training time as hyperparameter. When the model starts to overfit on the training set (i.e., the accuracy on the training grows, while the accuracy on the validation decrease), early stopping "stops" the training. Early stopping has one parameter, that is the *patience*, that state for how many epochs the network should achieve lower accuracy before interrupting the training. I set the patience to 7 epochs. With this technique implemented, none of the models reached 100 epochs of training.

Secondly, I used the scheduler **ReduceLROnPlateau** of PyTorch to reduce the learning rate if the validation IoU reaches a plateau (i.e., it does not improve or worsen within a certain range). Also in this case a *patience* parameter is set (5 epochs) and a minimum learning rate reachable is imposed (1×10^{-7}).

Chapter 5

Results and scores of the Neural Networks

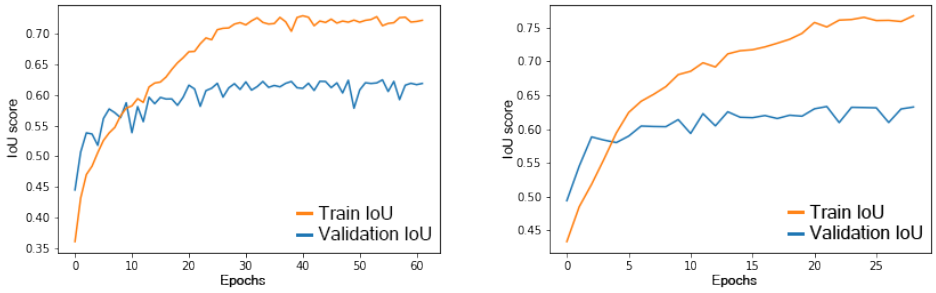
5.1 WaldoNet

As mentioned in the previous chapters, for this work WaldoNet was the ”playground” to test many configuration of loss, hyperparameters, batch size and so on. Therefore, in this section, for this network the results of more experiments than with the other two, PatchNet and FlowNet, are shown. The choices of loss, depth and hyperparameter, in particular for WaldoNet, greatly affects the reached scores. Different is the case on FlowNet, as it will be shown in section 5.3.

5.1.1 Loss choice

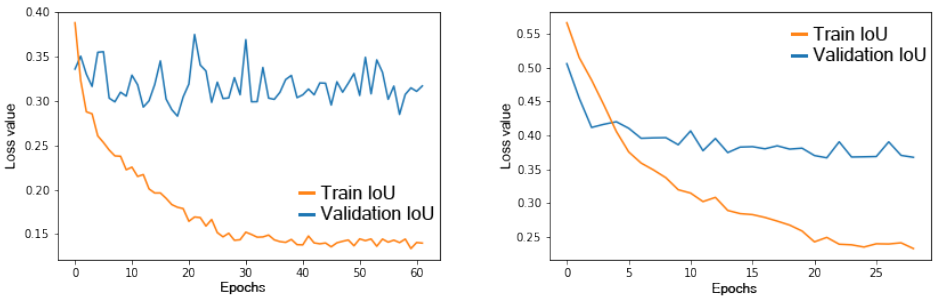
VGG11-VGG11: The choice of IoU in combination with the BCE custom loss allows a more regular training of the model (as shown in Figure 5.2 for the loss and Figure 5.1 for what concern the IoU), and the IoU increases faster than the model trained only with BCE loss.
The model trained with BCE loss reached 62.4% of IoU, while the one trained with IoU + BCE loss reached 63.3% of IoU.

Models from now on are intended as trained with IoU + BCE loss, unless differently specified.



(a) BCE loss (b) IoU + BCE loss

Figure 5.1: IoU graphs of VGG11-VGG11 model. Training with BCE loss 5.1a and IoU + BCE custom loss 5.1b



(a) BCE loss (b) IoU + BCE loss

Figure 5.2: Loss graphs of VGG11-VGG11 model. Training with BCE loss 5.2a and IoU + BCE custom loss 5.2b

5.1.2 Increasing the depth of the Network

The main and the Hyper networks extract features from the images thanks to well know CNNs for object classification and detection. Since increasing the depth of these networks in the task of object detection helps to reach a better accuracy, I have tried to do the same for the task of Template Matching. VGG19 is a CNN with 19 convolutional layers, while ResNet50 is deeper and exploits the residual connection to avoid high training errors.

VGG19 - VGG11: Increase the depth of the main network seems to produce

better results. The IoU reached is 68.2%.

ResNet50 - VGG11: Further increase the depth of the main network, e.g. using ResNet50, does not produce better results; conversely the IoU reached with this model is 64.9%.

5.1.3 Hyperparameter `d` of Hypernetwork

The hyperparameter d is the depth of the first fully connected layer of the Hypernetwork, and it greatly affects the amount of information transmitted from the template image to the query image [8].

VGG19-VGG11: The choice of d equals to 16 seems to be a good decision. Lower or larger `d` values yield worst results, respectively IoU of 63.5% with `d` = 8 and 62.2% with d = 32.

5.1.4 Decoder

To reach better results in IoU, I tried to implement a decoder part, that gently brings back the resolution of the computed tensor to the original image size.

VGG11-VGG11: The average IoU on the validation set is generally worst with respect to the model without the decoder part (60.6% vs. 63.3%). This could be due to the fact that the ground truth is the bounding box around the template; for this reason, models that yield a prediction with more background around the template are rewarded. As shown in Figure 5.3, the mask around the motorcycle is quite precise (image taken from epoch 5).

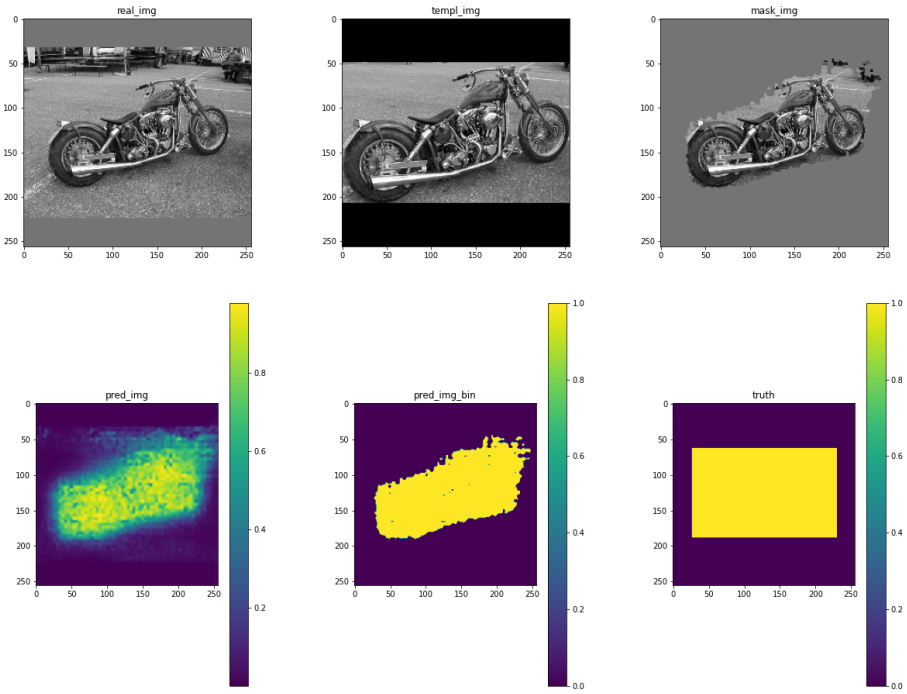


Figure 5.3: Example of prediction with decoder part (VGG11-VGG11)

5.2 PatchNet

The PatchNet network, as described in section 3.2, reaches an IoU of 46.6%, using the resolution of the input image of 256×256 pixels, which is the same resolution used in all the other network.

5.2.1 Different resolution

This result is much lower than the one obtained by WaldoNet or FlowNet at their first iteration. Because of this, I have tried two different strategies to align myself with the original Mao et al. paper [10]:

- feed the network with images at the same resolution used in the Mao et al. paper. So, the query image is resized to 120×120 pixels (in the paper they used 119×119, but it gives some problem with the dimension of the kernels), while the template is resized to 64×64. However, with these

new resolutions, the network has the worst performances, reaching a IoU of 25.3% only;

- in the dataset described in section 4.2, the size of the template is padded and only resized if it exceeds 256×256 pixels. Therefore, I have tried to feed the network with the template resized always to 256×256 pixels without keeping the aspect ratio. The idea behind is to avoid patches with only pad. But the reached value of the IoU, 36.8%, is still lower than that obtained in the first test.

After these disappointing results, the dataset used for the training of PatchNet-Dec e PatchNetBigPatches is the same described in section 4.2.

5.2.2 PatchNetBigPatches

The results obtained by using PatchNetBigPatches are similar to the one obtained by PatchNet. As a matter of fact, the reached IoU is 44.2%. For these training, I also kept track of the CPU time spent, that ranges between 15 and 24 hours. The CPU time for the training of the other networks, WaldoNet and FlowNet, is around 8 hours. This CPU consuming time is the reason why there are fewer experiments on PatchNet than for the other networks.

5.2.3 Decoder

With the use of the decoder, the obtained results seem more promising. PatchNet reaches a IoU score of 56.7%, while the use of BigPatches and the Decoder does not improve drastically the score, since it only reaches 48.8% of IoU. Finally, another experiment is done with the original resolution of 120×120 pixels for the query and 64×64 pixels for the template. The decoder turn out to be a big boost for this configuration, since the IoU reached is 46.3%.

5.3 FlowNet

FlowNet, differently from the case of WaldoNet, is less sensible to variations in the configuration.

ModFlowNetC, that has the same structure of FlowNet but adapted for the task of Template Matching, produces already impressive results, with a IoU score of 67.6%.

A test is made with up-convolutions till the original resolution. The result of the test is not encouraging, as the result was worst, with IoU score of 66.3%. In order to reach the original resolution, the Upsample layer is used. However, a slightly better result is obtained with a less deep encoder, ModFlowNetC2, that has reached a IoU score of 67.9%.

Then, the same test done using ModFlowNetC is repeated using ModFlowNetC2, by restoring the original resolution adding a up-convolution layer to the decoder. This version, called ModFlowNetC2v2, as expected from the previous test, does not performs better that ModFlowNetC2; as a matter of fact, it reaches only 66.6% of IoU.

5.4 Implementation of classical methods

The classical methods are taken from the OpenCV [32] library for Python. OpenCV provides a real-time optimized Computer Vision library. It provides both methods for ZNCC and SIFT. The function for Template Matching offered by OpenCV (from now on referred as *cv2 TM*) simply slides the template image over the input image and compares the template and patch of input image under the template one [33]. The similarity function used by this function is not exactly the ZNCC presented in section 1.1.2, but it is called

TM_CCOEFF (Correlation Coefficient) and its definition is given by the equation:

$$\text{TM\_CCOEFF}(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(T'(m,n)\cdot I'(i+m,j+n)) \tag{5.1}$$

where

$$\begin{aligned} T'(m,n) &= T(m,n) - \frac{1}{(M\cdot N)}\cdot\sum_{m'=0}^{M-1}\sum_{n'=0}^{N-1}T(m',n') \\ I'(i+m,j+n) &= I(i+m,j+n) - \frac{1}{(M\cdot N)}\cdot\sum_{m'=0}^{M-1}\sum_{n'=0}^{N-1}I(i+m',j+n') \end{aligned} \tag{5.2}$$

TM_CCOEFF is chosen among different type of similarity function, because it yields the best results.

The implementation of the SIFT algorithm is also present in the OpenCV library [34]. Both the SIFT and cv2 TM implementations are taken from the Zszazi GitHub repository [35] and adapted for this task.

5.5 Comparisons

To better summarize the results obtained by the networks that have been customized during my internship, I compared them with the classical methods and with the Blueye algorithm. This latter is a company proprietary software.

All methods have been tested on two datasets. The first is the original version of Pascal VOC 2010 (around 5000 images), where the image of the template is represented in the query without any transformation. The second is the same dataset, but this time augmented, with heavy transformations (rotations, scale changes, ...).

The classical algorithms produce as prediction four points of the bounding box enclosing the template found in the query. The networks, on the other hand, produce a binary segmentation mask, with value equal to 1 where the

template is found in the query. To make the predictions of the networks comparable with the classical algorithms, the function *masks_to_boxes* [36] of PyTorch library is employed. From the binary mask, this function provides the four corners of the bounding box. This method is a raw approximation of what the networks really predict. As a matter of fact, a wrongly predicted pixel far away from the others impacts greatly on the conversion mask to box.

A practical illustration of this is given in the following Figs. 5.4 and 5.5. Figure 5.4b shows an almost correct approximation of the bounding box around the predicted mask (5.4a). When an isolated, spurious pixel is present, as in the top part of Fig. 5.5a, a crude approximation of the bounding box is consequentially produced, Fig. 5.5b.

(a) Binary segmentation mask (b) Box retrieved from mask

Figure 5.4: Right approximation of box around the mask

(a) Binary segmentation mask (b) Box retrieved from mask

Figure 5.5: Wrong approximation of box around the mask

The table 5.1 encloses the results of networks and classical methods on the non-augmented dataset. It can be noticed that the best results are obtained by cv2 TM, followed by Datalogic's Blueye. The Neural Networks, also because of the problem of conversion between mask and bounding box previously described, do not perform well as the classical methods.

Not augmented set		
Method	**IoU**	**AP**
cv2 TM	95.5%	95.4%
SIFT (10 matches)	92.6%	92.5%
Blueye (conf 0.8)	94.1%	94.8%
ModFlowNetC2 boxes	79.9%	83.2%
WaldoNet (VGG19-VGG11) boxes	69.7%	71.5%

Table 5.1: Results of methods on non-augmented dataset

The strengths and advantages of the Neural Networks are evident with the heavy augmented dataset, with results reported in table 5.2. In this case, both of the classical methods proposed by OpenCV failed, and the drop in the percentage is quite impressive. On the other hand, Blueye reaches much higher IoU score with respect to the classical counterparts, almost comparable with the IoU score of the Neural Networks. However, the IoU score and the AP score of the Neural Networks, in particular ModFlowNetC, are very high, and the drop in percentage respect to the non augmented dataset is not drastic.

Augmented set		
Method	**IoU**	**AP**
cv2 TM	49.4%	53.5%
SIFT (10 matches)	48.5%	51.8%
Blueye (conf 0.8)	63.4%	66.9%
ModFlowNetC2 boxes	69.7%	74.8%
WaldoNet (VGG19-VGG11) boxes	67.4%	69.5%

Table 5.2: Score results of the methods on the augmented dataset

Even if there is a problem in converting the predicted binary masks of the networks to bounding boxes, the results obtained with the augmented dataset are impressive. However, even more astonishing are the results obtained without this conversion. Since is not a fair comparison to compare bounding boxes

with segmentation masks, the scores of the Neural Networks without conversion are reported in separate tables.

In table 5.3 the results of the networks on the non-augmented dataset are shown. In table 5.4 the results of the networks on the augmented dataset are listed. It can be noticed that the IoU scores are similar to the bounding box conversion, while the AP scores are much higher.

Not augmented set		
Method	**IoU**	**AP**
ModFlowNetC2	80.7%	96.9%
WaldoNet (VGG19-VGG11)	70.7%	89.5%

Table 5.3: Results of the Neural Networks considering the predicted binary masks on non-augmented dataset

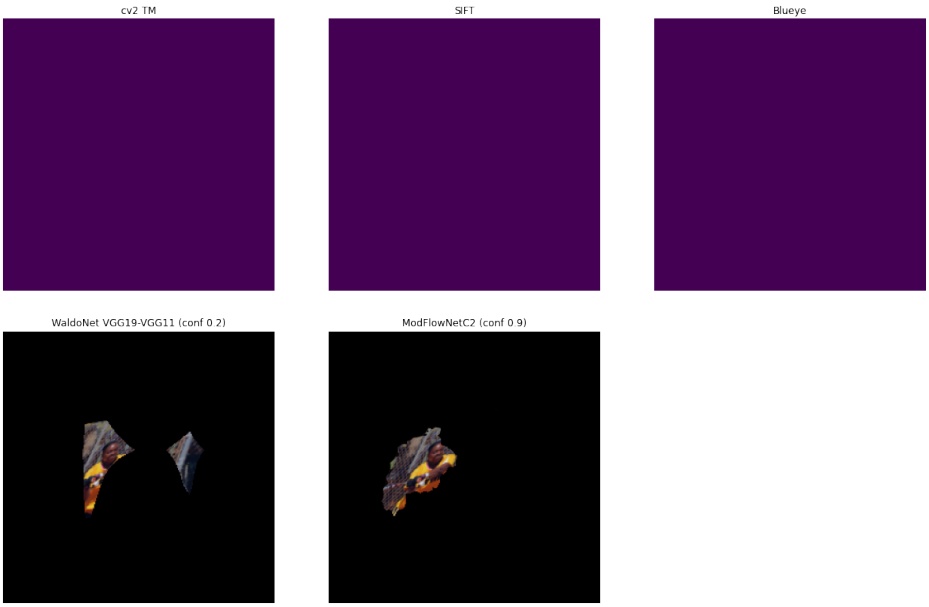
Augmented set		
Method	**IoU**	**AP**
ModFlowNetC2	69.5%	94.1%
WaldoNet (VGG19-VGG11)	68.1%	89.0%

Table 5.4: Results of the Neural Networks considering the predicted binary masks on augmented dataset

Finally, an example of an image obtained with non-affine transformation (perspective distortion [22] with degree of distortion equals to 0.7) is shown in Figure 5.6a. The query and the template are then processed by classical algorithms and the two Neural Networks (ModFlowNetC2 and WaldoNet VGG19-VGG11), and the predictions (segmentation masks for the networks and bounding boxes for the classical algorithms) are shown in Figure 5.6b. The classical methods show their limits, since they have not predicted any bounding box, while the Neural Networks show their strength. ModFlowNetC2 makes an high confidence prediction (around 90%), while WaldoNet has a confidence of 20% and also wrongly detect some background on the right of the query

image.

(a) Example of non-affine transformation on the query



(b) Template Matching results of classic algorithms and Neural Networks

Figure 5.6: Detection of the template in the query. In order cv2 TM, SIFT, Blueye, WaldoNet (VGG19-VGG11) with confidence 0.2, ModFlowNetC2 with confidence 0.9

Conclusions

The Neural Networks customized and tested in this thesis (WaldoNet, FlowNet and PatchNet) are a first exploration and approach to the Template Matching task. The possibilities of extension are therefore many and some are proposed below.

During my internship, I have analyzed the functioning of the classical algorithms and adapted with deep learning algorithms. The features extracted from both the template and the query images resemble the keypoints of the SIFT algorithm. Then, instead of similarity function or keypoints matching, WaldoNet and PatchNet use the convolutional layer to compare the features, while FlowNet uses the correlational layer. In addition, I have identified the major challenges of the Template Matching task (affine/non-affine transformations, intensity changes...) and solved them with a careful design of the dataset.

Overall, the results obtained with the Neural Networks seem promising, since they overcome some limits of the classical algorithms. They are robust to many distortions and even to non-affine transformations, one of the unsolved challenge of classical approaches. Still, classical algorithms perform better with images without distortions and their performances do not vary when different set of images are tested. Performances of the networks on unseen datasets can be increased with fine-tuning [1] on a small set of new images or even with distorted copies of only one new image.

[1] Fine-tuning is a technique which consists in taking a trained network and retrained with new images, but usually with a smaller learning rate. In this way the already learned weights are not completely changed.

Before mentioning some possible improvements, it is relevant to notice that some important decisions are strictly related to the choice of the output: bounding boxes or segmentation masks. The type of the output is strongly dependent on the use of the network.

In the bounding box case, the final layers of the above described networks should be replaced by a regression head predicting the four corners, instead of the sigmoid layer. Instead, in the segmentation mask case, a more suitable dataset for the training should be used. As a matter of fact, the dataset used in this thesis has bounding boxes as ground truth. Therefore, a more appropriate dataset should have as ground truth segmentation masks.

Finally, since classical algorithms have been studied for decades, their performances on not-augmented dataset are unbeaten. Therefore, an inductive bias[2] could be introduced in the networks, to direct the training in the ”right” direction.

[2]In machine learning, the term inductive bias refers to a set of (explicit or implicit) assumptions made by a learning algorithm in order to perform induction, that is, to generalize a finite set of observation (training data) into a general model of the domain [37].

Acknowledgment

This thesis is the result of my curricular internship in the company Datalogic S.p.A. [1]. The hardware, in particular server and GPUs, was provided by the company itself. I am grateful to this company for the warm hospitality and also to let me share this document.

The idea of this work is by Dott. Maurizio De Girolami, the head of the R&D group in which I worked in. I want to make a warm thanks to Maurizio and all his team, that have welcomed, advised and encouraged me. A special thanks goes to Dott. Angelo Carraggi, my company supervisor, with whom I had many exchange of ideas. He also provided me many papers mentioned in this thesis, that were the basis of my activity.

I want to acknowledge my supervisor, Professor Samuele Salti, for his patience and suggestions, and also for making me passionate about this subject.

Bibliography

[1] Automatic Data Capture and Process Automation. Datalogic. (n.d.). Retrieved September 26, 2022, from https://www.datalogic.com/

[2] Wikimedia Foundation. (2022, September 22). Datalogic. Wikipedia. Retrieved September 25, 2022, from https://it.wikipedia.org/wiki/Datalogic

[3] L. Di Stefano, S. Mattoccia, F. Tombari. ZNCC-based template matching using bounded partial correlation. Pattern Recognition Letters 26 (2005) 2129. https://doi.org/10.1016/j.patrec.2005.03.022

[4] Wikimedia Foundation. (2022, August 6). Convolution theorem. Wikipedia. Retrieved September 26, 2022, from https://en.wikipedia.org/wiki/Convolution_theorem

[5] L. Di Stefano, S. Mattoccia, "Fast template matching using Bounded Partial Correlation", Machine Vision and Applications (MVA), 13(4), pages 213-221, February 2003 http://vision.deis.unibo.it/~smatt/Papers/JMVA2003/JMVA_2003.pdf

[6] Wikimedia Foundation. (2022, August 22). Scale-invariant feature transform. Wikipedia. Retrieved September 13, 2022, from https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

[7] Bandara, R. (2017, August 30). Bag-of-features descriptor on SIFT features with opencv (BOF-SIFT). CodeProject. Retrieved September 26,

2022, from https://www.codeproject.com/articles/619039/bag-of-features-descriptor-on-sift-features-with-o?display=print&fid=1836776&df=90&mpp=25&prof=True&sort=Position&view=Normal&spc=Relaxed&fr=176

[8] Thomas Hossler, *Where's Waldo? A Deep Learning approach to Template Matching*, Department of Geological Sciences, Stanford University, 2017 http://cs231n.stanford.edu/reports/2017/pdfs/817.pdf

[9] Wikimedia Foundation. (2022, September 20). Where's wally? Wikipedia. Retrieved September 24, 2022, from https://en.wikipedia.org/wiki/Where%27s_Wally%3F

[10] Huizi Mao, Sibo Zhu, Song Han, & William J. Dally (2021). PatchNet - Short-range Template Matching for Efficient Video Processing. CoRR, abs/2103.07371.

[11] Fischer, P., Dosovitskiy, A., Ilg, E., Häusser, P., Hazırbaş, C., Golkov, V., Smagt, P., Cremers, D., & Brox, T.. (2015). FlowNet: Learning Optical Flow with Convolutional Networks.

[12] D. Ha, A. Dai, and Q. V. Le. HyperNetworks. 2016 https://arxiv.org/abs/1609.09106

[13] Olaf Ronneberger, Philipp Fischer, & Thomas Brox (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. CoRR, abs/1505.04597.

[14] XingruiWang. (n.d.). Xingruiwang/wheres-waldo: A Pytorch implementing of a deep learning approach to template matching. USIE Hypernet + VGG to match the templates. GitHub. Retrieved September 24, 2022, from https://github.com/XingruiWang/wheres-waldo

[15] RalphMao. (n.d.). Ralphmao/PatchNet. GitHub. Retrieved September 17, 2022, from https://github.com/RalphMao/PatchNet

[16] Nvidia. (n.d.). Nvidia/FLOWNET2-pytorch: Pytorch implementation of FlowNet 2.0: Evolution of optical flow estimation with Deep Networks. GitHub. Retrieved September 26, 2022, from https://github.com/NVIDIA/flownet2-pytorch

[17] Pytorch. PyTorch. (n.d.). Retrieved September 26, 2022, from https://pytorch.org/

[18] Tensorflow. TensorFlow. (n.d.). Retrieved September 26, 2022, from https://www.tensorflow.org/

[19] *Visual object classes challenge 2010 (VOC2010)*. The PASCAL Visual Object Classes Challenge 2010 (VOC2010). (n.d.). Retrieved September 2, 2022, from http://host.robots.ox.ac.uk/pascal/VOC/voc2010/index.html

[20] Random Rotation. RandomRotation - Torchvision main documentation. (n.d.). Retrieved September 22, 2022, from https://pytorch.org/vision/stable/generated/torchvision.transforms.RandomRotation.html

[21] Random Resized Crop. RandomResizedCrop - Torchvision main documentation. (n.d.). Retrieved September 22, 2022, from https://pytorch.org/vision/main/generated/torchvision.transforms.RandomResizedCrop.html

[22] Random Perspective. RandomPerspective - Torchvision main documentation. (n.d.). Retrieved September 22, 2022, from https://pytorch.org/vision/main/generated/torchvision.transforms.RandomPerspective.html

[23] Random Horizontal Flip. RandomHorizontalFlip - Torchvision main documentation. (n.d.). Retrieved September 21, 2022, from https://pytorch.org/vision/main/generated/torchvision.transforms.RandomHorizontalFlip.html

[24] Random Vertical Flip. RandomVerticalFlip - Torchvision main documentation. (n.d.). Retrieved September 22, 2022, from https://pytorch.org/vision/stable/generated/torchvision.transforms.RandomVerticalFlip.html

[25] Color Jitter. ColorJitter - Torchvision main documentation. (n.d.). Retrieved September 22, 2022, from https://pytorch.org/vision/main/generated/torchvision.transforms.ColorJitter.html

[26] B, N. (2019, January 18). Image data pre-processing for Neural Networks. Medium. Retrieved September 19, 2022, from https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258

[27] Normalize. Normalize - Torchvision main documentation. (n.d.). Retrieved September 19, 2022, from https://pytorch.org/vision/stable/generated/torchvision.transforms.Normalize.html#torchvision.transforms.Normalize

[28] Zhang, Xin & Han, Liangxiu & Robinson, Mark & Gallagher, Anthony. (2021). A GANs-based Deep Learning Framework for Automatic Subsurface Object Recognition from Ground Penetrating Radar Data. IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3064205.

[29] Gad, A. F. (2021, April 9). Mean average precision (MAP) explained. Paperspace Blog. Retrieved September 18, 2022, from https://blog.paperspace.com/mean-average-precision/

[30] van Beers, F. & Lindström, Arvid & Okafor, Emmanuel & Wiering, Marco. (2019). Deep Neural Networks with Intersection over Union Loss for Binary Image Segmentation. 10.5220/0007347504380445.

[31] Kingma, Diederik P., & Jimmy, Ba. "Adam: A Method for Stochastic Optimization." (2014)

[32] Home. OpenCV. (2022, August 22). Retrieved September 20, 2022, from https://opencv.org/

[33] Template matching. OpenCV. (n.d.). Retrieved September 20, 2022, from https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

[34] Introduction to SIFT (scale-invariant feature transform). OpenCV. (n.d.). Retrieved September 16, 2022, from https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

[35] Zszazi. (n.d.). Zszazi/opencv-template-matching-and-sift. GitHub. Retrieved September 20, 2022, from https://github.com/zszazi/OpenCV-Template-matching-and-SIFT

[36] Masks to boxes. masks_to_boxes - Torchvision main documentation. (n.d.). Retrieved September 20, 2022, from https://pytorch.org/vision/main/generated/torchvision.ops.masks_to_boxes.html

[37] Hüllermeier, E., Fober, T., Mernberger, M. (2013). Inductive Bias. In: Dubitzky, W., Wolkenhauer, O., Cho, KH., Yokota, H. (eds) Encyclopedia of Systems Biology. Springer, New York, NY. https://doi.org/10.1007/978-1-4419-9863-7_927