Deep Object Ranking for Template Matching

Jean-Philippe Mercier Ludovic Trottier Philippe Giguère Brahim Chaib-draa
Laval University
[jean-philippe.mercier.2;ludovic.trottier.1]@ulaval.ca
[philippe.giguere;brahim.chaib-draa]@ift.ulaval.ca

Abstract

Pick-and-place is an important task in robotic manipulation. In industry, template-matching approaches are often used to provide the level of precision required to locate an object to be picked. However, if a robotic workstation is to handle numerous objects, brute-force template-matching becomes expensive, and is subject to notoriously hard-to-tune thresholds. In this paper, we explore the use of Deep Learning methods to speed up traditional methods such as template matching. In particular, we employed a Single Shot Detection (SSD) and a Residual Network (ResNet) for object detection and classification. Classification scores allows the re-ranking of objects so that template matching is performed in order of likelihood. Tests on a dataset containing 10 industrial objects demonstrated the validity of our approach, by getting an average ranking of 1.37 for the object of interest. Moreover, we tested our approach on the standard Pose dataset which contains 15 objects and got an average ranking of 1.99. Because SSD and ResNet operates essentially in constant time in a Graphics Processor Unit, our approach is able to reach near-constant time execution. We also compared the F_1 scores of `LINE-2D`, a state-of-the-art template matching method, using different strategies (including our own) and the results show that our method is competitive to a brute-force template matching approach. Coupled with near-constant time execution, it therefore opens up the possibility for performing template matching for databases containing hundreds of objects.

1. Introduction

The automation of simple industrial operations, particularly for the small scale, fast turnaround robotics equipment, is seen as a key concept of the fourth industrial revolution (often dubbed *Industry 4.0*). A desirable characteristic for its acceptance would be the ability for a non-technical person to program advanced robotic equipment, such as robotic arms. The *Baxter* robot, from *Rethink Robotics*, exemplifies this concept, by being able to perform pick-and-place operations using intuitive programming methods. In this system, training by examples replaces the traditional and more tedious computer programming by experts.

In this paper, we explore the idea of improving both the robustness and speed of a key and traditional component used in pick-and-place solutions, namely *template matching*. We aim at exploiting recent developments in Deep Learning on object detection and classification to first detect, then sort, putative objects before performing template matching. These networks are trained by employing databases of images collected in a simple manner. This way, we stay within the philosophy of training by example. As our approach is fairly conservative (the final object localization is still performed via template matching), we believe that it might speed up the acceptance of Deep Learning approaches in the industry.

Deep neural networks, particularly the convolutional networks, have established themselves as clear winners in many classification tasks. Moreover, with the use of GPU (Graphics Processor Units), they have shown to perform classification over a thousand object categories in real-time. This speed is possible because *a)* GPUs provide massive parallel computing capabilities, up to several thousands of cores and *b)* the features are extracted only once by the neural network, and are available to all object classifiers, since the object classification is performed in the fully-connected layers at the top of the network architecture. By contrast, parallelizing template matching libraries is tedious, as it requires significant hand-crafting and tuning. Moreover, current template-matching approaches do not share the feature extraction pipeline across templates, making them inherently less efficient at exploiting the massive parallelism of GPUs.

Template matching [8] has been used for a long time in machine vision, and now has been widely accepted in industry. It has proven to perform well in retrieving the pose of an already-identified object. However, it comes with a number of known flaws. For instance, the matching speed is dependent on the number of templates the method is trying to match. Furthermore, the final decision about whether an

object has been well detected/located is based on a threshold on the matching score. Thresholds are notoriously difficult to select properly, as there is an intrinsic compromise to be made between a low threshold (which results in many false detections) and a high threshold (which has less false detections but will also reject good detections).

We propose a generic template matching pre-processing step that takes advantage of the success and properties of Deep Learning for object detection and recognition. A diagram of our approach is shown in Fig. 1. It consists in detecting, classifying and ranking the seen objects before using template matching. As such, our approach is completely agnostic to the subsequent step. This *cascading* philosophy is not novel; In fact, it is a common approach in vision-related tasks, in order to accelerate processing. For instance, the Viola-Jones real-time detection technique [31] uses a number of progressively more discriminative (and expensive) classifiers. Neural networks have also been employed in cascade for object understanding, notably a two-staged approach for grasp location detection in range images [18]. Deep Learning approaches might be used for initializing model-matching, such as with the tracking of body parts, as they are more robust due to their larger basin of attraction [7]. In some sense, our approach uses the same philosophy, where the coarse part (object detection and classification) is performed with a Convolutional Neural Network, while the fine-tuning of object pose estimation is completed via template matching.

A significant advantage of our proposed approach is that detection and classification modules, which are often expensive, are essentially executed in constant time. Indeed, one forward-pass of a typical deep neural network can recognize amongst a thousand objects in less than 80 *ms* [17]. On the other hand, if object recognition is to be done by template matching, one needs to run it until a sufficiently-high scoring template has been found. In this case, the average number of templates that will be tried by this template matching method should be, on average, around half of the total number of templates. As we will demonstrate, by pre-ranking templates by confidence of our object classifier before matching begins, this number tends to be significantly less. It can thus allow for fast matching, even if there is a large number of templates, as long as classification precision is sufficiently high.

Another positive effect of our approach is an increase in robustness for object classification. In some template-matching approaches, any score higher than the user-selected threshold results in a positive match and the termination of the algorithm. By using a low threshold on template matching scores (higher recall), it results in a high number of bad detections (lower precision), as the first template tried will return a positive match. In another kind of approach, every template is processed, and the template which scored the best is selected. By pre-ranking templates in a first step, our method improves the performance of the "first-match" approach, and get similar results to the "best-match" approach, while having to process significantly less templates.

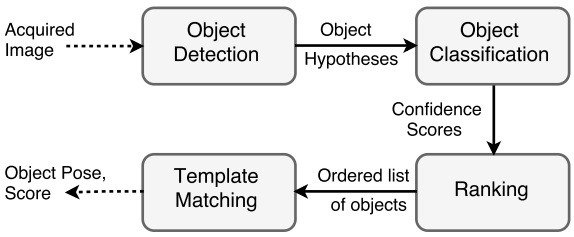


Figure 1: Block diagram of our proposed method. Object detection and classification are performed by Deep Neural Networks, providing robustness and speed. Objects in the database are sorted by their classification scores before being passed to the template matching module. Objects are processed one after the other until a template results in a score higher than a user-defined threshold, upon which the algorithm terminates.

2. Related Work

Some approaches have been used to reduce the computational load of template matching methods, which can be heavy, especially for those that searches for the complete set of possible transformations [16, 22, 28]. In [23], they used a convolutional neural network to predict the "matchability" of templates across different transformation of the input image. They then select regions around local maximums from the computed matchability maps as templates for any selected template matching method. In [2], they employed a Support Vector Regression to find a function which is used to generate template candidates at runtime, but strictly applied to visual tracking.

Object detection methods based on keypoint descriptors work well on textured objects [3, 21]. However, by looking at specific visual patterns on surfaces, they do not perform well on untextured objects by nature, and they tend to require heavy calculations [1]. On the opposite, template matching methods work well on objects with little visual texture [12, 24, 15], provided that they are located on a distinctive background. Since a significant number of objects in an industrial context are devoid of textures (such as metallic parts that are freshly machined), template matching tends to be more appropriate for those environments.

`LINEMOD` [12] is among the state-of-the-art methods for template matching on textureless objects. It uses quantized color gradients for the 2D version (`LINE-2D`) along with surface normals for the 3D version. They use some opti-

mizations, such as a look-up table for the color gradients and linearization of gradient response maps, to increase the speed of their method. To detect objects under varying poses, it requires a training phase in which a significantly large collection of templates needs to be acquired from different viewpoints. In practice, full 3D models are needed to generate these viewpoints. DTT-3D [24] improved the performance and speed of LINEMOD. In their approach, they train a SVM with positive and negative examples to learn the discriminative regions of templates, and use only them in the matching process. They also divide the templates of each objects into different clusters using their own similarity measure and the clustering method of [13], which represents a group of similar templates with what they refer as a "cluster template". On the other hand, Hashmod [15] uses a hash function on concatenated LINEMOD descriptors (color gradient and surface normals) to match templates. Following a sliding window approach, they compute hashed descriptor at each image locations and if the descriptor is close enough to a descriptor in the hash table (which contains the hashed descriptors for each object under a certain pose), it is considered a match. [32] followed a similar approach, but instead of using a hash function, they trained a CNN to learn descriptors for varying poses of multiple objects. Matching was performed with a Nearest Neighbor search. As opposed to the previously mentioned methods, [5] used only monocular grayscale images to retrieve the 3D pose of known objects (they still had a 3D model of the object), as methods based on depth sensors tend to fail for metallic objects due to their specular surfaces. They based their pose estimation method on the 2D reprojection of control points located on the different and manually labeled parts of the 3D models. From the 3D-2D correspondences of all control points of the different parts, they solve the PnP problem to estimate the 3D pose, which results in a method more robust to occlusions.

3. Overview of the proposed method

Fig. 1 shows a workflow of template matching including our proposed pre-processing steps, which detect and rank objects before further processing. As it can be observed, our method is highly modular. As such, it allows a drop-in replacement of any module (e.g. template matcher) if any better method becomes available. In the rest of this Section, we describe the implementation details of each of the modules.

3.1. Object Detection

The first step in our method is to perform object hypotheses detection. We have used the state-of-the-art *Single Shot Multi Box Detector* (SSD) [20] trained on the Microsoft COCO dataset [19], which contains more than 2 million instances of the 80 object categories. SSD is a convolutional neural network framework that can take as input images resized to 300×300 (SSD300) or 500×500 (SSD500) pixels, and outputs bounding boxes and classification scores of every class. On proper hardware (GPU), it can run in real-time (58 FPS for SSD300) or close to real time (23 FPS for SSD500). Objects are considered as detected when the score of a bounding box is higher than a user-specified threshold τ_{SSD}. In our experiments, we used the SSD500 version with a low threshold $\tau_{SSD} = (0.08)$. We preferred SSD500 over SSD300 as it outputs more bounding boxes, while still being sufficiently fast. An example of object detection on our object database with SSD is shown in Fig. 2. We can see in Fig. 2a that some objects are undetected. It sometimes happens that an object will be considered as part of the background in the presence of some clutter, i.e. when multiple objects are near each other. This effect tend to decrease when near objects are removed, as shown in Fig. 2b and Fig. 2c, where we can see that both the green and the gold were finally detected, once there was fewer objects around them. In a robotics grasping application where the goal is to empty a work cell from all objects, this is usually not a significant issue. Indeed, objects around the undetected one would eventually all be grasped. Once the undetected objects are the only objects remaining, the detection threshold can be decreased to get more object hypotheses.

3.2. Object Classification

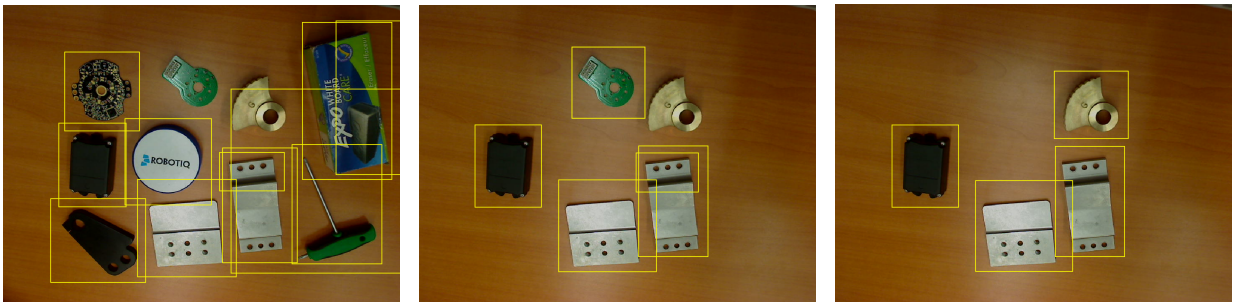
To classify objects, we used a Residual Network (ResNet) [9, 11]. It has been recently demonstrated as a top contender for ImageNet object classification. For instance, [9] won the ILSVRC 2015 classification task. The key idea of ResNet is that residual mappings are easier to optimize than absolute ones. To take advantage of this fact, ResNet adds skip connections that bypass several convolutional layers using identity mappings to form shortcuts in the network, called residual blocks. With a combination of Batch Normalization (BN) [14] and MSR initialization [10], stacking residual blocks significantly improves the training efficiency by reducing the vanishing gradient degradation. This allows learning very deep networks far more easier than it is without residual connections.

3.2.1 Model Description

The ResNet architecture and the residual blocks are shown in Fig. 3, where we can formulate each residual block as follows:

$$x_{l+1} = x_l + \mathcal{F}(x_l, W_l)\,, \tag{1}$$

where x_l and x_{l+1} are the input and output of the l-th residual block and $\mathcal{F}$ is the residual mapping. As seen in Fig. 3a, we opted for a Conv-PELU-Conv-BN residual mapping, where PELU stands for Parametric Exponential Linear Unit, an adaptive activation function that showed high



(a) (b) (c)

Figure 2: Image sequence showing object hypotheses retrieved from our object detection module. One can notice that as the clutter in the workspace diminishes, objects previously undetected by SSD can now be detected.

performance on several image classification task over other alternatives [29, 25].

The overall 18-layers network, which has about 11.7 M parameters to be learned, is shown in Fig. 3b. It takes as input a 224×224 color image, and using a series of various transformations, outputs a c-dimensional vector of class conditional probabilities. The first convolutional layer contains 64 7×7 filters, which are convolved with a 2×2 stride and 3×3 zero-padding. We will denote such convolutional layer as Conv(64, 7×7, 2×2, 3×3). PELU is then applied on the resulting feature maps of the first layer, followed by a max pooling layer MaxPool(3×3, 2×2, 1×1). The input then passes through four twice-replicated residual blocks, containing respectively 64, 128, 256 and 512 filters in each of their convolutional layers. The network performs a final PELU activation, followed by a global average pooling AvgPool(7×7, 1×1) and linear layer $512 \rightarrow c$, where c represents the number of classes. When the network doubles the residual block number of filters, the input spatial dimensionality is reduced by half. This particularity allows the network to gradually pool spatial information into its feature maps and extract higher-level information. For instance, between ResBlock 64 and ResBlock 128, the input spatial dimensionality is reduced from 56×56 to 28×28 pixels. However, this poses a problem for the first ResBlock 128, as we cannot apply an identity skip connection because the input and output spatial dimensionality are different. In this case, we use as skip connection a convolutional layer with 1×1 filters with a stride of 2×2, currently known as a type B connection [11].

3.2.2 Training Details

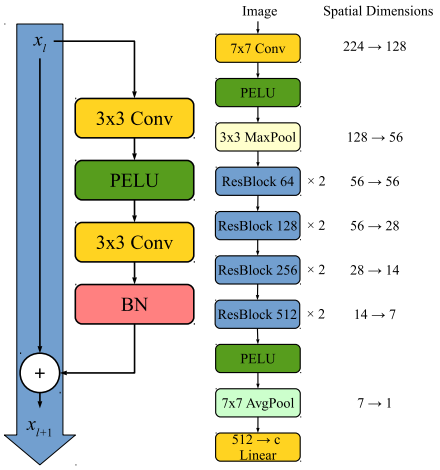
In spite of the recent experiments showing that fine-tuning an ImageNet [6] pre-trained network gives good results [26], we instead opted to trained our network from scratch. We relied heavily on data augmentation techniques for improving generalization performances. This powerful technique allows generating new images from existing datasets to artificially grow their sizes, and thus help the network converge at a better local optimum. During training, prior to inputting the image to the network, we applied the five following random transformation: color jittering (contrast, lighting and saturation are changed by a small amount), lighting Noise (a PCA-based noise in RGB space from [17]), color standardization (we subtracted the mean and divided by the standard deviation of each RGB channel to get a mean of 0 and unit-variance), horizontal flip (images are randomly flipped horizontally), rotation (images are randomly rotated from 0 to 360 degrees). At test time, we simply did color standardization. All transformations were performed on the fly during training, and due to the randomness of the process, the network had effectively seen a new image at each iteration, even though our dataset only contained 50 images per object (see section 4.1 for more information about the dataset). For training the network, we used the hyper-parameters and the Torch implementation of [29]. Note that this number of images is much less than in the standard ImageNet database, which has between 700-1300 images per class [6].

3.3. Object Ranking

This simple step takes the confidence scores returned by the classification step as input, creating a list ordered by their classification scores. The most likely objects to appear in the image, according to our classifier, are therefore placed at the beginning of the list. The whole algorithm is summarized in Algorithm 1.

3.4. Template Matching

To match object templates, we employed the `LINE-2D` [12] algorithm, which uses the orientation of gradients to perform matching. `LINE-2D` is considered as one of the state-of-the-art template matching method on monocular color images. We used its default implementation in the OpenCV library [4], with 2 pyramid levels and 63 features.



(a) Residual block (b) Residual network

Figure 3: Residual block and network used in this paper

Algorithm 1 Object Ranking

procedure RANKHYPOTHESES
  $ClassScores[1..class] = [0, 0 \,..\, 0]$
  $BoundingBoxes \leftarrow SSD(inputImage)$
  $CroppedImages \leftarrow crop(BoundingBoxes)$
  For i *in* $CroppedImages$:
    $scores[1..class] \leftarrow ResNet(i)$
    For j *in* $1..class$:
      if $scores[j] > ClassScores[j]$:
        $ClassScores[j] \leftarrow scores[j]$
  $ClassRanking \leftarrow sort(ClassScores)$

In an offline phase, we acquired, with a fixed camera, multiple templates per object under varying poses instead of generating them from CAD models. We also did perform some data augmentation to increase the number of templates per object. In our case, we limited ourselves to scaling. For each template in our training dataset, we added a smaller (90%) and a bigger (110%) version. This was deemed sufficient, as our experimental setup mimicked a robotic work cell, where objects are assumed to be placed on a flat surface at a fixed, known distance from the camera.

At runtime, objects are matched starting with the better ranked objects coming from our pre-processing. If no template from an object has a higher score than a given threshold, the next object on the list is tried. This process is repeated until the matching score of a template is higher than the threshold τ_{score}. In this case, this template is then considered as matched and the template matching algorithm is stopped. It is important to note that we used less templates (few dozens to few hundreds) than the original implementation of [12] (few thousands). Also, we used larger image resolutions (980x720 instead of 640x480 pixels).

4. Experiments

In this Section, we describe the experiments we performed to test our method. In particular, we detail the training and testing datasets used, motivating our choice of acquiring our own datasets. We then present results, in the form of speed up (ranking of objects) and F1-score classification results.

4.1. Datasets

4.1.1 In-house dataset

In order to validate our approach, we acquired our own training and test datasets, much in the same way that an operator would provide training examples during the configuration procedure of a robotic work cell. This stays in line with the concept that non-experts should be able to program such systems in an intuitive manner. Moreover, it complies with the situation where there are no a priori knowledge about objects, such as 3D models. Another reason for gathering our own datasets is that the majority of image datasets available in computer vision contest represents natural objects with significant intra-class variation, shown over complex background. These are not representative of the low intra-class variation typical of industrial objects, which are also generally placed on surfaces with uniform backgrounds. For our image datasets creation, we selected 10 objects that could be present in typical industrial robotic grasping applications. Care was taken to select them so that the majority had low-texture and/or are made of polished metal. The latter tend to have widely-varying appearance from viewpoints, due to specular reflections.

For the acquisition of the first dataset D_{green} (used for training purposes), our setup consisted in a fixed camera placed over a table, on which lied a green-colored cardboard. The cardboard allowed to easily segment objects from the background. For each object, a total of 50 images were captured, while being manually placed in random poses (translation and in-plane rotations). The goal was to try to capture the impact of the lighting on the appearance of the object. For instance in Fig. 4, one can see that the metallic part has one side slightly darker in appearance than the other (image with the green background on the left), while its surface has a more similar appearance in a different orientation (right image with the fake-wood background). For each image, ground truth images were created by our object segmentation method, which extracted image regions lying inside of the automatically generated bounding boxes. After this, a manual verification was done on all ground truth images to remove those for which the automatic segmentation failed. Specific training procedures were taken for `LINE-2D` and our ResNet, which are detailed in Sections 3.4 and 3.2.2, respectively.

The D_{other} dataset, used mainly for testing purposes,

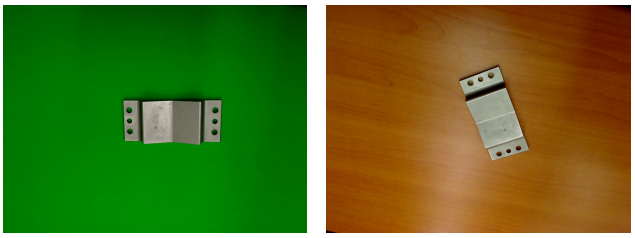


Figure 4: Different backgrounds used for our experiments: D_{green} (left) and test D_{other} (right) datasets

consisted of 20 images per object, for a total of 200 images. Like for the D_{green} dataset, objects were manually placed in random poses. The main difference is that objects were placed on a different background, seen in Fig. 4, namely glossy fake wood. It provided for extra challenge, as it is a different background than D_{green}, with texture and reflection of the wood. We have decided to use a different background for this other dataset for a simple reason: we wanted to estimate the impact of the change of background on the results. If we take a real scenario of an industrial user wanting to use many robotic grasping stations for multiple objects, it would be ideal for them to have a training station where they acquire the necessary information and then share this knowledge with every other grasping station. For small industrial robots, it is very likely that each grasping station will have different backgrounds. Therefore, this dataset will allow us to partially evaluate the performance of our approach for such cases.

4.1.2 Pose Dataset

We have also tested our method on the Pose dataset from [30]. It consists of 15 objects, for which half the images are objects that are rendered on a black background, and the other half on a cluttered background. There are around 700 images for both backgrounds, to offer a visual sampling of each object at regular angle intervals. We used the black background images along as our training dataset, as it bears similarities with our D_{green} datasets (uniform backgrounds). Additionally, we have randomly selected around a hundred images from the images with a cluttered background to the training set of our ResNet to avoid overfitting. We have also randomly selected another hundred images from the remaining images with a cluttered background as our test dataset. Fig. 5 shows an overview of the objects in the dataset and an example of an object rendered on a cluttered background.

4.2. Results

In here, we evaluated what we consider two advantages of our approach. First, we wanted to establish the quality of object template matching re-ranking, which directly translate into a speed up of matches. This is done in Section 4.2.1 by measuring the average rank of the ground truth object in the sorted output of our ResNet classifier. In Section 4.2.2, we evaluate the increase in matching reliability that the re-ranking inherently brings, as better matches are performed on average first.

Figure 5: Objects from the pose dataset [30]. The image on the left shows a number of its objects, rendered on a uniform background. On the right, an example of an object rendered on a cluttered background.

4.2.1 Average Ranking

We evaluated the ranking performance of our approach on the 2 datasets described in sections 4.1, namely our own acquired dataset D_1 and the pose dataset D_2.

We evaluated on our own dataset D_1 for two scenarios, which only differed slightly by the training dataset D_{train}. For the first experiment, the training dataset contained strictly images from the uniform green background, i.e. $D_{train} = D_{green}$. This was done in order to confirm the risk of overfitting with ResNet, if proper care is not taken when building training datasets. In the second experiment, a single image I_j per object is randomly pick from the D_{other} dataset containing the wood background, and added to our training dataset , i.e. $D_{train} = D_{green} + I_j$, with $I_j \in D_{other}$. For the first case, experiments reported an average rank of 4.05 over 10 objects on the testing dataset D_{other}, which is barely above randomness (the average rank would be 5 for an approach that uses a random order). This simply demonstrated, unsurprisingly, the overfitting of our ResNet, which has learned objects only when they were on a colored and textureless background. It then cannot generalize to objects shown on the textured background. More surprising was the fact that adding a single image from D_{other} was sufficient to nearly completely overcome this overfitting issue. Indeed, the average rank for this newly trained network was 1.37 over $D_{test} = \{D_{other} \setminus I_j\}$, very close to the results obtained directly on the training set (1.13). It therefore implies that for a real life scenario, full training for new background is not probably necessary, as only a few new images suffice to generalize the learning.

On the pose dataset D_2, we did not compute the average

ranking on its training dataset consisting of the images with a black background $D_{train} = D_{black}$, since we already confirmed the risk of overfitting with dataset D_1. For the experiment on this dataset, we randomly picked 100 images $\{I_j\}$ of the cluttered background D_{other}, which is a pretty conservative number considering our results with dataset D_1, and added them to our training set, $D_{train} = D_{black} + \{I_j\}$, with $\{I_j\} \in D_{other}$. The results for the average rank of the different scenarios are recapitulated in Table 1.

Training Set	Avg. Rank (D_1) (over 10 objects)	Avg. Rank (D_2) (over 15 objects)
D_{train}	4.05	-
$D_{train} + \{I_j\}$, with $\{I_j\} \in D_{other}$	1.37	1.99

Table 1: Average ranking of objects

4.2.2 F_1 Score

In this section, we present the results of evaluating the performance of `LINE-2D` template matching, with and without our pre-processing approach. We used the multi-class F_1 score defined in [27] to do the evaluation. The score can be calculated as follows:

$$\mathrm{F_1score}_\mu = \frac{2 \times \mathrm{Precision}_\mu \times \mathrm{Recall}_\mu}{\mathrm{Precision}_\mu + \mathrm{Recall}_\mu} \tag{2}$$

$$\mathrm{Precision}_\mu = \frac{\sum_{i=1}^{x} tp_i}{\sum_{i=1}^{x} tp_i + fp_i} \tag{3}$$

$$\mathrm{Recall}_\mu = \frac{\sum_{i=1}^{x} tp_i}{\sum_{i=1}^{x} tp_i + fn_i} \tag{4}$$

These last equations are valid to measure classification performance for classes C_i. The variables tp_i, fp_i, tn_i and fn_i are respectively for true positives, false positives, true negatives and false negatives, all for class C_i.

Scores are shown in Fig. 6 on our dataset D_1 and in Fig. 7 for the pose dataset D_2. These graphs show the performance of `LINE-2D` for 3 different methods: our pre-processing method, the best score and random order. For a description of our method, see section 3. For the best score approach, all templates are processed and the template with the highest score is considered a match. For the random order approach, a random order of objects is generated and all the templates of that particular object are processed. It is repeated until a template of the object gets a score higher than the selected threshold. In these graphs, we can see that our method clearly outperforms randomness. The effect of using our ResNet for classification is shown for low thresholds, where the score represents the classification performance of our ResNet. We therefore show that our method greatly mitigates the negative impact brought on by electing a bad threshold τ_{score}. Compared to the best score approach, we can see that our method performs slightly worse for lower thresholds, but gets slightly better performances as the threshold is increased. The big difference between these 2 methods is that our method performs the matching for only a few objects while the best score approach needs to process all of them.

5. Conclusion

In this paper, we proposed a generic and modular template matching pre-processing method which ranks objects by using Deep Learning for object detection and recognition. We have shown that our pre-ranking can speed up the matching of templates while increasing its reliability (in case of first-match search). More precisely, our method was able to reduce the number of objects that needed to be processed by template matching from an average of 5 objects down to 1.37 for our own dataset and from an average of 7.5 to 1.99 for the Pose dataset. This indicates that we would get near constant-time computation, for small to moderately sized databases of objects (up to 1,000). Moreover, F_1 scores indicate that we are competitive with a brute-force template matching approach, while being significantly faster.

With this project, we also hope to ease the transition of Deep Learning into industries by pre-processing a popular method instead of proposing a black box method, which is hard to debug when it fails. In order to do so, further testing in real environments and with larger object sets are needed.

References

[1] A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L. Di Stefano, and M. Vincze. Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2104–2111. IEEE, 2013.

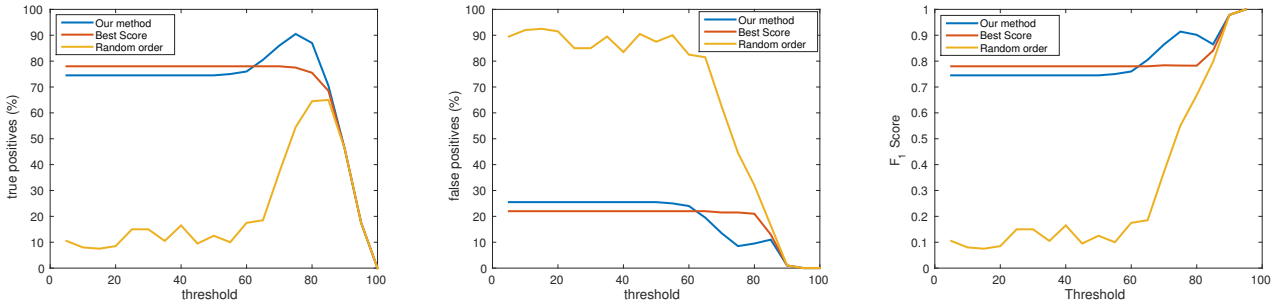
[2] N. Alt, S. Hinterstoisser, and N. Navab. Rapid selection of reliable templates for visual tracking. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1355–1362. IEEE, 2010.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[4] G. Bradski et al. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

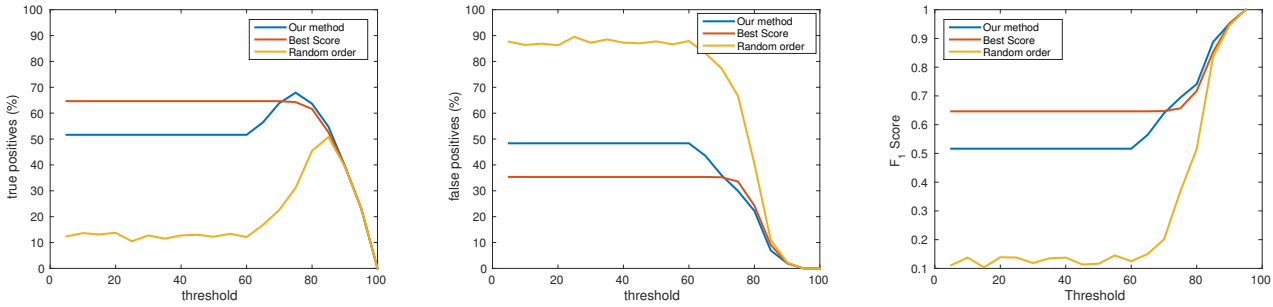
[5] A. Crivellaro, M. Rad, Y. Verdie, K. Moo Yi, P. Fua, and V. Lepetit. A novel representation of parts for accurate 3d object detection and tracking in monocular images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4391–4399, 2015.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database.



(a) True Positives (b) False Positives (c) F_1 Score

Figure 6: Different performance indicators with respect to template matching thresholds on our own acquired dataset D_1



(a) True Positives (b) False Positives (c) F_1 Score

Figure 7: Different performance indicators with respect to template matching thresholds on the pose dataset D_2

In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[7] D. Fox. Deep learning for tracking and intuitive physics. In *Robotics: Science and Systems Workshop, Are the Sceptics Right? Limits and Potentials of Deep Learning in Robotics*, 2016.

[8] K. Fu. *Sequential methods in pattern recognition and machine learning*, volume 52. Academic press, 1968.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

[12] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):876–888, 2012.

[13] S. Hinterstoisser, V. Lepetit, S. Ilic, P. Fua, and N. Navab. Dominant orientation templates for real-time detection of texture-less objects. In *CVPR*, volume 10, pages 2257–2264, 2010.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[15] W. Kehl, F. Tombari, N. Navab, S. Ilic, and V. Lepetit. Hashmod: A hashing method for scalable 3d object detection. *CoRR*, abs/1607.06062, 2016.

[16] S. Korman, D. Reichman, G. Tsur, and S. Avidan. Fast-match: Fast affine template matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2331–2338, 2013.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[18] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *I. J. Robotics Res.*, 34(4-5):705–724, 2015.

[19] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. Ssd: Single shot multibox detector. *arXiv preprint arXiv:1512.02325*, 2015.

[21] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[22] S. Mattoccia, F. Tombari, and L. Di Stefano. Fast full-search equivalent template matching by enhanced bounded correlation. *IEEE transactions on image processing*, 17(4):528–538, 2008.

[23] A. Penate-Sanchez, L. Porzi, and F. Moreno-Noguer. Matchability prediction for full-search template matching algorithms. In *3D Vision (3DV), 2015 International Conference on*, pages 353–361. IEEE, 2015.

[24] R. Rios-Cabrera and T. Tuytelaars. Discriminatively trained templates for 3d object detection: A real time scalable approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2048–2055, 2013.

[25] A. Shah, E. Kadam, H. Shah, and S. Shinde. Deep residual networks with exponential linear unit. *arXiv preprint arXiv:1604.04112*, 2016.

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[27] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[28] F. Tombari, S. Mattoccia, and L. Di Stefano. Full-search-equivalent pattern matching with incremental dissimilarity approximations. *IEEE transactions on pattern analysis and machine intelligence*, 31(1):129–141, 2009.

[29] L. Trottier, P. Giguère, and B. Chaib-draa. Parametric exponential linear unit for deep convolutional neural networks. *arXiv preprint arXiv:1605.09332*, 2016.

[30] F. Viksten, P.-E. Forssén, B. Johansson, and A. Moe. Comparison of local image descriptors for full 6 degree-of-freedom pose estimation. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2779–2786. IEEE, 2009.

[31] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, may 2004.

[32] P. Wohlhart and V. Lepetit. Learning descriptors for object recognition and 3d pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3109–3118, 2015.